Computation in Mind and World

A realist account of computation in cognitive science

Mark Daniel Sprevak
Trinity College

13 December 2005

This dissertation is submitted for the degree of Doctor of Philosophy
Dept. History and Philosophy of Science, University of Cambridge

To my parents

Contents

Declaration **iv**

Abstract **v**

Acknowledgements **vi**

Introduction **viii**

1 Chinese rooms and program portability **1**
1.1 Searle's argument 1
1.1.1 Qualified versions of Searle's argument 4
1.1.2 Standard replies 5
1.1.3 The Church–Turing thesis 8
1.2 Criticism of Searle's argument 9
1.2.1 Differences in architecture 11
1.2.2 Summary 12
1.3 Objections and replies 12
1.3.1 The virtual brain machine objection 12
1.3.2 The man with a brain objection 14
1.3.3 The syntax/physics objection 15
1.3.4 The abstraction objection 17
1.3.5 The 'not guilty' objection 19
1.3.6 The 'same conclusion' objection 20
1.3.7 The 'unnecessary baggage' objection 21
1.3.8 The Chinese gym objection 23
1.3.9 The syntax/semantics objection 24
1.3.10 Turing's definition of algorithm 25
1.4 Conclusion 29

2 The problem generalised **31**
2.1 Programs and computation 31

2.2 The threat of anti-realism 33
2.2.1 Searle's argument 37
2.2.2 Putnam's argument 40
2.2.3 Kripke's argument 43
2.3 Existing solutions 45
2.3.1 Syntactic approaches 46
2.3.2 Z and specification languages 46
2.3.3 Canonical architectures 47
2.3.4 Chalmers' realist approach 49
2.3.5 Copeland's realist approach 55
2.3.6 Mellor's realist approach 60
2.4 Conclusion 64

3 Preliminaries **65**
3.1 The discourse to be formalised 65
3.1.1 The computational theory of mind 65
3.1.2 Two key intuitions 71
3.1.3 Outline of problem and solution 76
3.2 Preliminary definitions 78
3.2.1 The relata of representation 78
3.2.2 Representation tokens 78
3.2.3 The representation relation 80
3.2.4 Representational contents 82
3.2.5 Determinate numerical identity relations 83
3.2.6 Non-actual tokens and contents 85
3.2.7 Processes 87
3.2.8 Extreme positions 93

4 The PR-model **95**
4.1 Representational processes 95
4.2 Unary computational processes 96
4.2.1 Spanning 97
4.2.2 The representation requirement 101
4.2.3 Simple processes 104
4.2.4 Structured representation 114
4.3 General computational processes 121
4.3.1 General representational processes 125
4.3.2 General computational processes 126
4.4 Conclusion 132

5 Applications **133**
5.1 Examples of the PR-model 133
5.1.1 Synchronous computation 133
5.1.2 Clerk-style computation 134
5.1.3 An adder 135
5.1.4 Connectionist computers 137
5.1.5 Stored program architectures 140
5.1.6 Memory, state, and constants 141
5.1.7 Recursion-based computation 144
5.1.8 Higher-level computational systems 145
5.1.9 Distributed computation 146
5.2 The semantics of computation talk 147
5.2.1 When a computation is performed 148
5.2.2 When two computations are the same 148
5.3 Pragmatics of computation talk 154

6 Metaphysics **157**
6.1 The terms of the PR-model 157
6.1.1 Identity 157
6.1.2 Counterfactual dependence 158
6.1.3 Explanation 159
6.1.4 Representation 161
6.1.5 Other notions 164
6.2 Conclusion 164

Conclusion **165**

A Representational subprocesses **170**

B The joining operator **173**
B.1 In and out functions 174
B.2 Unattached inputs and outputs 176
B.3 Time delays 176

Bibliography **180**

Declaration

1. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

2. This dissertation is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other university.

3. This dissertation does not exceed the limit of $80,000$ words, including footnotes and excluding bibliography.

I, Mark Daniel Sprevak, make these declarations in accordance with the requirements published by the Board of Graduate Studies of the University of Cambridge and place my signature below.

Abstract

This Ph.D. thesis concerns the truthmakers of the computational theory of mind (CTM). The CTM is the view that some, or all, of our cognitive processes are computations. A potential problem with the CTM is that it is not clear what it means for a system, such as the human brain, to perform a computation. This problem is often glossed over in the literature—computation is assumed to be just like what electronic computers do. Unfortunately, what electronic computers do is far from clear. My thesis addresses this concern. Specifically, it answers the question: under what conditions does a real-world system, such as a human brain or electronic computer, perform one computation rather than another? Searle (1992), Putnam (1988), and Kripke (1982) have argued that the only possible answers to this question are anti-realist. On their view, the computation that a system performs is a function of how human agents interpret that system, rather than of the mind-independent world. If this were true, then the consequences for cognitive science would be dire. Cognitive science would not explain cognition, it would presuppose cognition. In my thesis, I defend cognitive science from these anti-realist attacks. I argue that mind-independent facts can determine the computation that a system performs. My argument consists of two parts. First, I develop a semantics for computation talk in the context of cognitive science. This provides an account of the semantic content of our computational claims in that context. Then, I present an account of the metaphysical facts that make that semantic content true or false. I argue that these metaphysical facts are, at least possibly, mind-independent. The resulting notion of computation is compared to the alternatives of Chalmers (1996), Copeland (1996), and Mellor (1991a).

Acknowledgements

I would like to thank a number of people who have made helpful comments on earlier drafts of this work, including Arif Ahmed, Mike Beaton, David Chalmers, Ron Chrisley, Jack Copeland, Julien Deonna, John Hyman, Chris Jones, Tim Lewens, Neil Manson, Ben Morrison, Erica Roedder, Luc Schneider, John Searle, and two anonymous referees. I have benefitted greatly from informal comments and remarks by Stuart Ferguson, Carl Gillett, Laurie Paul, and Hugh Mellor. I have also benefitted from lively discussions with Alex Broadbent, Daniel Elstein, Axel Gelfert, Arash Pessian, Richard Price, and Christina McLeish.

Sections from this work have been presented at conferences and seminars at the University of Oxford, University of Cambridge, Columbia University, New York University, University of Sussex, and University of Geneva. I would like to thank the AHRC for providing funding for the first two years of this project, and Trinity College for providing funding for the final year. I would also like to thank the Department of History and Philosophy of Science for making the execution of this project easier and more enjoyable.

I have benefitted greatly from having an exemplary supervisor, Peter Lipton, and advisor, Martin Kusch. I owe an enormous intellectual debt to them. Not only have they been generous with their time and made detailed and rigorous comments on the drafts of this thesis, in a thousand ways they have provided a general model of how to do philosophy. I cannot imagine a more pleasurable way in which to learn.

Many people have helped me write this thesis by providing support (not always of a philosophical nature) in a friendly setting. I would like to thank Andreas, Matthew, Mukund, Helen, Serena, and Paul. Special thanks are due to Michela who provided support in many difficult times. Above all, I would like to thank my parents, without whom nothing would be possible, and to whom, with pleasure, I dedicate this thesis.

On the one hand, we have a very elegant set of mathematical results ranging from Turing's theorem to Church's thesis to recursive function theory. On the other hand, we have an impressive set of electronic devices that we use every day. Since we have such advanced mathematics and such good electronics, we assume that somehow somebody must have done the basic philosophical work of connecting the mathematics to the electronics. But as far as I can tell, that is not the case. On the contrary, we are in a peculiar situation where there is little theoretical agreement on such absolutely fundamental questions as, What exactly is a digital computer? What exactly is a symbol? What exactly is an algorithm? What exactly is a computational process? Under what physical conditions exactly are two systems implementing the same program?

(Searle, 1992, 205)

Introduction

Consider how one might multiply 12 by 10. There are a number of ways in which one might proceed. For example, one might write down the numeral '12' and place a '0' to its right, producing the numeral '120'. Or one might add ten twelves in a single sum: '12 + . . . + 12 = 120'. Or one might add ten twelves in eleven separate sums: '12 + 12 = 24', '24 + 12 = 36', '36 + 12 = 48', . . . , '108 + 12 = 120'. Or one might multiply 10 by 2 and add the result to 10 by 10: '$10 \times 2 = 20$', '$10 \times 10 = 100$', '100 + 20 = 120'. Or one might use a prepared multiplication table, matching the numerals '10' and '12' with the appropriate column and row and reading off the result.

These are only a few of the ways of computing 12 times 10. This thesis concerns the conditions under which a system performs one computation rather than another. In terms of the example above: what determines the way in which a system computes 12 times 10? When do two systems compute 12 times 10 in the same way? Can an electronic machine compute 12 times 10 in the same way as a human being?

These questions do not have obvious answers. As Searle (1992) remarks in the opening quotation, we have advanced mathematics and sophisticated electronics, but remarkably little in the way of philosophical theory on how the two relate. This lacuna is particularly evident in the study of the mind. There are currently two distinct traditions in the study of the mind: a physiological tradition focusing on the physical makeup of brains, and a cognitive tradition focusing on cognition and the computations supposed to constitute cognitive processes. It is not obvious how the two traditions relate. Discussions in cognitive science tend to gloss over this issue. Typically, some remarks are made about 0's and 1's, and the discussion quickly moves on to describe the failures and achievements of particular models. This misses an important point. As Searle (1992) says, a crucial question for any model is: 'How *exactly* does the model relate to the reality being modelled?' (p. 199).

An account of what it means to perform a computation is not just desirable for cognitive science, it is essential. Cognitive science has come under a number

of attacks in recent years. Instead of questioning particular cognitive science theories, these attacks question the discipline's entire foundation. Three of the best articulated attacks have come from Searle (1992), Putnam (1988), and Kripke (1982). The thrust of their attacks is that the computation that a system performs is not a mind-independent feature of that system. Computation is invariably sensitive to, and requires, an interpretative agent.

If this were correct, then the consequences for cognitive science would be serious. An expected pay-off from cognitive science is an explanation of mental life in non-mental terms. If Searle, Putnam, and Kripke are right then this pay-off is undeliverable: instead of explaining mentality, cognitive science presupposes mentality from the start. The concept of computation used by cognitive science is not an objective feature of the world, but reflects a feature of our own minds. Anti-realism about cognitive science ensues. In this thesis, I defend cognitive science from these anti-realist attacks. I put forward a realist account of computation that aims to put cognitive science on a secure and transparent foundation.

In what follows, the systems that perform computations are sometimes described as 'physical', but nothing turns on this. If non-physical systems can perform computations, then the same concerns apply equally to those systems. Alternatively, if all systems are physical then the qualifier 'physical' can be dropped as unnecessary. The term 'physical' is used only to emphasise that it is with the implementation of computation in real-world systems that this thesis is concerned. In what follows also, for the sake of brevity, the implementation of a computation in a real-world system is sometimes referred to simply as 'a computation'. Again, nothing turns on this. I do not wish to deny that the notion of computation has other strands, or claim that the notion of implementation in real-world systems is somehow primary or fundamental. If another sense of computation is intended—for example, computation as an abstract mathematical object—then that will be explicitly indicated in the text.

Cognitive science

Many cognitive processes appear to be mysterious and complex. It is not only unclear how they work, it is unclear how they are even possible. Cognitive science aims to answer these questions. The central claim of cognitive science is that cognitive processes are computations. According to cognitive science, we have certain cognitive processes because our brains perform distinctive computations. The task of cognitive science is, roughly speaking, to describe what those computations are.

Cognitive processes that have been explained in terms of computation in-

clude syntax parsing, recognition of simple shapes, rational choice, and deductive inference. According to the cognitive science theories of these domains, it is because our brains perform certain computations that we parse sentences, recognise simple shapes, make rational decisions, and so on. Notable successes of this approach include Chomsky's theory of syntax and Marr's theory of vision.[1] The computational approach remains today the dominant view of how the mind works.

Reliance on the notion of computation has both benefits and risks attached. The benefit is that it provides a way of explaining how cognitive processes are possible and how they work. Cognitive processes are possible because computation is possible. Cognitive processes are just like what goes on inside an electronic computer, and what goes on inside an electronic computer is clearly possible. How cognitive processes work is explained by the particular cognitive science theory in question. That theory provides a computation, or a sketch of a computation, that describes how the cognitive process works. Cognitive processes are explained by their computational structure.

The risk attached to the computational approach is that cognitive science is made hostage to fortunes of the notion of computation. If the notion of computation turns out to be problematic, or trivial, or interest-relative, then cognitive science is in trouble. This should be a cause for worry for advocates of cognitive science, since the nature of our notion of computation in these respects is not obvious. It is conceivable, and anti-realists have argued, that the notion of computation contains entailments that an advocate of cognitive science would be reluctant to accept. Note that such debates cannot be resolved simply by pointing to the widespread use of computation in electrical engineering and the computer industry. As we shall see, it is far from obvious what it means for an electronic PC to perform a computation either. In general, it is not obvious what makes it true that a system perform one computation rather than another, or even that it perform a computation at all.

A number qualifications should be added.

First, not everyone would agree with the characterisation of cognitive science given above. The central tenets of the discipline are still a matter of dispute. Nevertheless, many philosophers and cognitive scientists have argued that cognitive science should be characterised by the claim that cognition is computation.[2] I will work with this characterisation in this thesis. If one

[1] See Chomsky (1980); Marr (1982). Both theories have been substantially revised; for recent views, see Chomsky (1995); Wandell (1995). It is a matter of controversy how Chomsky's theory of syntax relates to how we actually parse sentences. Fodor (1975); Fodor et al. (1974) provide a computational interpretation of syntax parsing based on Chomsky's account; see Rey (2003) for a discussion of Chomsky's own views.

[2] For example, see Fodor (2000), 3–4; Searle (1992), 197–198; Pylyshyn (1984), xv; Johnson-Laird (1983), 8–9; Cummins and Cummins (2000), 6; Haugeland (1981), 31; Lachman et al. (1979), 107.

does not wish to call this view 'cognitive science', then another name can be substituted in its place.

Second, it is an open question whether the claims of cognitive science should be interpreted as literal statements of fact, or as metaphors and useful heuristics. In this thesis, I again follow the existing philosophical literature and interpret the computational claims of cognitive science literally.[3] This literal reading of cognitive science is perhaps the most interesting to defend. If it can be defended—and I shall argue that it can—then there seems little reason to retreat to a metaphorical reading.

Third, cognitive science is not committed to the claim that computation is the whole story about the mind. Cognitive science makes claims about particular cognitive processes, e.g. syntax parsing, shape recognition, deductive inference. The truth of these particular claims is compatible with other aspects of the mind, including other cognitive processes, being non-computational. Indeed, it may turn out that only relatively minor aspects of the mind are computational.

In what follows, the characterisation of cognitive science given above—the claim that some cognitive processes literally are computations—shall be called the 'computational theory of mind' (CTM).

Three anti-realist challenges

Three challenges to the realist status of computation are considered in this thesis.

The first challenge comes from Searle (1992). Searle claims that 'syntax is not intrinsic to physics'.[4] By this, Searle means the performance of a computation is not determined by the mind-independent world. A system only performs a computation because we, as interpreting agents, interpret it in a certain way. Interpretation enters into the performance of a computation in two ways. First, the performance of a computation cannot be a matter of the system having a certain pattern of mind-independent activity. Suppose that this is false, e.g. suppose that a desktop computer runs a particular program, say Microsoft Word, because a certain pattern of activity takes place inside it. Searle points out that the world is full of patterns of activity. Even a brick wall has patterns of activity: vibrational activity, thermal activity, atoms changing state, and so on. There is *so much* activity inside a brick wall that, argues Searle, there is almost certain to be a pattern identical to that inside a desktop computer. Therefore, as far as the mind-independent facts are concerned, a brick wall runs Microsoft Word. This is clearly absurd. The performance of a computation

[3]For advocacy of a literal reading, see Fodor (2000), 3–4; Pylyshyn (1984), xiv–xv; Jackendoff (1987), 17; Johnson-Laird (1988), 34–35; Newell and Simon (1976).

[4]Searle (1992), 207.

must have extra conditions attached. Searle argues these extra conditions involve the beliefs, interests, and attitudes of observers. The second way in which interpretation enters into the performance of a computation is in the assignment of computational states to physical states. Searle claims that there is nothing intrinsic to a particular physical state that makes it a '0' or '1'. We, as interpreting agents, must interpret it as a '0' or '1'. Therefore, the computational identity of a system depends on the interpretations of observers.

The second challenge comes from Putnam (1988). Putnam gives a formal proof, based on the assumptions of classical physics, that every open system, considered in mind-independent terms, implements every finite state automaton. The proof is specific to the implementation of finite state automata, but Putnam claims that a similar result holds for any other computational architecture. Therefore, every open system, considered in mind-independent terms, performs every computation. This is bad news for cognitive science because cognitive science tries to explain how cognitive processes work in terms of the performance of *distinctive* computations. If Putnam's result is correct, then there are no distinctive computations that the brain performs. The brain, like everything else, performs every computation. Notably, Putnam's result leaves open the possibility of mind-dependent distinctive facts about computation. Although in terms of the mind-independent facts every system performs every computation, if mind-dependent facts—beliefs, interests, attitudes, and so on—are included, then there perhaps can be distinctive facts about computation. One assumes that this is how Putnam thinks computation should be understood. Putnam is an anti-realist about all claims, so it is no surprise that he is an anti-realist about computation claims too.

The third challenge comes from Kripke (1982). Kripke presents a sceptical argument concerning rule-following and meaning. As part of his argument, Kripke considers the case of machine rule-following. Computation and rule-following are closely connected topics: in order for a machine to perform a computation that machine must follow a rule. Kripke considers what fact about a machine determines that it follows one rule, i.e. performs one computation, rather than another. His answer comes in two parts. First, he claims, the computation that a given system performs cannot be fixed by the mind-independent facts: an interpreting agent is needed for a system to perform a computation. Kripke gives three arguments for this claim. An interpreting agent is needed in order to: (i) interpret the inputs and outputs of the machine, (ii) extend the machine's finite actual behaviour to infinite possible behaviour, and (iii) make the distinction between correct and incorrect functioning of the machine. If there are facts about computation at all, then those facts are mind-dependent. In the second part, Kripke goes on to argue that there are no

mind-dependent facts about rule-following either. Therefore, according to his argument, there are no facts about rule-following at all, either for humans or computers. However, we shall only be concerned with the first step of Kripke's argument. Kripke's second step has received attention elsewhere.[5] We shall restrict attention here to Kripke's specific claim about machines: his claim that the only way for a machine to follow a rule is for us to interpret it as doing so.

Searle, Putnam, and Kripke all conclude, for various reasons, that the facts about which computation a system performs, or whether a system performs a computation at all, are mind-dependent. In this thesis, I argue that such anti-realism can be resisted. A realist notion of computation is available.

The positive position

This thesis puts forward a realist position on computation. The availability of a realist account of computation depends on whether the facts that determine which computation a system performs are mind-dependent or mind-independent. If those facts can be mind-independent, then realism about computation is true. If those facts must be mind-dependent, then realism about computation is false. The primary concern of this thesis is therefore: what kind of metaphysical facts constitute a system performing one computation rather than another?

Metaphysical questions like this cannot be answered in a single step. Before arriving at an answer, it is necessary to have a clear view of the semantics of the relevant area. We must be clear on what we mean by our computation claims before we can give an account of the facts that make them true or false. The situation can be phrased in the following way. Whether a particular system performs a computation depends on two factors: (1) on what we mean by 'performs a computation'; and (2) on the way the world is. The first component concerns the semantics of our computation talk. The second component concerns the metaphysics, the truthmakers, of that talk. A potential problem facing the project described above is that the second component cannot be addressed without presupposing an account of the first.

The aim of this thesis is to give an account of the metaphysical component. I am interested in whether the facts that make computation claims true or false are mind-dependent or mind-independent. Searle, Putnam, and Kripke argue that those facts must be mind-dependent. I argue that they can be mind-independent. It is worth noting that there is no argument over whether the facts underlying the *semantic* component are mind-dependent or mind-independent. Both realists and anti-realists about computation can agree that what we mean

[5] For example, see Miller and Wright (2002) and references.

by our words is mind-dependent (although they may disagree about whether all representations are mind-dependent). The string of letters 'computation' means *computation* because of the way in which we interpret those symbols in our linguistic community. In another community, with different beliefs and attitudes, the string of letters 'computation' might not mean anything at all. The semantic content of our words is clearly mind-dependent. The distinctive claim that the realist about computation makes is that the second, metaphysical, component is mind-independent.

As noted above, the metaphysical and semantic projects are inter-related. The nature of the metaphysical component cannot be settled without some view on the contribution of the semantic component. Therefore, it would not be satisfactory if this thesis were to attempt to answer the metaphysical question without providing some justified account of the semantic component. The strategy that I follow in this thesis is to start by giving an account of the semantic component, and then tackle the metaphysical component. Metaphysics can of course inform semantics, but at least initially it seems reasonable to start with a *prima facie* semantics. The order of this two-step approach is not uncommon. It is the way that Lewis deals with counterfactuals: first he provides a semantic account of the content of counterfactual talk (Lewis, 1973), then he provides an account of the metaphysics that make that content true or false (Lewis, 1986b). One of the clearest exponents of the two-step approach is Hugh Mellor, as demonstrated in his analyses of properties and dispositions (Mellor, 1991b, 2000).

Let us consider each part of the two-step approach.

The semantic part gives an account of what we mean by our computation talk. The semantic account put forward in this thesis is based on two key intuitions. The first intuition is that computation involves representation. The second intuition is that computations are made up of parts, and that two computations are the same just in case they have the same parts and those parts are connected in the same ways. The work of formalising these intuitions takes up most of Chapters 3 and 4. The resulting formal semantic model is called the *process and representation model* (PR-model). The PR-model specifies what we mean when we say that a given system performs a particular computation. In other words, it specifies the semantic content of our computation claims. Once this semantic model is in place, we are then in a position to evaluate what kinds of metaphysical facts can make that content true or false.

The PR-model produces statements of the following form: 'System S performs computation C' is true iff conditions X, Y, Z obtain. The metaphysical part of the project gives an account of the facts that make conditions X, Y, Z obtain. In particular, it considers whether those facts can be mind-independent, or

whether they must be mind-dependent. The realist about computation claims that they can be mind-independent, the anti-realist claims they must be mind-dependent. The PR-model specifies X, Y, Z in terms of standard metaphysical notions. These metaphysical notions are representation, numerical identity, and counterfactual dependence. Therefore, according to the PR-model, statements such as 'System S performs computation C' are true just in case a number of conditions concerning representation, numerical identity, and counterfactual dependence obtain. Therefore, the metaphysical facts that make computation claims true *are just facts about representation, numerical identity, and counterfactual dependence*. If one can be a realist about these metaphysical notions, then one can be a realist about computation.

This thesis does not specifically argue for realism about representation, numerical identity, or counterfactual dependence. That has been argued for at length elsewhere.[6] The purpose of this thesis is to establish the conditional claim that *if* facts about representation, numerical identity, and counterfactual dependence are mind-independent, *then* facts about computation are mind-independent too. Many philosophers have defended realism about these antecedent notions for reasons independent of any concerns about computation or cognitive science. If one is willing to accept these defences—and it is not clear that they are wrong—then realism about computation can be had at no extra cost. (The argument is, of course, symmetrical: if realism about these antecedent notions is impossible, then according to the account of computation in this thesis, realism about computation is impossible too).

Outline of thesis

The problem that this thesis aims to address is what makes a system perform one computation rather than another. The first two chapters of the thesis introduce the problem. The remaining four chapters provide a solution.

Chapter 1 introduces the problem via the example of Searle's Chinese room argument. I argue that Searle's Chinese room argument fails because it assumes that the Chinese room can run any program. The Chinese room cannot run any program. The programs that a system can run depend on that system's computational architecture. This raises part of the main question of this thesis: what determines a system's computational architecture?

Chapter 2 generalises the problem in two ways. First, the focus is widened from programs to computation in general. Second, it is shown that wider issues than the Chinese room argument depend on the nature of computational identity, including realism about cognitive science. The anti-realist arguments

[6] See references in the final section of this introduction.

of Searle, Putnam, and Kripke are introduced. An overview is given of existing solutions, including those of Chalmers (1996), Copeland (1996), and Mellor (1991a). I argue that none of these existing solutions is satisfactory.

Chapter 3 begins with a brief overview of how computation is used in cognitive science. I then introduce the two key intuitions involving computation: (i) computation involves representation, and (ii) computations are the sum of their parts. The PR-model aims to formalise these intuitions into a substantial account of computation. The rest of the chapter is devoted to the basic concepts of the PR-model and the kinds of metaphysical commitments they entail.

Chapter 4 defines the PR-model itself in a step-by-step way. The PR-model provides an account of what we mean by our computation talk in terms of the notions of identity, counterfactual dependence, representation, and explanation. The details of how these notions fit together to form the notion of computation are specified by the definitions given in this chapter.

Chapter 5 applies the PR-model to real-world examples of computation talk. It is shown how stored-program, connectionist, and other kinds of computation talk can be accommodated in the PR-model. In the second half of the chapter, an account, based on the PR-model, is given of what it means for two computations to be the same or different. Finally, some pragmatic features of computation talk are discussed.

Chapter 6 deals with the truthmakers of computation talk. I argue that these truthmakers can be mind-independent. The PR-model provides an account of what we mean in terms of identity, counterfactual dependence, representation, and so on. I describe pre-existing strategies for how one can be a realist about these notions, and thereby be a realist about computation.

Relation to the literature

There are a number of topics that I do not address in this thesis. I do not attempt to provide a new analysis of representation, counterfactual dependence, identity, or explanation. The purpose of this thesis is to reduce the notion of computation to these other notions, not to provide a new analysis of these notions in turn. The nature of representation, counterfactual dependence, identity, and explanation has already received considerable attention elsewhere.[7] It bears repeating that this thesis does not provide an account of representation

[7]See Lewis (1973, 1979, 1986b) and references for the nature of the counterfactual dependence relation. See Cummins (1989); Dretske (1981); Fodor (1990b); Haugeland (1990); Millikan (1986); Stich and Warfield (1994) and references for the nature of the representation relation. See Brody (1980); Noonan (1980, 1993); van Inwagen (1990); Wiggins (2001) and references for the nature of the identity relation. See Achinstein (1983); Boden (1988); Cummins (1983); Dennett (1971); Haugeland (1978); Hempel (1965); Neisser (1967); Pylyshyn (1984, 1999); Simon (1969) and references for the nature of explanation and explanation of cognitive processes.

or intentionality. I presuppose representation in order to give an account of computation. Representation has to be accounted for in some other way.

There are a number of other challenges to cognitive science that I do not consider in this thesis. These challenges include the claims that: (1) the function computed by certain mental processes is, in principle, not reproducible by a computer (Lucas, 1961; Penrose, 1989); (2) certain aspects of mentality involve a background that cannot be formalised in terms of symbols and computations (Dreyfus, 1992; Searle, 1992); (3) central cognitive processes, such as those involved in solving the frame problem, are not computational (Fodor, 1983, 2000); (4) non-computational accounts of cognition, such as a dynamical systems approach, an ecological approach, or a Gestalt approach, are in some cases preferable to a computational account (Gibson, 1979; van Gelder, 1995; Wertheimer, 1985); (5) consciousness, moods, skills, and understanding cannot be accounted for by computation (Haugeland, 1978, 1981).

These challenges are, at least in one sense, less troubling than the anti-realist challenge. This is because, for each of these challenges, the cognitive scientist can in the worst case concede the challenge without irreparable damage to her position. Cognitive science does not claim that all mental processes are computational, only that some are. Even if central cognitive processes, consciousness, and so on are not computational, that is no reason to think that cognitive science is not true of other mental processes, such as syntax parsing and shape recognition. The same concessive strategy cannot be used in reply to Searle, Putnam, and Kripke. Here the challenge threatens the status of all claims of cognitive science, not just the discipline's application to particular domains.

There are a number of topics addressed by this thesis that are sometimes erroneously grouped under the heading of challenges to cognitive science. These topics include connectionist approaches to cognition (Rumelhart et al., 1986; Smolensky, 1988) and extended cognition (Clark, 1997; Clark and Chalmers, 1998; Hutchins, 1995). I believe that, properly understood, these approaches are not challenges to cognitive science. They are compatible with the claim that cognition is computation. In Chapter 5, I show how these theories can be accommodated by a notion of computation.

For the purpose of this thesis, I am neutral on a number of key debates in cognitive science. I am neutral on whether the representations on which cognitive computations operate are sentence-like and directly map onto propositional attitudes (Fodor, 1975; Fodor and Pylyshyn, 1988; Harman, 1973; Pylyshyn, 1984), or whether those representations are of a different kind and have less direct relations to the notions of folk psychology (Churchland, 1981, 1986; Dennett, 1978b; Stich, 1983). I am neutral on whether the architecture of

the mind is classical or connectionist, whether folk psychology is true or false, whether the computational structures of the brain are innate or acquired, and whether, and to what extent, the mind is modular. These are all important questions, but they do not directly impact on the question of this thesis.

It is worth emphasising that my intention is not to argue that the CTM is true. Whether a particular computational claim is true or false is, I shall argue, an empirical matter. My intention is rather to show that computational claims *have empirical content*—to show that computational claims are made true or false by the world, not by our attitudes as interpreting agents. It is the possibility of realism about cognitive science theories that is defended by this thesis, not the truth of particular theories.

Finally, there are there are a number of topics that I would like to have addressed but did not have the time or space. These topics include: (1) showing that traditional functionalism cannot provide an account of the metaphysics of computation suitable for cognitive science; (2) an account of the epistemology of computation, i.e. how we know which computation a system is performing; (3) a discussion of how the notion of computation developed in this thesis relates to the notion of computation in mathematical logic; (4) a discussion of how the notion of computation developed in this thesis relates to Pylyshyn's (1984) notion of cognitive architecture; and (5) a discussion of how internalism/externalism about representation relates to internalism/externalism about computational identity. I hope to address these topics in future work.

Chapter 1

Chinese rooms and program portability

This chapter introduces the topic of this thesis using the example of Searle's Chinese room argument. I argue that Searle's Chinese room argument fails because it depends on the assumption that the Chinese room can run any program. This assumption is false, and it cannot be weakened in such a way as to save Searle's claim. A number of possible objections are considered. I argue that these objections fail. Responding to the objections introduces some of the main themes of this thesis.

1.1 Searle's argument

Searle's Chinese room argument is an argument against the possibility of Strong artificial intelligence (Strong AI). The thesis of Strong AI is that the running of a particular program is sufficient for, or constitutive of, mentality: it is simply in virtue of running a particular program that a system has mentality. One aspect of mentality is understanding, and this is the aspect on which Searle focuses. Searle appreciates that understanding is a complex notion, so he has a particular form of understanding in mind: the understanding of simple stories. He considers whether the Strong AI thesis holds in this case.

It seems intuitively obvious that when I read a simple story in English, I understand that story: somewhere in my head there is understanding going on. However, if I read a simple story written in Chinese (a language that I do not speak), there is no understanding going on. What makes the difference between these two cases? The advocate of Strong AI says that the difference lies in the fact that I run a particular program in the case of English stories and that

I do not run a particular program in the case of Chinese stories. If the program for understanding Chinese stories were given to me, then I would be able to understand Chinese stories. Similarly, if that program were given to any other sufficiently complex system (e.g. a Turing machine), then it too would be able to understand Chinese stories. The general thesis of AI is labelled below as (*AI**). The thesis that Searle considers—whether the running of a particular program is sufficient for, or constitutive of, understanding simple Chinese stories—is labelled (*AI*):

(*AI**) Running a program is sufficient for, or constitutive of, mentality.

(*AI*) Running a program is sufficient for, or constitutive of, understanding simple Chinese stories.

Searle argues that (*AI*) is false. If this were so, then the general claim (*AI**) would be false too, since one of its instances is false. At this juncture, it would be open to an advocate of Strong AI to modify her position to accept the falsity of (*AI*), while still holding on to the general idea that running a program is constitutive of perhaps other aspects of mentality. However, this is not an option that I shall consider. I shall focus only on Searle's argument against (*AI*).

Searle's argument against (*AI*) is as follows. Imagine a monolingual English speaker inside a room with a rule-book and sheets of paper. The rule-book contains instructions in English on what to do if presented with Chinese symbols. The instructions are of the form: 'If you see Chinese symbol X on one sheet of paper and Chinese symbol Y on another, then write down Chinese symbol Z on a third sheet of paper'. Pieces of paper with Chinese writing are passed into the room and the person inside follows the rules and passes pieces of paper out. Chinese speakers outside the room label the sheets that are passed in 'story' and 'questions' respectively, and the sheets that come out 'answers to questions'. Imagine that the rule-book is as sophisticated as you like, and certainly sophisticated enough that the responses that the person inside the room gives are indistinguishable from those of a native Chinese speaker. Does the person inside the room thereby understand Chinese? Searle claims that they do not. The person inside the room does not understand Chinese simply in virtue of following a particular rule-book. Some respondents to Searle reject this claim, arguing that the person inside the room does understand Chinese. However, I am going to grant Searle his key intuition, and claim that his argument fails on other grounds.[1]

Searle notes that the Chinese room is a computer, and he identifies the

[1] In this and what follows, the expression 'understanding simple Chinese stories' is sometimes abbreviated to 'understanding Chinese'.

rule-book with the program that it runs. He then reminds us that the thought experiment does not depend on the particular rule-book used. It does not matter how sophisticated the rule-book, the person inside the room will still just be mindlessly shuffling symbols, and not understanding Chinese. Since rule-books are programs, the thought experiment demonstrates that the Chinese room cannot understand Chinese *no matter what program it runs*. The Chinese room is a universal computer, which according to Searle, means that it can run any program. Therefore, we can conclude that *no program can be constitutive of understanding*.

The argument is sometimes presented in an alternate form. Take the best attempt at a program that is constitutive of understanding—for example, the best program that the AI research could ever hope to produce, or the program that actually runs on Chinese speakers' brains. Since the Chinese room is a universal computer, it will be able to run that program. However, we cannot imagine the person inside the Chinese room *ever* understanding Chinese, no matter what program they are given. Hence, no such program that is sufficient for, or constitutive of, understanding can exist.

Both arguments appear in Searle's work.[2] Both arguments depend on an assumption that I am going to challenge. I will lay out the arguments in more detail now.

The first argument has two premises. The first premise is Searle's key intuition that the person inside the room does not understand Chinese stories. The second premise is that the person inside the room can run any program. These two premises deductively entail that no program can be sufficient for, or constitutive of, understanding Chinese stories. The argument has the following form:

Searle's straight argument

	The man in the room cannot understand Chinese stories
(*G*)	The man inside the Chinese room can run any program
∴	No program can be sufficient for, or constitutive of, understanding Chinese

Therefore, (*AI*) is false.

Searle begins his original article with the second argument given above. This argument has the form of a *reductio ad absurdum*. It runs as follows: 'Let's

[2]An example of the first argument: 'I [the man inside the Chinese room] have inputs and outputs that are indistinguishable from those of a native Chinese speaker, and I can have any formal program you like, but still I understand nothing.' (Searle, 1980b, 418). An example of the second: 'I offer an argument that is very simple: instantiating a program could not be constitutive of intentionality, because it would be possible to instantiate the program and still not have the right kind of intentionality. That is the point of the Chinese room example.' (Searle, 1980a, 450).

assume that the mind actually works this way, let's assume that Strong AI is correct, what consequences ensue?' One consequence is that the person inside the room, if given the right rule-book, would be able to understand Chinese. But, says Searle, that is *absurd*. We cannot imagine the person inside the room ever understanding Chinese no matter what rule-book they are given. Therefore, something must have gone wrong with our premises. Our main premise was that Strong AI is true, so it must be that Strong AI is false. This argument has the following form:

Searle's reductio argument

(*AI*)	Running a program is sufficient for, or constitutive of, understanding Chinese stories
(*G*)	The man inside the Chinese room can run any program
∴	The man inside the room can understand Chinese stories

Can't be!

So either (*AI*) or (*G*) must be false.

(*G*) is true, therefore (*AI*) must be false.

Both versions of Searle's argument depend on assumption (*G*): the assumption that the Chinese room can run any program. It is easy to see why. Searle's argument only works as an argument against Strong AI if the Chinese room argument speaks to the general case. Otherwise, it would leave open the possibility that *there is* a program constitutive of understanding, but it so happens that the Chinese room cannot run that program. In order to make the Chinese room argument work as a general argument against Strong AI—in order to make it speak to the general case of Strong AI rather than to just the specific case of the Chinese room—Searle needs a generalising assumption like assumption (*G*). My strategy in criticising Searle is to argue that assumption (*G*) is false.

1.1.1 Qualified versions of Searle's argument

It is possible to weaken assumption (*G*) and keep Searle's conclusion. Below are variants of the arguments above that weaken assumption (*G*) as much as possible while preserving his anti-Strong AI conclusion:

Searle's qualified straight argument

	The man in the room cannot understand Chinese stories
(*G**)	The man inside the Chinese room can run all programs putatively constitutive of understanding
∴	No program can be sufficient for, or constitutive of, understanding Chinese

Therefore, (*AI*) is false.

Searle's qualified reductio argument

(*AI*)	Running a program is sufficient for, or constitutive of, understanding Chinese stories
(*G***)	The man inside the Chinese room can run a program constitutive of understanding
∴	The man inside the room can understand Chinese stories

Can't be!

So either (*AI*) or (*G***) must be false.

(*G***) is true, therefore (*AI*) must be false.

The argument that I present below works just as well against assumptions (*G**) and (*G***) as it does against assumption (*G*). Therefore, in what follows my claim is not just that (*G*) is false, but also that (*G**) and (*G***) are false. The discussion is phrased in terms of assumption (*G*) only for reasons of convenience. Searle's argument cannot be saved by weakening assumption (*G*).

1.1.2 Standard replies

Before turning to the details of the argument, it is worth comparing this strategy to standard replies to Searle.

Robot reply

Many advocates of artificial intelligence argue that an extra ingredient is required for a Chinese room to understand, and that extra ingredient is human-like causal connection to the world. A computer working in isolation could not understand Chinese stories, but a robot with appropriate causal connections potentially could. This reply, as Searle notes, concedes the main point of his argument. The robot reply admits that more than just computation is needed for understanding: appropriate causal connections are also needed. My reply differs from the robot reply. The robot reply accepts the conclusion of the Chinese

room argument, while I reject it. The robot reply describes how computers can be *supplemented* so as to understand, my reply focuses on a mistake in the original Chinese room argument. Searle goes on to argue that even if one were to accept the robot reply, Strong AI still cannot be saved: a robot with appropriate causal connections with the world still would not be able to understand Chinese. However, Searle's additional argument will not be considered here.[3]

Systems reply

The systems reply claims that although the man cannot understand Chinese, the system as a whole—the man, plus rule-book, plus pens, plus pieces of paper—can understand Chinese. The systems reply switches the subject of understanding from the man to the room. The hope is that by performing this switch, the advocate of the systems reply can avoid Searle's conclusion. She does this in the following way.

First, she avoids Searle's *reductio* argument. Searle's *reductio* was that if Strong AI were true, then the man inside the room would be able to understand Chinese given the right rule-book, and that is absurd. The systems reply holds that if Strong AI were true and the person inside the room were given the right rule-book, then *the room as a whole* would be able to understand Chinese. The advocate of the systems reply claims that this conclusion is not absurd.

The system reply variation I

(*AI*)	Running a program is sufficient for, or constitutive of, understanding Chinese stories
(*G****)	The room can run any program
∴	The room can understand Chinese stories

The advocate of the systems reply accepts this conclusion.

Searle's straight argument can also be avoided if one accepts that the room as a whole can understand Chinese stories. In Searle's straight argument, if (*G*) is replaced with (*G****), then Searle's conclusion—that no program can be sufficient for, or constitutive of, understanding Chinese—is no longer deductively entailed.

The system reply variation II

	The man in the room cannot understand Chinese stories
(*G****)	The room can run any program

Searle's original conclusion does not follow.

[3]For this argument against the robot reply, see Searle (1980b), 420.

In both cases, the advocate of the systems reply needs to accept assumption (G^{***}), or something like it, in order to support her claim that the room can understand Chinese. My reply differs from the systems reply in that I reject not only assumption (G), but also assumption (G^{***}). I claim that it is not only false that the man can run any program, it is also false that the room can run any program.

Such a variety of arguments have been grouped under the heading of systems replies that this cannot do justice to all of them. Some variants of the systems reply are compatible with my reply. For example, Copeland's (2002) 'logical reply' claims that Searle commits a fallacy in the inference from the man not understanding Chinese to there being no Chinese understanding inside the room, but does not itself assert that the room understands Chinese. The logical reply is compatible with my reply since it is not committed to the claim that the room itself can run programs constitutive of understanding. Nevertheless, the logical reply and my reply are different. Both replies involve pointing out a mistake in Searle argument, but they are different mistakes. My reply claims that Searle makes a mistake by assuming that the Chinese room can run the relevant programs, the logical reply claims that Searle makes a mistake by assuming that if the man inside the Chinese room cannot understand Chinese then the system as a whole cannot understand Chinese.[4]

Procedural semantics reply

Another popular reply to Searle is to claim that Searle's key intuition—that the person inside the room cannot understand Chinese—is false. This reply typically takes one of two forms. In the first form, the respondent claims that the man inside the Chinese room *fully* understands Chinese. Any commonsense intuitions to the contrary are, like many commonsense intuitions, mistaken.[5] The second form of this reply admits that the man inside the room cannot fully understand Chinese, but claims that he nevertheless can have a *partial* understanding of Chinese. Typically, the suggestion here is that the man can learn the procedural semantics of Chinese, even though he cannot learn what the Chinese symbols refer to.[6]

Both kinds of reply face problems. Searle's intuition that the man does not understand Chinese is fundamental to his position. It is not obvious how to find grounds on which to over-rule this intuition without begging the question against Searle. For example, appeal to third-person evidence is regarded as suspect by Searle in this context because the priority of third-person evidence

[4] See Copeland (2002) for details of the logical reply.
[5] For example, Block (1980); Hofstadter and Dennett (1981).
[6] For example, Boden (1989); Sloman (1986).

over his first-person intuitions is itself at stake in the Chinese room argument.[7] The second kind of reply faces the problem that it leaves an important sense of Chinese understanding untouched, namely, that associated with referential semantics. Even if computation is sufficient for some aspects of understanding Chinese, it appears to be enough for Searle's argument to work if other aspects of understanding Chinese are not achievable by computation. It is not obvious that either of these problems is serious, but they both require careful attention if either reply is to be successful against Searle. My strategy in criticising Searle differs from both these replies. I grant Searle his key intuition and claim that his argument fails on other grounds.

1.1.3 The Church–Turing thesis

Before discussing why assumption (*G*) is false, let us first consider why Searle might think that it is true. Searle does not say much about this in Searle (1980b), but it is possible to infer what his views are likely to be. Consider the following statement in Searle (1992):

> We begin with two results in mathematical logic, the Church–Turing thesis and Turing's theorem. For our purposes, the Church–Turing thesis states that for any algorithm there is some Turing machine that can implement that algorithm. Turing's thesis says that there is a universal Turing machine that can simulate any Turing machine. Now if we put these two together, we have the result that a universal Turing machine [which is what Searle thinks a Chinese room is] can implement any algorithm whatever. (Searle, 1992, 202)

If this characterisation is correct, then it would justify Searle's assumption that a Chinese room can run ('implement') any program. However, there are two problems. First, Searle's statement of the Church–Turing thesis is incorrect. The Church–Turing thesis does not state that for any algorithm there is some Turing machine that can implement that algorithm. The Church–Turing thesis states that for any computable *function* (set of input–output pairings) there is some Turing machine that can reproduce that function. There is a big difference between a function (a set of input–output pairings) and an algorithm used for computing that function. Searle needs equivalence in algorithms, not functions, for his argument to work, and the Church–Turing thesis is silent about that. The second problem with Searle's statement is that the notion of simulation in Turing's theorem is a technical one. In the context of Turing's theorem,

[7] See Sprevak (2005) for a discussion of this issue, and Searle (1980a) for the status of third-person evidence.

simulation means reproduction of input–output pairings. Simulation does not mean, as Searle suggests, implementation of the same algorithm.

The Church–Turing thesis and Turing's theorem do not justify the assumption that the Chinese room can run any program, even under the dubious assumption that the Chinese room is a universal Turing machine. Therefore, the Church–Turing thesis cannot support Searle's argument in the required way. However, my argument is not just that assumption (*G*) is unjustified, but that it is false. If I am correct, then one should not be surprised that the Church–Turing thesis does not licence assumption (*G*), since such an entailment would be false.[8]

1.2 Criticism of Searle's argument

Searle's argument against (*AI*) depends on some version of the assumption that the Chinese room can run any program. Without this assumption, Searle would leave open the possibility that there is a program constitutive of understanding, but that it so happens that the Chinese room cannot run that program. We saw that Searle's reasons for believing that the Chinese room can run any program were doubtful. I will now argue that the assumption that the Chinese room can run any program is false.

The assumption that the Chinese room can run any program is false because even if the Chinese room is a universal computer, it is *not true* that a universal computer can run any program. The programs that a computer (universal or otherwise) can run depend on that machine's architecture, on the basic operations that machine can perform. Programs are at the algorithmic level, and that level is tied to the implementation on particular machines. One cannot take a program that computes the addition function on, say, a register machine and run *that same program* on a Turing machine. The programs that can be run on the two machines are different in each case. This does not mean that a register machine and a Turing machine cannot compute the same functions. Famously, they can. The Church–Turing thesis states that any computer's input–output behaviour can be exactly reproduced by a universal computer. But this is equivalence at the level of input–output behaviour, and what Searle needs for

[8]Copeland (1998) also criticises Searle for making a mistake with the Church–Turing thesis. Searle says that the Church–Turing thesis entails that any physical process can be simulated by a computer (Searle, 1992, 200). Copeland correctly points out that this is false: the Church–Turing thesis entails that a process can be simulated only if there is an *effective method* governing that process's input–output behaviour. Copeland argues that this requirement need not be met by all physical processes. Both the current and Copeland's point concern the Church–Turing thesis, but they are different points. Copeland's point concerns the conditions under which the Church–Turing thesis can be applied to certain physical systems. My point concerns *what can be gained from* the Church–Turing thesis *once it has been applied to* those systems.

his argument to work is equivalence at the level of *how* that input–output is achieved, i.e. equivalence in the programs run.

The Chinese room can, by definition, reproduce the input–output characteristics of any Chinese speaker. However, this is not enough for Searle's argument. What Searle needs is for the Chinese room to reproduce that input–output pattern in the same way—to run the same programs—as a Chinese speaker. In particular, Searle needs the Chinese room to be able to run the programs that Strong AI claims are constitutive of understanding. However, the programs that a system can run depends on that system's architecture. Certain architectures can run some programs and not others. This is true independently of whether the system is a universal computer or not. Being a universal computer reflects a capacity for universality in input–output behaviour, not a capacity for universality in programs that can be run.[9]

No computational architecture can run all programs. A Chinese room can run some programs and not others. The problem for Searle is that there is no reason to think that a Chinese room can run all or even any of the programs that are putatively constitutive of understanding. Indeed, there are good reasons to think that it cannot. Systems that do understand, for example, human brains, have a radically different computational architecture to that of a Chinese room. An advocate of Strong AI could claim that only these brain-like architectures are capable of running programs that are constitutive of understanding. Since Chinese rooms cannot run these programs, Searle's argument cannot rule out the possibility that these programs produce understanding. Therefore, an advocate of Strong AI could legitimately hold onto her claim.

Searle's argument fails because it fails to consider all the relevant programs. Searle's assumption—that the Chinese room can run the programs that the advocate of Strong AI thinks are constitutive of understanding—seems false. At the very least, it is an assumption that would require substantial defence. We've seen that Searle's defence in terms of the Church–Turing thesis does not work. Searle does not have anything to add to this defence. So as things stand, the advocate of Strong AI has the upper hand. She can justifiably reject the Chinese room argument. She can say that the Chinese room argument fails to show that understanding cannot consist in the running of a program, because the programs that she thinks are constitutive of understanding *aren't even considered by the argument*.

[9] Briefly put, the mistake that Searle makes is to conflate the computational and algorithmic levels distinguished by Marr. Marr's computational level concerns *what* function is computed, his algorithmic level concerns *how* that function is computed. See Marr (1982), 22–24.

1.2.1 Differences in architecture

I have said that the architecture of the Chinese room is different from that of brains. What are the specific differences between the two systems? There is not enough known to give a full answer, but there are good reasons for thinking that there will be radical and irreconcilable differences.

For example, consider the difference between parallel and serial architectures. The Chinese room is specified as having a serial architecture (it performs one operation at a time). Assume, as seems reasonable, that the brain has a parallel architecture (it can perform more than one operation at a time). Programs that exploit specifically parallel architectures, for example, parallel search algorithms, literally cannot be run on serial machines. They rely on certain capabilities and atomic operations that are not available on those machines. Serial and parallel computers can compute the same functions, but they cannot use the same methods for computing those functions. If the brain has a parallel architecture, and the programs that it runs are essentially parallel—as seems likely—then those programs literally cannot be run on a Chinese room.

Serial–parallel differences are only one of many ways in which the architecture of brains is likely to differ from that of Chinese rooms. Another possible difference is that the two systems may support different types of atomic operation. As specified by Searle, the Chinese room supports the atomic operations *compare*, *copy*, and *concatenate*. There seems no reason why the brain should support just these operations. If there is any difference in the atomic operations supported, then the individual steps in the programs on the two machines cannot be the same, creating problems for the notion that the two machines can run the same program.

Another possible source of difference is that the brain may have a different way of managing control from the Chinese room. There are many examples of architectures with different control from the von Neumann model that the Chinese room approximates. Consider the difference between the programs that can be run on an ordinary von Neumann-style personal computer and those that can be run on a computer based on Church's λ-calculus. Imagine trying to run a PC version of Microsoft Word on a machine with a λ-calculus architecture. Such a program could not be run as it currently stands. A λ-calculus version of Microsoft Word—a program with the same input–output pairings—would have to work in a radically different way. (For one thing, on a λ-calculus architecture there are no loops, instructions, or stored variables. Therefore, the standard von Neumann construction of a looped assignment statement cannot be run.) The brain may have a computational architecture that is as distant, or more distant, from the von Neumann model than the λ-

calculus. For example, the von Neumann distinction between central processor and memory seems not to hold for the brain. Neurons, or group of neurons, appear to act as independent processors, and memory seems to be encoded in these processing units rather than in an independent central store.[10]

1.2.2 Summary

Searle's Chinese room argument fails because his assumption (G), or any suitably qualified version of it, is false. The architecture of brains and Chinese rooms is different. There is no reason for the advocate of Strong AI to admit that *even one* program that is putatively constitutive of understanding can be run on a Chinese room. She could legitimately claim that understanding-producing programs are run on brain-like architectures. So as things stand, the advocate of Strong AI can hold onto her Strong AI thesis for architectures other than that of the Chinese room. Searle's argument against Strong AI fails to rule out the possibility that understanding can consist in the running of a program.[11]

1.3 Objections and replies

1.3.1 The virtual brain machine objection

Objection 1. *Can't a Chinese room simulate any other computer? Can't one create, on the Chinese room computer, a virtual machine with the same architecture as the brain and run the brain's program on that virtual machine? Wouldn't the Chinese room then be able to run any program that could be run on the brain?*

Consider how such a proposal would work. Suppose that Strong AI claims that there is a program P that runs on the brains of Chinese speakers, and that program P is sufficient for, or constitutive of, understanding Chinese. Program P will be peculiar to the architecture of the human brain. It will use distinctive atomic operations, and manage flow of control and data in distinctive ways. Program P cannot be run on a Chinese room as it stands. The current suggestion is that one can get around this incompatibility by creating another program Q that runs on the Chinese room, a virtual brain machine (VBM). Program Q takes

[10] See Backus (1978) for a survey of different computational architectures and the effect that architecture has on the programs that a system can run. See Johnson-Laird (1988) for a comparison between the von Neumann model and the brain.

[11] This reply to Searle has not, to my knowledge, been made in the literature. The replies to Searle that come closest are those of Lycan (1980); Rey (1986); Roberts (1990). None of these authors make the connection with the Church–Turing thesis or the architecture-dependence of programs. I came upon my criticism independently, but my discussion can rightly be seen as an elaboration and defence of their original suggestion.

programs designed to be run on the human brain as data. Program *Q* processes these brain-like programs and produces output that is indistinguishable from that of the programs being run on a real brain. For many intents and purposes, a Chinese room running a virtual brain machine can be treated as a machine with the architecture of a brain. In particular, one can run program *P* on it. Therefore, *contra* my claim, the Chinese room can run program *P*—all that it needs is a VBM to provide an emulation layer. Furthermore, adding a VBM only adds another level of symbol shuffling, so it still seems likely that even with a VBM, the person inside the room would still not be able to understand Chinese. Therefore, program *P* cannot be sufficient for, or constitutive of, understanding Chinese. Searle's original argument goes through after all.

This objection relies on the notion of a virtual machine, and although that notion is compelling, it is easy to take it too literally. Virtual machines are not the machines they appear. Virtual machines present interfaces that give the impression that a real machine is there, but they do not strictly speaking run the program that they are given. Virtual machines are automated procedures for turning one program into another. A VBM does the following: for each step or small group of steps *S* in program *P*, it transforms those steps into a group of steps *S** for the Chinese room to run, and then runs *S**. The VBM does not run program *P*, it runs a transformation of program *P*. In this respect, virtual machines are like compilers: they transform one program (the one that the user writes) into another program (the one that the machine runs). The difference is that compilers do the translation beforehand and all at once, virtual machines do the translation piecewise and at runtime.

The original program and the one produced by the VBM have the same input–output characteristics, but they will almost certainly work in different ways. This is invariably the case when one compiles across different architectures, or when a virtual machine has to adapt to the underlying architecture of a particular machine. If the architectures of the real and the virtual machine are significantly different, then the transformed program and the original will only be distantly related: this can be to the extent that the instructions are different, the order in which instructions are executed is different, extra steps inserted, steps removed, and large-scale features of the program changed.

Therefore, even if one were to construct a VBM, and give that VBM program *P*, the Chinese room would still not be running program *P*. The Chinese room would be running a systematic transformation of program *P*: a program with different steps, different atomic operations, and a different structure. A virtual brain machine creates the illusion of the real brain being there—it presents the same I/O interface to the world—but it does not literally run the program that it is given. Adding a VBM does not get one closer to running program *P* on the

Chinese room.

The VBM does not allow a Chinese room to run program P because the original program P and its VBM-transformed counterpart P^* are likely to be very different. In the case of the Chinese room, they are likely to be different because the architecture of the virtual machine (the human brain) the real machine (the man and the rule-book) is so different. However, in the cases where the architectural difference between machines is small, decisions about program identity are not so clear. For example, if the virtual machine involves relatively minor transformation, then programs P and P^* might well be counted as the same. How much transformation is too much? What is the cut-off point for P and P^* being the same program? The answer is unclear. In later chapters, a finer-grained treatment of this issue is provided. An analysis is given of the conditions under which a system performs one computation rather than another.

1.3.2 The man with a brain objection

Objection 2. *Your criticism was that the Chinese room cannot run suitably brain-like programs, but the man inside the Chinese room has a brain, so surely he can run brain-like programs.*

This objection conflates two distinct machines that are both inside the Chinese room. One machine is the human brain, the other is the von Neumann-esque machine produced by the activity of the man inside the room. Call the man inside the Chinese room the 'clerk'. The claim of Strong AI is that a program that runs on the first machine, the brain, constitutes understanding. Searle's argument is that if merely running a program is constitutive of understanding, then we can *take* the program that runs on the brain and run that program on the other machine: we can get the clerk to work through the program by hand. If merely running a program is sufficient for understanding, then understanding should be produced in this second case too. But, Searle says, it is absurd to think that the clerk could understand Chinese merely by working through a program. So, running a program cannot be sufficient for, or constitutive of, understanding. My point is that you cannot transfer programs in the way that Searle suggests. The architecture of the two machines, the brain and the von Neumann-esque machine, is just too different.

A possible confusion underlying this objection is that there are two distinct ways in which the phrase 'running a program' is used in the Chinese room argument. In some cases, 'running a program' means running the program directly on the hardware of the brain. In other cases, 'running a program' means the clerk working through the program by hand. Both senses of 'running a pro-

gram' are legitimate. However, programs are being run on different machines in each case. Searle requires this distinction in order to make his argument. His crucial move in the Chinese room argument—taking the program that runs on a brain and giving that program to the clerk—does not make sense otherwise. The current objection conflates the two machines. Its upshot would be that Searle's argument, and my criticism, would be impossible to understand.

One might ask what the relationship is between the two machines inside the Chinese room. They are clearly not independent: the von Neumann-esque machine would cease to exist if the clerk's brain were absent. However, the nature of the relationship is not obvious. Fortunately, the nature of the relationship does not matter to the current argument. All that matters is that there *are* two machines inside the Chinese room. In later chapters, an account is given of how a single physical system—such as the clerk's body—can house two different computers: the von Neumann-esque machine and the brain.

1.3.3 The syntax/physics objection

Objection 3. *You say that the Chinese room cannot run the relevant program, but according to Searle's 'syntax is not intrinsic to physics' argument, any sufficiently complex physical system runs any program (Searle, 1992, Ch. 9). So the Chinese room can run any program after all.*

This response would constitute a retreat for Searle, since he has claimed that the Chinese room argument stands independently of his later syntax/physics argument.[12] However, for the purposes of this reply, I shall consider how such a defence would work.

The conclusion of Searle's syntax/physics argument is that any sufficiently complex physical system runs any program. According to Searle, all that it takes for a system to run a program is for an observer to be able to interpret that system's physical states in the right way. If the system is sufficiently complex, like a Chinese room, then an observer can interpret it as running any program she likes. My claim is that there is something about the Chinese room, its architecture, that constrains the programs that it can run. If Searle's syntax/physics argument is correct, then there are no interesting constraints on the programs that a Chinese room can run.

There are two issues to be addressed here. First, there is the issue of whether Searle's syntax/physics argument is correct. If Searle's syntax/physics argument is incorrect, then the advocate of Strong AI is under no compulsion to accept its consequences. In subsequent chapters, I argue that Searle's syntax/physics argument is incorrect, or rather, that it presupposes certain forms of anti-realism

[12] For example, see Searle (1992), 210; Searle (1994), 548; Searle (1997), 14, 17.

that the advocate of Strong AI need not accept. The second issue is whether the syntax/physics argument is a good way for Searle to defend the Chinese room argument. I believe that it is not. The syntax/physics argument is double-edged: it cuts against the original Chinese room argument just as much as it does against Strong AI. The syntax/physics argument cuts against the Chinese room argument because it undermines the ability to state the Chinese room argument and it undermines the motivation for giving the Chinese room argument.

The syntax/physics argument undermines the ability to state the Chinese room argument because the Chinese room argument itself presupposes a determinate notion of computation. Searle acknowledges this: 'For the purposes of the original [Chinese room] argument, I was simply assuming that the syntactical characterization of the computer [assignment of computations to physical states] was unproblematic.' (Searle, 1992, 210). The Chinese room argument assumes that a system runs *a* program. Without the assumption, it is difficult to make sense of Searle's talk of taking 'the' program that is putatively constitutive of understanding and 'giving' that program to the Chinese room, or his identification of the rule-book with the program that the Chinese room runs.

The syntax/physics argument undermines the motivation for giving the Chinese room argument because it renders Strong AI false from the start. One of Searle's most firmly entrenched background assumptions is that understanding is an intrinsic property of a system. A system, such as a human being, either understands or does not understand, and this is true independently of how observers view that system. If one were to conjoin to this assumption the conclusion of the syntax/physics argument—that the program a system runs is an observer-*relative* property—then the conclusion immediately follows that running a program cannot be sufficient for, or constitutive of, understanding. An observer-relative property cannot be an observer-independent property. Strong AI is refuted without any need for the Chinese room argument. This is a mixed blessing. One of the reasons why the Chinese room argument was so interesting is that it shows that *even if one helps oneself to a notion of determinate computation*, understanding cannot consist in the running of a particular program. Searle grants as many assumptions as possible to Strong AI in an effort to provide a convincing argument against Strong AI. In the revised argument above, one of these assumptions—the determinacy of computation—has been given up, and the overall argument is less interesting as a result.

1.3.4 The abstraction objection

Objection 4. *You say that the Chinese room cannot run the relevant program, but the programs that a computer runs can be characterised more or less abstractly depending on the interests of the users (Lycan, 1987, 46–47). If programs are construed abstractly enough, then the Chinese room can run them all.*

Lycan (1987) claims that there is no single program that a computer runs. His reasoning is that there is no single correct level of program characterisation. A computer can be characterised as running any number of programs depending on how abstractly one wishes to treat it. For example, a chess-playing computer can be characterised as running a program that develops knights and looks for kingside weaknesses, or as running a program that evaluates trees of chess moves in a certain order, or as running a program whose steps are instructions in the programming language that the chess program was written, or as running a program whose steps are the machine code instructions of the processor on which the program is running.

At the most abstract level of program characterisation—developing knights and looking for kingside weaknesses—it seems plausible that two systems with different architectures could run the same program. At the least abstract level—executing a set of specific machine code instructions—the possibility of running the same program looks less likely. For Lycan, the question, 'Do two machines run the same program?' does not have a single answer. Whether two machine run the same program depends on the level of program characterisation, and that level can vary depending on the goals and interests of the users.

First, note that even if Lycan is right, it is unclear how much this result would help Searle. Lycan may be right that questions about program identity are not easy to decide. There is often considerable room for debate about whether the programs that two systems run are the same or different. However, whatever flexibility there is, it is unlikely to be enough to support Searle's argument. Consider, for example, the task of moving a parallel search program, running on a parallel machine, to a serial machine. In what sense could the two machines run the same program? If, as seems likely, an important feature of the parallel search program is that the search be performed in parallel, how could that feature be preserved on a serial machine? Moving to higher levels of abstraction just does not help. No matter how abstractly one characterises the program, one cannot perform parallel operations on a serial machine. If the parallel nature of the search is important, then that feature cannot be replicated on a serial machine, no matter how abstractly the program is construed.

Here is another example. Consider two versions of Microsoft Word, one for a λ-calculus computer and one for an ordinary PC. At what level of abstraction,

short of I/O equivalence, could they be said to run the same program?

These are just two examples and involve relatively small differences in architecture. The architectural differences between the Chinese room and the human brain may be much greater. For this reason, it is unclear how appeal to levels of abstraction can solve Searle's problem. Given the difference in architecture between brains and Chinese rooms, it is unclear, and *prima facie* implausible, that the two systems could run the same programs no matter how abstractly construed, short of treating programs as mere sets of I/O pairings.

It is easy to forget the degree to which programs are architecture-dependent. Nearly all of the electronic computers with which we are familiar have more or less the same underlying architecture. This makes moving programs between machines relatively straightforward. It is possible to move programs while preserving many of their key structural features. However, the straightforward nature of these cases should not lead us to assume that moving programs to machines with radically different architectures is similarly easy. Implementing the same computational method on a completely different architecture may not be possible at all.

A second problem is that it is by no means clear that Lycan's position about program identity is correct. Lycan's argument presupposes that the facts about program identity can be read off from our informal intuitions about program identity. According to our informal intuitions, each of the levels of abstraction that Lycan mentions is equally valid, and so there is no single answer to a question of program identity. But it could be argued that these intuitions are wrong. For example, Pylyshyn (1984) argues that there is a privileged level of abstraction at which questions about program identity should be decided. This level is the level of the system's primitive representations and primitive operations. Two systems with different architectures will not be able to run the same programs unless they share the same primitive representations and operations. For Pylyshyn, talk of more abstract levels of program organisation may be of pragmatic value, but it should not be given any metaphysical weight in deciding questions about program identity. Pylyshyn's position requires considerable defence, but it at least shows that there are alternatives to Lycan.[13]

Finally, consider how remarkable it would be if Searle's claim were true. The Church–Turing thesis is a remarkable result. It guarantees I/O equivalence between certain classes of computational systems. The assumption that Searle makes is stronger: it entails the Church–Turing thesis *and* that any sufficiently powerful computational system could run the same programs as any other system. If this were true, it would be truly extraordinary. The absence of any such result from the mathematical logic should make us cautious. Consider the

[13] Pylyshyn (1984), 91–92. See Section 2.3.3 for more discussion of Pylyshyn's views.

effort that has gone into supporting the Church–Turing thesis. A huge amount of detailed work was required to show that various computational systems are I/O equivalent. Searle makes a stronger claim and needs extra support for that content. It is not clear how he could have this.

1.3.5 The 'not guilty' objection

Objection 5. *The Strong AI thesis that Searle attacked was not the thesis that you suggest. The thesis against which he argued was either the behaviourist view:*

(BH) *Intelligent input–output behaviour is sufficient for, or constitutive of, understanding.*

or, the architecture-restricted view:

(AR) *Programs that can be run on a PC with a von Neumann-style architecture are sufficient for, or constitutive of, understanding.*

If the target of Searle's argument was either (*BH*) or (*AR*) then, according to the considerations raised so far, his argument would have succeeded. However, it is clear that Searle intended something else. The intended target of Searle's argument is (*AI*):

(*AI*) Running a program is sufficient for, or constitutive of, understanding simple Chinese stories.

It is easy to see why Searle's target is (*AI*) and not (*BH*) or (*AR*).

First, it does not make sense to understand the Chinese room argument as an argument against just (*BH*). The Chinese room argument would be an overkill. An argument like Block's intelligent lookup-table would suffice.[14] Searle himself cautions against a behaviourist interpretation of his argument:

> Notice that the force of the argument is not simply that different machines can have the same input and output while operating on different formal principles [some of which are constitutive of understanding, some of which are not]—that is not the point at all. Rather, whatever purely formal principles you put into the computer, they will not be sufficient for understanding, since a human [in the Chinese room] will be able to follow the formal principles without understanding anything. (Searle, 1980b, 418)

[14] See Block (1981).

Searle also intends his argument as more than just a refutation of (*AR*). Searle explicitly says that his claim is not just that machines with a specific architecture (Chinese room or von Neumann) cannot understand, but that *no* computing machine can understand. His argument is intended to cut across all computer architectures: 'My objection would hold against any program at all, *qua* formal program.' (Searle, 1980a, 452); 'The Churchlands are correct in saying that the original Chinese room argument was designed with traditional AI in mind but wrong in thinking that connectionism is immune to the argument. *It applies to any computational system*.' (Searle, 1990, 22, emphasis added).

It is perhaps true that many of the original advocates of Strong AI did not distinguish their claim from (*BH*) or (*AR*) sufficiently clearly. Those original advocates perhaps can, to some extent, be accused of making the same architecture-independence fallacy as Searle. Nevertheless, regardless of original intentions, Searle's position is unequivocally an attack against thesis (*AI*). Although it is interesting that his argument can be reinterpreted as an attack on (*BH*) and (*AR*), that does not affect the point that it leaves (*AI*) untouched.

1.3.6 The 'same conclusion' objection

Objection 6. *It does not matter if Searle made a mistake, your conclusion about the importance of the underlying architecture, the brain, is the same as his anyway.*

The architecture-dependent view that I place at the disposal of the advocate of Strong AI does mesh with some of Searle's comments about the importance of underlying biology. According to the view that I suggest, the brain runs a particular program because it has a distinctive architecture, and it has a distinctive architecture because of its physical makeup.[15] If one were to create an artificial system that runs the same programs as the brain, then one would create a system that, in some important respects, is similar to a brain. Therefore, the view I suggest and Searle's own position both give an important role to biology. However, the two positions are different.

The difference lies in two respects. First, the motivation for appealing to underlying biology, and the role played by that biology, are different in each case. Searle's appeal to biology has been justly criticised for lack of motivation.[16] Searle seems to conclude that since computation is not enough for understanding, there must be some other factor, and he appears to pick biology for lack of a better candidate. On his account, it remains mysterious

[15] Cf. Marr (1982): 'The algorithm depends heavily on the computational theory, of course, but it also depends on the characteristics of the hardware in which it is to be implemented.' (p. 337). Remember that for the purpose of the Chinese room argument, both Searle and I are bracketing syntax/physics worries, see Objection 3.

[16] For example, see Block (1980); Dennett (1980); Fodor (1980b); Pylyshyn (1980).

what features of the underlying biology are relevant, or *why* biology should be relevant at all—biology appears to play the role of a *deus ex machina*. On the view that I suggest, there is a clear rationale for appealing to biology. The physical makeup of brains determine their computational architecture, and therefore the programs that brains can run. Biology is important because it determines the type of hardware that supports the programs required for understanding. Only appropriately brain-like architectures can run the kinds of programs constitutive of understanding.

The second difference between the two views is that the claim that Searle denies—understanding can consist in running a program—is asserted on the competing view. Therefore, the end positions of the two views are different. It is worth noting that even on the view that I suggest, it is still possible for the claim of Strong AI to turn out to be false. Even if the Chinese room argument does not prove it false, the claim that understanding can consist in running a program may turn out to be false for independent reasons.

On the view that I suggest, appeal to biology is motivated by the general claim that physical makeup determines computational architecture. However, nothing has been said to justify this claim, or to explain *how* physical makeup determines computational architecture. In computer science, it is often accepted that physical makeup does determine computational architecture. But as we shall see, it is far from clear that this claim is true. In subsequent chapters, these two issues are dealt with in detail. I argue that physical makeup *can* determine computational architecture, and I provide a detailed account of the relationship between the two.

1.3.7 The 'unnecessary baggage' objection

Objection 7. *The situation that you suggest—a brain-like computer running a characteristic program—involves no more than reproducing the causal powers of the human brain, which was Searle's original suggestion. Why not drop the idea of running a program as unnecessary baggage, and say, as Searle does, that understanding systems should be characterised in physical terms, i.e. in terms of their causal powers?*

This objection is different from Objection 6. Objection 6 claimed that the positions suggested by Searle and myself are same. The current objection admits that the two positions are different, but claims that Searle's position is preferable to my own. The claim is that the explanatory virtues of the account that Searle suggests are greater than those of the account that I suggest: the notion of computation in my account is just unnecessary baggage. The disagreement here is not about the *members* of the class of systems that understand—both Searle and I agree that only a class of brain-like systems can understand. The

disagreement is *how that class should be characterised*: how one should describe what membership of that class involves. Should the class be characterised purely in terms of physical features, as Searle suggests, or should it be characterised in terms of computational features, as I suggest? Let us consider the pros and cons of each approach.

The computational approach that I suggest has had numerous explanatory successes. It explains a number of puzzling features of systems that understand such as the systematicity and productivity of thought, the possibility of rational inference, and why some aspects of the relationship between thought and language obtain. There are deficiencies in the computational account, and computation may not be the whole story about the mind, but it does have something useful to say about what understanding systems have in common.

Now consider Searle's way of characterising the class of understanding systems. It is hard to see how his account could give a satisfying explanation of what such systems have in common. What purely physical features do understanding systems have in common? Acetylcholine and serotonin? Unlikely, for other substances could have been used in their place. There is nothing special about acetylcholine and serotonin apart from their ability to occupy certain roles in the physiology of understanding systems. Accounts of how systems understand, like accounts of other biological phenomena, should be structural or functional. It is causal roles and their interrelation that explain why a given biological system performs the function it does. This is true in general of explanations of biological phenomena. It is true even of Searle's paradigmatic example of biological phenomena, photosynthesis. Look at any scientific account of photosynthesis and one finds functionalist explanation *par excellence*.[17] For this reason, it is unclear why, or even how, Searle could be hostile to functionalist accounts.

If Searle is not hostile to functionalist accounts, then it is unclear why he is hostile to giving the causal roles involved a computational gloss. As we saw in the previous objection, there need be nothing unbiological about such an approach. If computational accounts offer the chance of explaining the cognitive phenomena listed above, then, provided that the notion of computation is not shown to be otherwise problematic, why not use it? Searle's alternative just does not seem plausible.

The same point can be phrased as a dilemma. If Searle's proposal is understood as a brute physical account with no functionalist component, then it would provide a massively disjunctive and unexplanatory characterisation of the class of understanding systems. If his account is understood as explanation in terms of causal role, hence functional, it is difficult to see why he is so

[17] For example, see Lawlor (2001).

hostile to giving those causal roles a computational gloss. There seems nothing to gain and much to lose by such a move. Provided that Searle cannot show that the notion of computation is otherwise unrespectable, there no reason why one should not use the computational explanations of cognitive phenomena described above.

1.3.8 The Chinese gym objection

Objection 8. *If the setup inside the Chinese room cannot run brain-like programs, why not change the setup so that it can? Once the setup is changed to reflect the architecture of the brain, Searle's argument against Strong AI can be run as before.*

Searle (1990) proposes a modification along these lines in response to connectionist criticisms. He suggests replacing the single man inside the Chinese room with a collection of monolingual English-speaking men inside a Chinese gym. Each man would simulate the computational behaviour of a single neuron. The men would pass messages to each other, so that the gym as a whole would perform a connectionist computation. Searle claims that none of the men understand Chinese, and infers from this that there is no Chinese understanding inside the gym. He concludes that connectionist computation cannot be sufficient for, or constitutive of, understanding.[18]

Copeland (1993) correctly points out that this argument would not convince a connectionist. A connectionist does not claim that individual neurons understand, she claims that the system as a whole understands. Searle does nothing to show that the failure of the individual men to understand Chinese entails that the system as a whole cannot understand Chinese. Therefore according to Copeland, this version of the Chinese room argument falls victim to a particularly vicious form of the Systems reply.

However, the fundamental problem with this response—and the reason why it so easily falls victim to the Systems reply—is that unlike Searle's original thought experiment, the key intuition behind this new thought experiment is simply not convincing. A large part of the appeal of Searle's original argument was that his key intuition—that there is no Chinese understanding going on inside the room—was plausible. The reason why it was plausible is that in the original thought experiment it was possible to imagine *oneself* performing the computation and failing to understand Chinese. One could *empathise* with the system carrying out the computation. This feature is lost in the Chinese gym scenario. It is just not possible to imagine what it is like to be a Chinese gym. As a result, Searle's key claim is unconvincing in a way that arguably it is not in the original argument.

[18] For the argument, see Searle (1990), 22.

Searle has not developed the Chinese gym response further, perhaps for this reason. He now favours a different response to connectionists. This involves the claim that the Chinese room is 'computationally equivalent' to connectionist systems:

> Computationally, serial and parallel systems are equivalent: any computation that can be done in parallel can be done in serial. If the man in the Chinese room is computationally equivalent to both, then if he does not understand Chinese solely by virtue of doing the computations, then neither do they. (Searle, 1990, 22)

It should be clear from Section 1.1.3 that this argument does not work. Serial and parallel computers *are* computationally equivalent, but only in the sense that they can reproduce the same I/O behaviour, i.e. compute the same functions. Serial and parallel computers are not computationally equivalent in the sense that they can use the same methods for computing those functions. It is this second, illegitimate, sense of computational equivalence that Searle needs for his argument.

1.3.9 The syntax/semantics objection

Objection 9. *Searle now prefers a different argument, which he calls his 'axiomatic' argument, to the Chinese room argument. You have said nothing about this argument, so why should Searle worry?*

Searle says that his axiomatic argument is a generalisation, and a simplification, of the original Chinese room argument. The axiomatic argument is as follows:

(*A1*)	Programs are formal (syntactical)
(*A2*)	Minds have contents (semantics)
(*A3*)	Syntax is not sufficient for semantics
∴	Programs are not minds

Searle says the Chinese room argument is an illustration of the truth of (*A3*).[19] However, he also says that the Chinese room argument is not necessary, because (*A3*) can be shown to be true in other ways. According to Searle, (*A3*) is a conceptual truth. All that is required to see that (*A3*) is true is to reflect on the nature of the relevant concepts. If one reflects on the concepts of syntax and semantics, one can see that syntax is not sufficient for semantics.

This argument is simple but deceptive. It is deceptive because it trades on an ambiguity in the term 'syntax'. In logic and linguistics, the notion of syntax

[19] See Searle (1994), 547; Searle (1997), 11.

is well defined. Syntax concerns the rules by which expressions in a language qualify as well-formed. It is trivial to show that syntax is not sufficient to fix the semantic content of expressions. Syntax, in the context of logic and linguistics, is not sufficient for semantics.

However, this sense of syntax is not the sense at issue in Strong AI. A different sense of the term is meant. Unfortunately, Strong AI does not provide a positive account of what the exact content of this other sense is. According to Strong AI and the CTM, it is not clear either: (i) what a computation is, or (ii) in what sense a computation is syntactic. Searle's argument takes advantage of this lack of clarity and substitutes their logical and linguistic equivalents. However, lack of clarity does not mean that this notion of syntax is defective or identical to the notion of syntax in logic and linguistics. These are claims that the advocate of Strong AI should reject. Although it is not clear what the exact content of the notion of syntax is, it is clear that it goes beyond the notion of syntax in logic or linguistics in a number of ways. For example, the advocate of Strong AI typically holds that syntax has causal powers: the syntactic properties of one computational token literally cause the occurrence of another computational token.[20] Searle, unsurprisingly, finds this incomprehensible. He insists that syntax cannot have causal powers.[21] If the concept of syntax were that of logic and linguistics, then Searle would be right, but something different is meant.

In the following chapters, an account of the conditions under which two computations are the same or different is provided. In Searle's terminology, this is an account of syntactic identity. This account goes some way to explaining the sense in which computation is syntactic. Among other things, this shows that any plausible account of syntactic identity presupposes a number of semantic notions. Syntax, in the context of the CTM, is not the strictly non-semantic notion that Searle assumes.

1.3.10 Turing's definition of algorithm

Objection 10. *You say that the Chinese room cannot perform certain brain-like computations. But Turing's original definition of computation defined computation as what could be achieved by a man working by himself. Therefore, the man inside the Chinese room can run any program.*

Turing's (1937) original article characterised an algorithm as a procedure involving a finite number of steps that could in principle be carried out by a human unaided by any machinery save pencil and paper, and demanding

[20] For example, Fodor (2000), 17.
[21] Searle (1992), 215.

no insight or ingenuity on the part of the human. This sounds suspiciously like the situation inside the Chinese room. For the purposes of this objection, make whatever changes are necessary to Searle's setup to make it identical to Turing's. It now appears that my claim that there are algorithms that the Chinese room cannot run is false. The Chinese room can, by definition, run any algorithm. Therefore, Searle's argument avoids my criticism. It is worth looking at responses to this argument in detail.

Worst case scenario

What if this objection is fatal to my argument? First, note that this defence of Searle's argument does not challenge my claim about the architecture dependence of algorithms. The argument is that the Chinese room can run any algorithm, not because algorithms are hardware or architecture independent, but because the architecture of Chinese room happens to be just that one that can run any algorithm. If Searle had put any other machine inside the Chinese room (a Turing machine, a register machine, an electronic PC with unbounded memory), then my argument would have gone through.

Second, if this objection goes through, then it shows that the Chinese room argument is strong in an unusual direction. It not only entails that my criticism fails, it also entails that connectionist criticisms fail. Connectionist criticisms claim that one can avoid Searle's conclusion by switching to a different architecture (connectionist instead of von Neumann). My original criticism could be understood as providing a rationale for such a switch. However, if the current defence is correct, then all of this is wrong. Any criticism that relies on alternative architectures cannot succeed. This is because connectionists, to the extent they admit that they are contesting the same proposition as Searle—to the extent that they admit that the proposition at issue is whether understanding consists in the running of a particular program—and to the extent that they accept Turing's definition of program, have to admit that the Chinese room can, by definition, run any program. Hence, the Chinese room can run any program that a connectionist system can run. The advocate of connectionism might protest that a connectionist system is in many ways unlike a Chinese room. But this is irrelevant unless accompanied by a rejection of either the thesis that running a program is sufficient for understanding (hence conceding that Searle is correct), or a rejection of Turing's definition of a program.

Third, Searle himself does not seem to be aware of this defence. Searle makes little or no mention of the match even though it would significantly strengthen his argument. Furthermore, the match between the two situations is not exact, some changes need to be made to make Searle's setup fit Turing's.

It would have cost Searle nothing to have made these changes in his original article, but he did not.[22]

Finally, Searle's setup with the man inside the room is commonly thought of as motivated on other grounds, namely, the man was introduced to enable one to empathise with the system carrying out the computation. The Chinese room argument relies on one imagining *oneself* inside the room, shuffling the Chinese symbols around and not understanding Chinese. Issues about empathy seem largely orthogonal to those concerning the definition of an algorithm. One could imagine a world in which algorithms were defined differently—for example, in terms of what a group of humans could achieve, or what certain mechanical devices could achieve—and yet the Chinese room argument would still require a single human being in order to be convincing.

I hope that this shows that even if the current objection is fatal, my original criticism is still of some interest. I shall now argue that this defence of Searle's argument fails.

Criticism of the defence

This section argues that: (1) it is not clear that Turing gave the informal definition of algorithm used by Searle's defence; it is compatible with what Turing wrote to hold a different view and one not conducive to Searle's argument. (2) On this alternative view, there are algorithms that the Chinese room cannot run (e.g. parallel algorithms). (3) Even if one were to accept the premise of the defence, it is not clear that it would ultimately help Searle's argument.

(1) Turing and Church can be consistently read in a way that does not support Searle's defence. Turing and Church were primarily interested in constructing a formal predicate for picking out a certain class of functions, namely, computable functions. That class of functions can be characterised in a number of different ways. One way is in terms of the informal definition given above. Another way is in terms of the formal predicates introduced by Church and Turing. But there are plenty of other ways, both formal and informal, of picking out this class. There is no reason to take the setup described by Turing as particularly fundamental or primary. It is just one way among many of characterising that class. If one likes, the situation can be taken as definitive of what functions can be computed, but it should not be taken as the definitive word on what an algorithm is. What is special about the setup with the clerk working by hand is that it was useful for Turing in arguing that his formalised predicate (i.e. Turing machines) ends up capturing the same class of functions as are captured by our informal notions of computation.

[22] Perhaps not quite nothing, as we shall see in the next section.

(2) On at least some levels, it seems plausible that there are algorithms that a human working by herself cannot run. A human working by herself can compute any computable function, but that does not mean that she can use any possible method to do so. There may be methods that she cannot access but that do not happen to produce any new functions. Consider parallel search algorithms, or data-flow algorithms, or topological algorithms, or quantum computing algorithms, or enzymatic algorithms.[23] It seems plausible that a human being working by herself cannot run these algorithms. Humans can run related algorithms that compute the same functions, but these are different algorithms in each case.

One might object: Why call the two algorithms different? Why not say that the serial and parallel algorithms are different versions of the same algorithm? The problem with this line of argument is that it only works as a defence of Searle if one already has a clear view on how algorithms should be individuated. What criteria should we use to decide when two algorithms are the same or different? One solution is to individuate algorithms extensionally, i.e. by the function they compute. This provides a simple, clear way of individuating algorithms. But individuating algorithms extensionally is no good for Strong AI or the CTM. It would collapse those theories into a form of behaviourism. What is distinctive about Strong AI and the CTM is that they care about differences between algorithms beyond differences in their I/O behaviour.

The question then arises of what features, over and above I/O behaviour, should count as making two algorithms the same or different. Should algorithms that differ in serial–parallel structure count as different? The answer is unclear. The reason that it is unclear is that it is unclear according to Strong AI and CTM *what it is about running a particular algorithm* that is supposed to produce understanding. Maybe serial–parallel differences are significant, no one knows. In this sense, Strong AI is an inchoate position. It is perhaps unsurprising that Searle is unable to refute it.

The problem can be summarised as follows. Turing's definition of an algorithm has difficulties with certain kinds of processes that we would intuitively like to call algorithms. For example, it has difficulties with parallel processes. One of two approaches can be taken: (i) admit that there are algorithms that a human working by herself cannot run; or (ii) say that these processes are actually the same as algorithms that she can run. Approach (i) concedes that Searle's defence fails as a defence of the Chinese room argument against my original criticism. Approach (ii) opens up a new and difficult area. It relies on a notion of algorithm individuation. Individuation conditions for algorithms are hard to come by. A simple approach, such as individuating

[23] For a discussion of enzymatic algorithms, see Barrett (2005).

algorithms extensionally is not going to work. However, it is unclear how to improve on it. Strong AI and CTM leave the issue entirely open. Maybe such things as serial–parallel differences do matter. If Searle wishes to use Turing's definition in his defence then he had better have an acceptable notion of algorithm individuation ready, and it is not clear how he could have this.

(3) Even if Turing's definition can be applied in this context, it is not clear that it would ultimately help Searle. Changes need to be made to Searle's setup in order to match the situation described by Turing. One of these changes is that the rule-book, instead of containing three columns of Chinese symbols and a rewrite rule, can now contain any instructions in English that can be followed by the man inside the room without undue insight or ingenuity. This gives a lot more room for manoeuvre to a critic who argues that the man inside the room *can* understand Chinese. In the original thought experiment, it was intuitively plausible that the man could not understand Chinese no matter what Chinese characters were put in the three columns, or how complicated the rewrite rule. But now that almost any instruction in English can be given, the situation is not so clear. Why not give the man a Chinese–English dictionary and compositional theory of meaning? Davidson has indicated that such a theory of meaning could be suitably extensional and followed without undue insight, provided that one has a pre-existing grasp of the meta-language (English in this case), and Searle explicitly allows us to assume that this is true.[24] If Searle wishes to exclude this sort of case then he has to move away from the situation described by Turing, and if he moves away from that situation then he opens himself up to my original criticism.

1.4 Conclusion

The Chinese room argument fails as a general argument against Strong AI. The advocate of Strong AI can justifiably reject the Chinese room argument because it only applies to those programs that can be run on a Chinese room: the programs that she thinks are constitutive of understanding are not even considered. The reason why the Chinese room argument fails is that the Chinese room cannot run certain programs. However, we saw that the factors that determine *which* programs a system can run are far from clear. The best we could do was argue that there are certain programs that the Chinese room clearly cannot run, and brain-like programs are in that class. The rest of this thesis provides a finer-grained treatment of these issues. It provides a finer-grained account of: (1) the conditions under which two systems perform the

[24] Searle (1980b), 418.

same computation, and (2) how the physical makeup of a system relates to its computational identity.

Chapter 2

The problem generalised

This chapter generalises the problems raised in the previous chapter. This is done in two ways. First, the focus is widened from programs to computation in general. Second, it is shown how wider issues than the Chinese room argument depend on the nature of computational identity. Lack of clarity about the nature of computational identity has fueled anti-realism about computation. A consequence of anti-realism about computation is that cognitive science would be unable to provide a naturalistic account of the mind. Three major anti-realist arguments are introduced: Searle's, Putnam's, and Kripke's. A number of realist responses are considered. I argue that none of these existing realist responses is satisfactory.

2.1 Programs and computation

Not every computation involves running a program. Some computers, instead of running a program, have the computation that they perform hard-wired. The former kinds of machine are called 'general purpose', the latter are called 'special purpose'. A general purpose computer has two kinds of input: program and data. A special purpose computer takes only data as input. General purpose computers are familiar to us in the form of desktop PCs. The attraction of a general purpose computer is that it can perform a variety of different computations without requiring physical rewiring. However, there is nothing essentially general purpose about the notion of computation. The computations that a desktop PCs performs could equally well be performed by a series of special purpose devices (e.g. a word processor, a calculator, and so on). Examples of special purpose computers can be found inside modern cars, aeroplanes, and household appliances. These devices contain integrated circuits ('chips') that compute functions, but (generally) lack a programming input.

The computations that such devices perform can only be modified by physical rewiring. Another example of a special purpose computer is an desktop calculator. Most desktop calculators cannot be reprogrammed to use, say, Polish notation instead of infix notation, or to play chess instead of adding numbers. One is stuck with the computational design that has been hard-wired.

In the previous chapter, the discussion was phrased primarily in terms of general purpose computers. This was partly motivated by the way in which Searle phrased his argument, and partly by our greater degree of familiarity with such machines. However, nothing in the argument depended on general purpose computers. The points raised in Chapter 1 apply equally to special purpose and general purpose computation. For both special and general purpose computation, it is unclear: (i) under what condition two systems perform the same computation, and (ii) how the physical makeup of a system affects its computational identity.

I shall use the general term *computation* to include both special purpose and general purpose computation. In what follows, the discussion will be carried out in terms of this more general notion. Therefore, talk of *running a program*, which is specific to general purpose computers, will be eschewed in favour of *performing a computation*, which applies to both.

As a side point, it not obvious whether the CTM should say that the human brain is a special purpose or a general purpose computer. Either option is conceivable. It is conceivable that the computations that the brain performs can be modified by providing it with a special kind of programming input. It is also conceivable that the brain contains dedicated and non-programmable devices whose computation can only be changed by physical rewiring.[1] The CTM is, at least from the standpoint of our current knowledge, neutral between these two options. It is therefore misleading to characterise the CTM by the slogan 'the mind is to the brain as program is to hardware'. This is for two reasons. First, the hardware/program distinction only makes sense for general purpose computers, and as we have said, the CTM is not committed to the brain being a general purpose computer. Second, even if the brain *is* general purpose computer, there seems no reason to think that the mind should be its program. Being a general purpose computer means that the computation performed can be modified by providing a special kind of input. There seems no reason why the CTM should be committed to the claim that this special input to the brain is *the mind*.

[1] 'Programmable' is restricted here to mean that the function computed can be changed by providing a different input. This does not include, for example, the sense in which our genes 'program' for our brain.

2.2 The threat of anti-realism

The nature of computational identity and the relationship between computational identity and the physical world has wider implications than just Searle's Chinese room argument. Searle (1992), Putnam (1988), and Kripke (1982) have argued that consideration of these notions shows that we should be anti-realists about computation. According to them, computation, including those computations performed by the brain, are not objective features of the world, but a reflection of our interests and values as interpreting agents. This anti-realism has serious consequences for cognitive science's ambitions to provide an objective account of the mind.

I would like realism and anti-realism to be understood in this context in terms of truthmakers. If one is a realist about a claim X, then one believes that the facts that make that make X true are mind-independent. If one is an anti-realist about a claim Y, then one believes that the facts that make Y true are mind-dependent. Suppose that one is a realist about certain claims about electrons. One might think that the facts that make the claims 'electrons exist' and 'an electron has a charge -1.602×10^{-19} C' true are mind-independent. A realist does not claim that the *meaning* of these claims is mind-independent. She can admit that the meaning of these claims, like the meaning of any claim in natural language, is mind-dependent. The distinctive claim that a realist makes is that *once the semantic content of the claims has been fixed*, then the facts that make that semantic content true or false are mind-independent. In the case of electrons, this means that once the semantic content of 'electrons exist' and 'an electron has a charge -1.602×10^{-19} C' has been fixed, then *it does not matter what beliefs, interests, and values we have*, the propositions expressed are true or false independently of us. In contrast, the claim 'a starry night sky is beautiful', if true at all, seems to be made true in a different kind of way. Conceivably, if human beings did not have particular beliefs, interests, and values, then this claim would be false instead of true. The claim depends, even after its meaning has been settled, on human beliefs, interests, and values. The truthmakers of the claim are mind-dependent.[2]

It seems plausible that we should be realists about some areas of discourse and anti-realist about others. Searle, Putnam, and Kripke argue that we cannot be realists about *computation talk*. According to them, computation cannot be a mind-independent feature of the world. Before considering their arguments, let us consider what consequences this conclusion would have for cognitive science.

[2] For general discussion of the notion of truthmaking, see Armstrong (2004); Fox (1987); Mulligan et al. (1984); Restall (1996).

Cognitive science explains how cognitive processes work in terms of computation. According to cognitive science, some cognitive processes literally are computations. What is the point of explaining cognition in terms of computation? In part, such explanations provide useful knowledge for manipulating cognitive processes. But the main motivation for such theories is to provide a naturalistic account of cognitive processes. Advocates of cognitive science claim that the distinctive breakthrough of their discipline is that it can naturalise the mind. Instead of explaining mental phenomena in terms of other mental phenomena—which we have done for millennia—we can explain mental phenomena in terms of objective features of the physical world, namely, computations performed by the brain.[3]

Let us consider three consequences, for cognitive science, of anti-realism about computation.

First, if anti-realism about computation is correct, then cognitive science has to give up its claim to naturalise the mind. The central notion by which cognitive science explains the mind—computation—turns out, not to be an objective feature of the world, but to be a reflection of our own minds. Therefore, instead of explaining the mind in terms of objective features of the world, cognitive science explains the mind in terms of, *inter alia*, other mental states. Cognitive science does not explain mentality in non-mental terms, it *presupposes* mentality in order to explain how mental life is possible. In this respect, cognitive science is no different from many other, less respected, forms of psychological explanation, such as folk psychology or psychological explanation in novels. Cognitive science has to withdraw the claim that originally made it so interesting: its claim to naturalise the mind.

Second, if anti-realism about computation is correct, then cognitive science cannot explain an important class of mental processes. According to anti-realism about computation, some mental processes are partially constitutive of a system performing a computation. Call these the 'interpretative' mental processes. If cognitive science aims to give an account of interpretative mental processes, then it cannot do so by appeal to the notion of computation. The reason is that what it is to perform a computation is, *inter alia*, that these interpretative processes obtain. A cognitive science theory that explained interpretative processes in terms of computation would, illegitimately, be appealing to the very processes it was trying to explain: its *explanans* would contain its *explanadum*. Cognitive science explanations, if they are to avoid circularity, must therefore be restricted to *non*-interpretative mental processes. However,

[3]For example: 'Rationality is a normative property; that is, its one that a mental process ought to have. This is the first time that there has ever been a remotely plausible mechanical theory of the causal powers of a normative property. The first time ever.' (Fodor, 2000, 19).

interpretative processes play an important part of our mental life beyond the role described above, and *prima facie* seem like good candidates for cognitive science explanation.

Third, a lesson that one might draw from anti-realism about computation is that cognitive science is not on a par with the other sciences. Intuitively, one might think that physics, chemistry, and physiology describe the world in a mind-independent way—if there were no minds, then the facts about physics, chemistry, and physiology would still largely be the same. According to the anti-realist, cognitive science is different. If there were no minds, then there would be no facts about computation. Facts about systems performing computations depend, for their very existence, on agents to interpret them. If, as seems reasonable, one thinks that a science should aim to describe the world in a mind-independent way, then this means that cognitive science cannot be a science.[4]

Two points should be noted.

First, although anti-realism about computation would damage the ambitions of cognitive science, it by no means entails that cognitive science is worthless. An explanation of mental phenomena in mental terms is still useful. For example, explanations in folk psychology are useful. The careful and controlled nature of studies in cognitive science would enable it to go beyond folk psychology to explain how our mental states are related in non-obvious ways. Anti-realism about computation does not entail that cognitive science is worthless, it just entails that cognitive science cannot deliver what we originally thought: a naturalistic account of the mind.

Second, one might wonder whether anti-realism about computation only causes trouble for someone who already has a general realist outlook. If one is a global anti-realist—an anti-realist about all claims—then perhaps it does not matter if anti-realism about computation is true. However, this is not the case. A global anti-realist would not be affected by the first and the third consequences mentioned above, but she would be affected by the second consequence. In other words, a global anti-realist would face the circularity problem that certain cognitive processes cannot be explained in terms of computation because those processes are constitutive of what it is to perform a computation. Therefore, anti-realism about computation has consequences even for a global anti-realist.

[4]Searle argues for this: 'The aim of natural science is to discover and characterize features that are intrinsic to the natural world. By its own definitions of computation and cognition, there is no way that computational cognitive science could ever be a natural science, because computation is not an intrinsic [mind-independent] feature of the world.' (Searle, 1992, 212).

Two senses of mind-dependence

A trivial and substantive sense of mind-dependence should be distinguished in this context. There is a trivial sense in which the facts that make the claims of cognitive science true are mind-dependent: they are *about* minds. This is clearly not the sense of mind-dependence at issue in the realist/anti-realist dispute. Both the realist and the anti-realist agree that the claims of cognitive science are mind-dependent in *this* respect. What they disagree about is whether such claims are mind-dependent in a further respect: namely, whether the *general* concepts under which cognitive science explains the mind make ineliminable reference to mental states. Cognitive science claims that certain mental processes are computations. If the notion of computation makes ineliminable reference to mental states, then this kind of explanation is not explanation in non-mental terms. However, if, as the realist claims, the notion of computation does not make reference to mental states—if it refers to general features of the non-mental world—then this kind of explanation *is* explanation in non-mental terms, and cognitive science can justify its claim to provide a naturalistic account of the mind. It is concerning *this* question of mind-dependence that the realist/anti-realist dispute lies.

Another way of drawing the distinction—a way that is helpful to consider in many cases—is to ask *whose* mental states are the truthmakers of the claims of cognitive science. Typically in cognitive science, there is an experimenter (the cognitive scientist) and an experimental subject (the person under examination). Both the realist and anti-realist agree that the mental states of subject are part of the truthmakers for cognitive science claims about that subject. However, the anti-realist claims that the *experimenter's* mental states are also part of the truthmakers for such claims. According to the anti-realist, the truth of a computational claim depends on the experimenter having certain beliefs, interests, and values. In contrast, the realist claims that whether the subject performs a computation is independent of the beliefs, interests, and values of the experimenter. It does not matter what mental states the experimenter has, or whether an experimenter exists at all, it is simply an independent matter of fact whether the experimental subject performs a certain computation or not. Unfortunately, this way of drawing the distinction, although helpful in many cases, does not work in general. This is because it is possible for the subject to be her own experimenter. In this case, there is no easy way of drawing a distinction between the mental states of the subject and the mental states of the experimenter. This is why I wish to draw the distinction in the terms discussed above.

I identified the relevant sense of mind-dependence above with whether

the general concepts make ineliminable reference to mental states. However, I wish to refine this way of phrasing the distinction. It is not as a question about reference that I would like to phrase the dispute, but as a question about truthmakers. Talk of the reference in the context of general concepts is unclear. It is better to rephrase the issue in terms of truthmakers. The sense of mind-dependence can be rephrased in terms of truthmakers as follows. Cognitive science claims that certain mental processes are computations. If one is a realist about computation, then one thinks that the truthmakers of computation talk are mind-independent—they do not involve mental states. If one is an anti-realist about computation, then one thinks that the truthmakers of computation talk are mind-dependent—they do involve mental states. If the truthmakers of computation talk are ineliminably mind-dependent, then an explanation of mental states in terms of computation is not an explanation in non-mental terms, and therefore cognitive science cannot justify its claim to naturalise the mind. In contrast, if the truthmakers of computation talk are ineliminably mind-independent, then an explanation of mental states in terms of computation *is* an explanation in non-mental terms, and cognitive science *can* justify its claim to naturalise the mind. It is in these terms—in terms of the truthmakers of computation talk—that the question of mind-dependence should be phrased.

Let us now consider the arguments that aim to show that realism about computation is impossible.

2.2.1 Searle's argument

Searle (1992) gives two arguments against realism about computation.[5]

Syntax is not intrinsic to physics

Searle (1992)'s first argument is a *reductio ad absurdum* of realism about computation. He claims that any plausible form of realism about computation entails that all physical systems perform all computations. This kind of realism about computation could not be tolerated by cognitive science. Cognitive science claims that cognitive processes are *distinctive* computations performed by the brain. If all systems perform all computations, then there are no distinctive

[5]Searle provides two further arguments in Searle (1992): 'Syntax has no causal powers' and 'The brain does not do information processing'. These arguments will not be addressed here. Searle's claim about syntax and causal powers seems to result from conflating the notion of syntax in linguistics with the notion in implementation of computation. His second argument, the claim that the brain does not perform information processing, does not seem essential to the CTM. The content of that claim, unless taken in the sense of mathematical information theory, is obscure, and an advocate of the CTM need not hold it in order to say that the brain performs computations.

computations performed by the brain. The brain, like everything else, performs all computations. Therefore, computational theories cannot explain why humans have certain cognitive processes rather than others, or why they have cognitive processes at all.

Searle's argument is as follows. Suppose that the performance of a computation is a matter of a system having a certain pattern of mind-independent activity. Now imagine a desktop computer running Microsoft Word. There are many patterns of activity inside the computer: patterns of electrical activity, thermal activity, vibrational activity, and so on. According to our realist assumption, the computer runs Microsoft Word because one of these patterns of activity—the pattern of electrical activity—has a particular structure. If one were to construct another system, perhaps made out of different materials, that had a pattern of activity with the same structure, then it too would run Microsoft Word. Now consider a brick wall. A brick wall is teeming with microscopic activity: vibrational activity, thermal activity, atoms changing state, subatomic particles flying around, and so on. In fact, there is *so much* activity inside a typical wall that there is almost certain to be at least one pattern of activity with the same structure as that inside the computer. Therefore, according to our realist notion of computation, the wall is also running Microsoft Word. The same point applies, *mutatis mutandis*, to other physical systems. Therefore, (almost) every physical system performs every computation.

Of course, this cannot be true—the conclusion is absurd. However, it is not easy is to explain what has gone wrong without giving up realist principles. For example, a natural reaction is to say that the wall does not run Microsoft Word because walls cannot be used as word processors—it does us, as human users, no good that there is a pattern of activity inside the wall that has the same structure as Microsoft Word, because we do not know where that pattern is, or how to use it. This response rules out the absurd conclusion, but at the cost of giving up realism about computation. It makes the performance of a computation dependent on human interests and values. Searle claims this kind of response, which introduces anti-realist factors, is the only way out for computation talk. Unadulterated realism about computation has absurd consequences.[6]

[6]Searle sometimes phrases his argument in two different ways. (1) He sometimes draws attention to the intuition that we can *choose* to interpret a physical system like the wall as performing any computation we like. This is misleading for two reasons. First, it would do nothing to move a realist who claims that there is a fact of the matter about which computation a system performs and we might get this fact right or wrong in our choices. Second, it is not reasonable to assume that we can choose to interpret the wall in any way we like, since even on an anti-realist view, the claim that we have absolute freedom to choose our own values and interests is implausible. (2) Searle sometimes presents his argument as the claim that computation is observer-relative: computations are *assigned to* the physics not *discovered within* the physics. This however, merely states the anti-realist position, it does not provide any argument for it.

The homunculus fallacy

A potential weakness in Searle's *reductio ad absurdum* argument is that it leaves open the possibility that the realist can develop a more sophisticated account of computation that avoids Searle's conclusion. For example, the realist might claim that only certain patterns of activity, such as counterfactual-supporting patterns of activity, count as computation.[7] Searle acknowledges this possibility and argues that realism about computation fails for a deeper reason.[8] This reason in expressed in Searle's second argument.

Searle's second argument against realism about computation revolves around the claim that computation requires an interpreter. When we speak of computation without an interpreter—for example, when we say that computations are performed by our brains without us being aware of them—we commit a fallacy. We implicitly assume the existence of an interpreting homunculus.

An example of a cognitive process that has been claimed to be a computation is the recognition of simple shapes by the visual system. Marr (1982) describes a series of computations that transform a two-dimensional array of activity on the retina into a three-dimensional description of a scene. However, as Searle points out, the inputs and outputs of a computation require interpretation. A component of the visual system only performs, say, line recognition, if its input is interpreted as visual data, and its output as judgements of lines. Who interprets the inputs and outputs of cognitive processes? Where is the interpreting agent?

In the case of electronic computers, human users are the interpreting agents in question. *We*, as users of a computer, interpret its input and output as, for example, a string of English text or a picture. However, the same cannot be true of the computations posited by cognitive science. The computations posited by cognitive science can take place without our awareness. No human need be aware that they are performing a computation when they, say, parse a grammatical sentence. So who interprets their input and output? We appear to be forced to invoke a homunculus who interprets various parts of the brain as performing computations.

A number of philosophers argue that this is not a problem, since the hom-

[7] Block (1995) makes this suggestion.

[8] 'I do not think that the problem of universal realizability is a serious one. I think it is possible to block the result of universal realizability by tightening up our definition of computation. Certainly we ought to respect the fact that programmers and engineers regard it as a quirk of Turing's original definitions and not as a real feature of computation. Unpublished works by Brain Smith, Vinod Goel, and John Batali all suggest that a more realistic definition of computation will emphasize such features as the causal relations among program states, programmability and controllability of the mechanism, and situatedness in the real world. All these will produce the result that the pattern is not enough . . . But these further restrictions on the definition of computation are no help in the present discussion *because the really deep problem is that syntax is essentially an observer-relative notion*.' (Searle, 1992, 209, emphasis in original).

unculus in question can be 'discharged'.[9] Computational operations can be analysed into progressively simpler units, until we reach simple 'yes-no', '1-0' patterns. Therefore, a high-level homunculus can be progressively replaced by stupider homunculi, until we reach a bunch of homunculi who just say 'zero, one, zero, one'. However, Searle argues that replacement by stupid homunculi does us no good. Even at the level of simple '1-0' patterns, we still need an interpreting agent to interpret 'zero' as 0, and 'one' as 1. The homunculus fallacy is with us after all.

2.2.2 Putnam's argument

The second challenge to realism about computation comes from Putnam (1988). Putnam's main concern is to show that propositional attitudes, such as *believing that snow is white* cannot be computational states of the human brain.[10] According to Putnam, propositional attitudes cannot be computational states of the human brain because the content of propositional attitudes is not reducible to computational or physical properties and relations. There are a number of sophisticated arguments that Putnam gives to support this claim.[11] However, it is important to see that even if Putnam's claim is correct, that claim does not have a direct bearing on this thesis. This thesis is concerned with cognitive *processes*, not propositional attitudes. It is possible for cognitive processes to be computational even if, as Putnam argues, propositional attitudes are not. There are at least two ways in which this could happen.

First, propositional attitudes could participate in computations, even if they get their nature and identity in non-computational ways. Propositional attitudes may get their identity from their causal relations to the outside world (although Putnam independently denies this), or from the interpreting attitudes of other speakers. This does not exclude those propositional attitudes from participating in computations. Furthermore, there is no reason to think that computational explanations cannot be given of topic-neutral features of our inferences even if the content of propositional attitudes cannot be similarly explained. Putnam's target is not the claim that the brain performs computations, or the claim that the brain performs computations with propositional attitudes. His target is the claim that the computations performed by the brain

[9] For example, see Block (1995); Dennett (1978a); Haugeland (1981).

[10] For a defence of the opposite position, see the earlier works of Putnam (1975b), Chs. 14, 18–21.

[11] These include: (i) that a significant part of content lies outside the head; (ii) content presupposes human rationality and interpretative practice, and this cannot be captured by a single computational relation; (iii) it is intuitively plausible that two systems can share the same propositional content and yet be in distinct computational states; and (iv) a system would need to both know all the physical facts about the world, and to survey all possible rationalities, near-rationalities, not-too-far-from-rationalities-to-be-still-intelligible, in order to work out if two computational states have the same content. See Putnam (1988), Chs. 2, 3, 5, 6.

determine the content of its propositional attitudes. Strictly speaking, his argument is not against the CTM *per se*, but against an application of the CTM to provide an account of content.

Second, the cognitive processes explained by the CTM need not involve propositional attitudes at all. Theories in cognitive science often concern relatively modest cognitive process, such as the detection of edges by our visual cortex or syntax parsing. These cognitive processes need not involve full-blown propositional attitudes. Therefore, one can claim that certain cognitive processes are computational without making any claim at all about propositional attitudes. On a similar point, Fodor (1983) explicitly restricts computational theories of the mind to peripheral parts of the cognitive system; parts that are, not coincidently, prior to the construction of full-blown propositional attitudes.

However, in the course of his argument, Putnam establishes a result that does have a bearing on this thesis. This result is that every ordinary open physical system implements every abstract finite state automaton.[12] Putnam states the result for finite state automata, but he claims that a similar result holds for any other computational formalism. Therefore, every ordinary open physical system performs every computation. As was argued above, if this were true, then it would be a *reductio ad absurdum* of the notion of computation.

Putnam's theorem

The proof of Putnam's theorem is as follows. Pick any open physical system, S. An open physical system is system that is not shielded from outside influences. Open systems are vulnerable to the influence of naturally occurring clocks. It seems reasonable to suppose that such clocks exist, and that their effects (perhaps gravitational or electromagnetic) propagate inside and affect open systems. Assume that the changes they induce inside the system are of a non-cyclical nature. Therefore, we can assume that the open system S is in different states at different times. (Putnam calls this the 'Principle of Noncyclical Behaviour').

It can be shown that open system S implements all finite state automata. For the sake of simplicity, pick a finite state automaton that has two states, A and B, and that undergoes the following state transitions: $ABABABA$. Suppose that the system is observed for a 7-minute interval from 12:00 to 12:07. It can be shown that system S implements the finite state automaton during this time period.

Let $\mu(S, t)$ denote the maximal physical state of S at time t. The parameter $\mu(S, t)$ is an entire characterisation of the physical state of S at t. In classical

[12] See Putnam (1988), Appendix.

physics, $\mu(S,t)$ is the value of all the field parameters inside the boundary of S at time t. We assumed above that system S is in a different physical state at different times. Therefore, the value of $\mu(S,t)$ is different for different t. If one were to plot all the possible values of $\mu(S,t)$, one would produce a 'phase space' for S. The phase space is the space of all possible states of S. As time increases, S traces out a path in its phase space. Regions of S's phase space pick out ranges of possible states that S could have.

Divide up the time period between 12:00 and 12:07 into seven 1-minute intervals. Let $t_1, t_2, \ldots, t_7$ denote the beginning of each 1-minute interval, $t_1 =$ 12:00, $t_2 =$ 12:01, $\ldots$, $t_7 =$ 12:06, and let t_8 denote the end of the entire 7-minute period, $t_8 =$ 12:07. For each of the time intervals, define the 'interval state' s_i to be the region in phase space that contains all the states $\mu(S,t)$ for $t_i \leq t < t_{i+1}$, $i = 1, \ldots, 7$. In other words, an interval state s_i is the region in phase space occupied by the system during a 1-minute interval from t_i to t_{i+1}. We will say that the system is in s_i at time t just in case $\mu(S,t)$ is contained in the region of phase space s_i. Therefore, system S is in s_1 from t_1 to t_2, in s_2 from t_2 to t_3, $\ldots$, and in s_7 from t_7 to t_8.

Now define $A = s_1 \vee s_3 \vee s_5 \vee s_7$, and $B = s_2 \vee s_4 \vee s_6$. System S is in state A from t_1 to t_2, from t_3 to t_4, from t_5 to t_6, and from t_7 to t_8, and in state B at all other times between t_1 and t_8. Therefore, the system is in the following sequence of states during the 7-minute period: $ABABABA$. The disjointness of each of these states is guaranteed by the Principle of Noncyclical Behaviour. Therefore, the system satisfies the sequence of state transitions specified by the finite state automaton, with each machine state implemented by a distinct range of physical states.

It remains to be shown is that each implemented state A or B 'brings about' its subsequent state. Putnam argues that this can be shown by demonstrating that each state A or B 'causes' the subsequent state. The notion of causation with which Putnam operates is as that a state X causes a state Y, just in case a mathematically omniscient being (a being who knew all the laws of physics and could calculate all their consequences) could *predict* that the system would go into state Y at the relevant time given only the information that the system was in state X and the boundary conditions of the system. More precisely, a state X of a system causes that system to go into state Y just in case any maximal state of the system which lies in the region of phase space corresponding to state X, and which is compatible with the boundary conditions at the moment of transition, and which is compatible with physical law, will be followed by a state which lies in the region of phase space corresponding to state Y.

It can be shown that this condition is satisfied by system S. Consider a mathematically omniscient being who knows that the system is in state A at

time t for $t_1 \leq t < t_2$, and who also knows the boundary conditions B_t at t. The omniscient being can deduce from this that the only possible state for the system to be in at time t is $\mu(S, t)$. The deduction requires the use of a lemma that is discussed and justified in Putnam (1988).[13] Let us assume, for the sake of argument, that this lemma is correct. Then, the result is established. Given the information about the boundary conditions at t, a mathematically omniscient being can work out that the system is in $\mu(S, t)$ state. If this can be done, then the being can determine, given boundary conditions at subsequent times and the laws of nature, how system S evolves in the whole time interval. Therefore, the A and B states of system S satisfy the above condition for causation.

The technique of proof above is not specific to any particular finite state automaton. The same reasoning can show that system S implements *any finite state automaton one pleases*. Therefore, we can conclude there are no distinctive computations that a physical system performs. Physical systems, considered just in terms of their physical parameters, perform all computations. If the mind-independent facts do not outrun the physical facts (which seems plausible), then this means that a mind-independent notion of computation is not available. In terms of the mind-independent facts, all computational claims are trivially true.[14]

2.2.3 Kripke's argument

Kripke (1982) presents a paradox concerning rule-following and meaning. Kripke describes a sceptic who argues that no facts about past history determine the correct way in which to continue to follow a rule. According to the sceptic, any future course of action is compatible with past stipulations or intentions. Meaning requires accord or discord with past intentions concerning words. Therefore, the sceptic argues, there are no facts about meaning. The same argument applies to mental content. The paradoxical conclusion is that there are no facts about meaning or mental content.

[13] Briefly, the lemma states that a system S cannot have the *same* internal state μ at times t and t' but *different* boundary conditions. The reasoning relies on what Putnam calls the 'Principle of Continuity' which states that the electromagnetic and gravitational fields are continuous (except possibly at a finite or denumerably infinite set of points). Putnam's justification of the lemma is as follows. If a system S is exposed to signals from a clock C, then those signals can be thought of as forming an 'image' of C on the surface of S. For the same reason, there are also 'images' of C *inside* the boundary of S. The 'image' of C at $t' = 12$ might be thought of as showing a 'hand at the 12 position', while the 'image' of C at $t = 11$ might be thought of as showing a 'hand at the 11 position'. Therefore, at $t' = 12$, the system will have a '12 image' on its boundary and an '11 image' an arbitrarily small distance inside its boundary. This means that the fields that constitute the 'images' would have a discontinuity along a continuous area, violating the Principle of Continuity.

[14] Note that the restriction to open physical systems is not a serious one; all physical systems that we are likely to consider are open.

As part of the argument, Kripke considers the case of machine rule-following.[15] Computation and rule-following are closely connected: in order for a machine to perform a computation that machine must follow a rule. Kripke consider what fact about a machine determines that it follows one rule, i.e. performs one computation, rather than another. He argues that there are *no mind-independent* facts of the matter. The computation that a machine performs, if it performs a computation at all, depends on the intentions of its designer or the interests of its users. Computation is a mind-dependent matter. Kripke goes on to argue that there are no facts about the intentions of designers or interests of users either. Therefore, there are no facts about rule-following, either for humans or computers. Here, however, we will not follow Kripke's argument. We shall only consider the first step in the argument: the claim that computation is mind-dependent. The further aspects of Kripke's paradox will not be considered. It is generally assumed that there must be some answer to the sceptic on these points. In any case, the problem has been extensively addressed elsewhere.[16]

Kripke gives three arguments for the mind-dependence of computation.

Kripke's first argument

Kripke's first argument is that the input and output of a machine require interpretation. A machine is a physical system, and in order for that machine to follow a particular rule, a 'code' or 'machine language' has to be used to interpret its physical states. For example, consider a machine that appears to compute the plus function. This machine, by itself, is no more than a physical system with a variety of physical states. It computes the plus function only if one interprets some of its physical states as codes for numbers. The coding system is not something that is determined by the machine. Instead, the coding system is chosen by an interpreting agent who wishes to use the machine for certain purposes. A coding system is necessary for a machine to perform computation. Therefore, there can be no facts about the computation a machine performs independent of interpreting agents.

Kripke's second argument

Kripke's second argument is based on an observation about the finite nature of real-world machines. We often treat a real-world machine as performing a computation with an infinite domain. For example, we might say that a machine computes the plus function. The machine itself cannot be the truthmaker for this computation claim. Real-world machines are finite; they only accept

[15] Kripke (1982), 32–37, especially n. 24.

[16] For example, see Miller and Wright (2002) and references.

finitely many numbers as input, and only yield finitely many numbers as output. An indefinite number of computational behaviours extend the actual finite behaviour of any given real-world machine into an infinite domain. The attribution of one of these behaviours, such as the plus function, to a machine cannot be due to any fact about that machine. It must be due to other factors. Kripke says that these other factors involve the intentions of the designers and the interests of the users. Computation, at least on infinite domains, must be mind-dependent.

Kripke's third argument

Kripke's third argument is that we cannot make sense of malfunction without interpreting agents. Real-world machines malfunction. Wires melt and gears slip, and this causes machines to give incorrect answers. What makes a wire melting, or a gear slipping, a malfunction? The physical nature of the machine alone cannot be responsible. A designer might exploit the fact that wires melt or gears slip to achieve an intended behaviour. A machine that was 'malfunctioning' for me would be performing perfectly well for him. There seem no mind-independent facts of the matter about who is right in this case. The machine, considered as an isolated object, behaves as it behaves; whether it is malfunctioning or not depends on the intentions of designers and users. If mind-independent malfunctioning does not make sense, then mind-independent correct functioning does not make sense either. This distinction—between correct functioning and malfunctioning—seems essential to the idea of a machine performing a distinctive computation. Therefore, performing a distinctive computation cannot be a mind-independent matter.

2.3 Existing solutions

This section provides a survey of existing realist answers to the following questions: (1) what makes a system perform one computation rather than another; and (2) what makes a system perform any computation at all. It is worth noting that the anti-realist has an easy answer to both these questions. She can say that a system performs one computation rather than another just in case we find it useful, or we choose, to interpret it as such. Similarly, she can say that a system performs any computation at all just in case we find it useful, or we choose, to interpret it as performing a computation. She is not committed to every system performing every computation, because we generally do not choose, or find it useful, to interpret brick walls as performing computations. Let us see whether a realist can provide similar answers.

(Note that the realist responses discussed in Sections 2.3.1 to 2.3.3 provide an answer to the question of computational identity—question (1) above. The realist responses discussed in Sections 2.3.4 to 2.3.6 provide an answer to the question of what makes a system perform any computation at all—question (2) above. None of the responses below provides an answer to both questions, unlike the view put forward later in this thesis.)

2.3.1 Syntactic approaches

A common way of individuating computations is to appeal to purely syntactic features of their specification. Within a given architecture, it seems possible to individuate computations by comparing the syntactic form of the specification of one computation with another. One might say that if two computations can be specified using the same symbols in the same order, then the two computations are the same. However, although this way of individuating computations might work within a single architecture, it does not work in the general case. In the context of the CTM, we need a cross-architecture notion of computation individuation: we need to be able to say when, for example, a human and a computer perform the same computation. In the cross-architecture case, there is no guarantee that the syntactic symbols in the specification mean the same thing. The symbol 'ADD' might mean *add* on one architecture, and *divide* on another. Therefore, one cannot compare computations just by comparing the syntactic properties of their specification. (It is not obvious that this strategy works even in the intra-architecture case. Two specifications of a computation can differ in their syntactic symbols, for example, they might differ by a comment character, without differing in the way in which they work. Two computations may also work in the same way, but be described, even within a single architecture, using different symbols.)

Even if one could individuate computations in terms of their syntactic specification, that does not settle the issue of realism about computation. One still has to explain *how* the symbols of an computation specification relate to the real-world machine that implements the computation. The syntactic approach to computational identity, by itself, is silent about the nature of this implementation relation.

2.3.2 Z and specification languages

Another way to individuate computations is by appeal to a specification language. Specifications languages are designed to give an unambiguous statement of what a computation does in formal/mathematical terms. One popular

specification language, Z, is based on Zermelo–Fraenkel set theory with first order predicate logic. One might say that two computations are the same just in case they satisfy the same specification in Z. Unfortunately, this strategy for individuating computations does not work either.

Specification languages like Z are designed to specify the I/O behaviour of programs, not to distinguish different ways of achieving that I/O. A specification in Z is a specification of what a system does, not a way in which it achieves its behaviour.[17] The purpose of Z is to enable formal reasoning about the I/O behaviour of a program, and in particular, to provide correctness proofs. A correctness proof is a demonstration that a program satisfies a certain Z specification and consequently that it *must* have a certain I/O behaviour. Correctness proofs are important, for example, they are important in safety and security-critical systems. However, correctness proofs do not address our question of the conditions under which two computations are the same. Two systems can satisfy the same Z specification, but still work in different ways. The same holds true of other formal specification languages.

Z does not secure realism about computation either. A Z specification consists of mathematical formulae. Such a specification, at best, makes statements about mathematical objects. It does not provide an account of how these abstract objects relate to the nuts and bolts of real-world machines. Therefore, a Z specification, by itself, does not establish realism about computation.

2.3.3 Canonical architectures

Another way of individuating computations is to appeal to a canonical architecture. The idea here is as follows. Suppose one wishes to determine whether two computations c_1 and c_2 are identical. One transforms them into two equivalent computations c_1′ and c_2′ that can be run on a canonical architecture, say, a universal Turing machine. One then compares the syntactic specifications of c_1′ and c_2′ to see whether they are the same or not. If the syntactic specifications of c_1′ and c_2′ are the same, then the original computations c_1 and c_2 are the same, if not, then c_1 and c_2 are different. Comparing the syntactic specifications of c_1′ and c_2′ should, in theory, be easy since they involve operations on a single architecture—this is setting aside the problems with intra-architecture comparisons raised at the end of Section 2.3.1.

[17] '. . . we ignore issues relating to *how* a task is to be carried out and focus instead on accurately stating *what* has to be done. One of the things that people find difficult to do when they start writing formal specifications is to stop thinking procedurally. In writing a specification it is best to think declaratively. Issues relating to efficiency and even implementability should be shelved for the time being. There is a time and a place to think about how a specification is going to be implemented efficiently, but this is best done when the specification has been finished.' (Diller, 1994, 4–5).

Pylyshyn (1984) suggests that this is the way in which computations should be individuated in cognitive science:

> In my view, two programs can be thought of as strongly equivalent or as different realizations of the same algorithm or the same cognitive process if they can be represented by the same program in some theoretically specified virtual machine. A simple way of stating this is to say that we can individuate cognitive process in terms of their expression in the canonical language of this virtual machine. (Pylyshyn, 1984, 91–92)

A similar approach is used in computer science to compare the space and time requirements of different computations: one transforms a computation to be compared into its Turing machine equivalent and measures the space and time it requires as the space and time it would take up on a Turing machine.

It might seem reasonable therefore that computation individuation in terms of a canonical architecture, e.g. Turing machines, is the true notion of computation individuation. Unfortunately, it does not work.

First, even on a sympathetic reading, the proposal uses the notion of 'transformation into the same computation', which was the notion we wished to capture. There seems no way of understanding 'equivalent' in the proposal without presupposing the supposed end product of the analysis, namely, the notion of computational identity.

Second, the proposal does not give a justification for the right canonical architecture. There seems no reason why Turing machines should be privileged. Turing machines are adopted in measurements of complexity only as a matter of convention. Why should we choose one architecture, with its concomitant notion of computation individuation, over another?

Third, even if the choice of a particular architecture can be justified, the resulting notion of computation individuation is almost certain to be too coarse-grained. We sometimes wish to make very fine-grained distinctions between computations. For instance, we might wish to distinguish between computations that fill an array left-to-right versus right-to-left. For a machine with memory access set up in a certain way, it is possible that right-to-left runs orders of magnitude faster than left-to-right.[18] It makes sense in this context to speak of two computations: a left-to-right computation and a right-to-left computation. However, since this difference is tied to the specific details of the architecture of the machine, it could well disappear in a translation to a canonical architecture such as a Turing machine. Translation to canonical architecture can fail to preserve fine-grained distinctions between computations.

[18] This can happen in real-world machines as a result of caching effects.

Why is the use of canonical architectures legitimate in measurements of complexity? First, such an account does not presuppose what it sets out to define. Complexity measurements can presuppose computation individuation without obvious circularity. Second, ranking computations in terms of their performance on Turing machines is a convention. No claim need be made that such a transformation gives insight into the true identity conditions of computations. Third, ranking computations in terms of their performance on Turing machines is only an estimate of their performance on real-world machines. For the reasons discussed above, there is no easy inference from the complexity profile on a Turing machine to a computation's performance on a real-world machine.[19]

Like the approaches discussed in the previous two sections, transformation to canonical architecture does not address the question of what makes a real-world system perform a computation—question (2) in Section 2.3 above. We shall now consider three theories that directly attempt to address this question, and to provide it with realist answers.

2.3.4 Chalmers' realist approach

Chalmers (1996) proposes a theory of performing a computation aimed at avoiding Putnam's (1988) anti-realist conclusion. Chalmers argues that Putnam makes two mistakes in his analysis. First, Putnam takes too weak a view of the modal requirements for performing a computation. Second, Putnam focuses on the wrong class of computational architectures. If these two mistakes are rectified, then the performance of a computation can be seen to be a mind-independent matter.

Chalmers begins by summarising Putnam's theory of what it means to perform a computation. According to Putnam, a physical system implements a finite state automaton (FSA) provided the following condition is met:

> A physical system implements an inputless FSA in a given time-period if there is a mapping f from physical states of the system to formal states of the FSA such that: if the system is in physical state p during the time-period, this causes it to transit into a state q such that formal state $f(p)$ transits to formal state $f(q)$ in the specification of the FSA. (Chalmers, 1996, 311)

Chalmers argues that this account needs to be supplemented in three ways. Let us consider each step in turn.

[19] Note that space and time complexity profiles by themselves do not individuate computations either. For one thing, two computations can share the same complexity profile and yet be different computations, e.g. Bubble sort and in-place Merge sort are both $O(n^2)$.

Step 1: Transitions must have modal force

As specified by Putnam, the relationship between the implementing states need not support counterfactuals. All that matters is that, given the conditions in the actual world, if the system is in state p, then it transits into state q. Nothing is required about the behaviour of the system if the conditions in the real-world were different. Chalmers argues that this fails as an account of implementation of computation for two reasons.

First, in order for a system to implement a computation, it seems a minimal requirement that its state transitions be reliable. Not only should a system perform in a certain way in the actual world, the system should have performed in that way even if the environmental conditions had been slightly different. Chalmers' second point concerns unexhibited state transitions. Not only should the state transitions that are manifested on the actual run be mirrored in the physical structure of the system, but every implemented state transition should be so mirrored. For example, if the system implements an FSA with state transitions $A \rightarrow B$, $B \rightarrow A$, $C \rightarrow D$, then even if, as a matter of contingent fact, the system always starts in A and so always transits between A and B, it should be the case that if the system were to start in state C, then it would transit to state D.

Chalmers argues, however, that increasing the modal force of the relevant conditionals does not by itself avoid Putnam's anti-realist conclusion. A damaging anti-realist result still holds. Define a 'clock' as a system that reliably transits through a sequence of states $s_1, s_2, \ldots$ over time. Define a 'dial' as system with an arbitrary number of states such that when it is put into one of those states it stays in that state come what may. Chalmers argues that every system with a clock and a dial implements every FSA. His argument is as follows. Each machine state of an FSA can be associated with a physical state [i, j] of the implementing system, where i corresponds to a unique clock state, and j to a unique dial state. If the system starts in [1, j], then it will reliably transit to [2, j], [3, j], and so on as the clock progresses. Assuming that the system starts its actual run on dial state 1, the start state of the FSA can be associated with [1, 1], and subsequent states with [2, 1], [3, 1], and so on. At the end of this assignment process, if some FSA states have not come up, then choose an unmanifested state, P, as a new start state and associate [1, 2] with it. Then associate physical states [2, 2], [3, 2], and so on with the states that would follow P in the evolution of the FSA. Continue this process until all of the unmanifested states are used up. For each state of the FSA, there will be a non-empty set of associated physical states [i, j]. Assign the disjunction of each of these states to each FSA state. The result satisfies the modally strengthened account of implementation.

Therefore, Putnam's anti-realist result is preserved albeit in slightly modified form: every open system with a clock and a dial performs every computation. It is conceivable that many open physical systems have a clock and a dial. If they do not, then a clock and a dial can be trivially added just by placing an old-fashioned clock inside the system. Another condition must be added to the requirements on computation in order for non-trivial mind-independent computation to be possible.

Step 2: Add input and output

The FSA that Putnam considers have no input or output. Chalmers argues that in addition to increasing the modal force of the conditionals describing the transitions, we should also require that the system have inputs and outputs. The inputs and outputs of a computation can, according to Chalmers, be specified in physical terms. A typical output might be to print two of the following ink-marks '1' in a row. Not just any physical state can realise this output. Therefore, an FSA with inputs and outputs cannot be implemented by any physical system.

Unfortunately, adding inputs and outputs does not by itself avoid Putnam's conclusion. As Putnam himself argues and Chalmers acknowledges, a weaker but still damaging result holds. This result is that every physical system with a given I/O behaviour implements any FSA with that I/O behaviour. We saw in Chapter 1 that the same I/O behaviour can be achieved in many different ways. A notion of computation that cannot slice computations finer than their I/O behaviour is unsatisfactory. As Putnam notes, it would collapse functionalism into behaviourism. Therefore, there must be more constraints on performing a computation than just having correct I/O behaviour.

Step 3: Switch from monadic to combinatorial state architecture

The final modification proposed by Chalmers is to move away from the FSA architecture to a more complex computational architecture. Chalmers argues that Putnam's result can be resisted for a type of machine he calls combinatorial state automata (CSA). For combinatorial state automata, there are non-trivial mind-independent conditions for performing a computation.

Note that this approach concedes that Putnam is correct about FSA, which is not an altogether happy result. Even if realism can be secured for CSA, anti-realism would still reign for FSA. Nevertheless, this conclusion can be tolerated if realism about computation can be secured *for all computational architectures relevant to the CTM*. It was originally a realist worry concerning the CTM that motivated the discussion. If this worry can be quelled, then much of the heat

would go out of the realist/anti-realist dispute about computation. For this reason, Chalmers not only claims that realism is true for CSA architectures, he also claims that CSA architectures include all those relevant to the CTM.

Combinatorial state automata are like finite state automata, except that their states have a combinatorial structure instead of a monadic structure. Instead of having a single internal state, S, the internal state of a CSA is a vector of substates, $[S_1, S_2, \ldots, S_n]$, where the ith component of the state vector is the ith substate of the system. The state transitions of a CSA are defined by specifying, for each component of the state vector, how its new value depends on the old state vector and an input vector.

A physical system implements a CSA if the following conditions are met:

> A physical system implements a given CSA if there is a decomposition of its internal states into substates $[s_1, s_2, \ldots, s_n]$, and a mapping f from those substates onto corresponding substates S_j of the CSA, along with similar mappings for inputs and outputs, such that: for every formal state transition $([I_1, \ldots, I_k], [S_1, \ldots, S_n]) \rightarrow ([S_1', \ldots, S_n'], [O_1, \ldots, O_l])$ of the CSA, if the system is in internal state $[s_1, \ldots, s_n]$ and receiving input $[i_1, \ldots, i_n]$ such that the physical states and inputs map to the formal states and inputs, this causes it to enter an internal state and produce an output that map appropriately to the required formal state and output. (Chalmers, 1996, 325)

This completes Chalmers' account of computation. He claims that this produces the non-trivial mind-independent notion of computation needed to secure realism about the CTM.

Problems

There are three problems with Chalmers' approach.

First, it is unclear whether CSA architectures really exhaust all the computational architectures relevant to the CTM. Chalmers argues that CSA architectures are more relevant to the CTM than FSA. This is almost certainly true. However, it does not show that CSA architectures are the only, or the most, relevant architectures to CTM explanations. The possibility is left open that there are other computational architectures that are equally, or more, relevant to the CTM than CSA and for which realist implementation conditions cannot be secured. This should be a worry for a realist about the CTM, because many CTM explanations appear not to involve CSA, or state-based automata, at all. For example, Marr's (1982) theory of vision involves series of discrete

computational filters that pass signals to each other in serial or parallel. This computational architecture is nothing like a CSA or FSA, in which a single monolithic unit undergoes state transitions.

Chalmers responds to this worry by claiming that the CSA architecture can reproduce any other computational architecture, including all those relevant to the CTM. Chalmers' claim here is not that CSA are I/O equivalent to other architectures—that much is uncontroversial.[20] Rather, his claim is that all other computational architectures *can be adequately expressed in the CSA formalism*. I am wary of this claim, for reasons that should be familiar from Chapter 1. I doubt that many of the architectures relevant to the CTM can be expressed in the CSA formalism without doing serious violence to their characteristics. For example, consider the filter-based architecture described above, or connectionist architectures. Even Chalmers' own example of an equivalent system, the Turing machine, is arguable. Chalmers argues that a Turing machine with finite tape can be identified with a CSA whose state vector components record the squares on the tape. I am not so sure. For Turing machines, there is a qualitative distinction between the state of the tape and the state of the head. In the CSA transformation, these two kinds of state are collapsed into one: they are both components of the state vector. It is not obvious whether this change really does violence to the architecture of Turing machines. At the same time, it is not obvious that this claim is false, and that the two systems really do share the same computational methods.

The second problem with Chalmers' solution concerns the mapping relation that Chalmers posits between physical states and abstract machine states. What is this mapping relation, and what are its truthmakers? Chalmers says nothing about this, but the problem should be pressed. Whether the mapping relation obtains or not is a key part of whether a physical system performs a computation. If, as Chalmers claims, performing a computation is a mind-independent matter, then this relation should have mind-independent truthmakers. But it is not clear what its truthmakers are, or whether they could be mind-independent. In order for Chalmers' approach to be genuinely realist, mind-independent truthmakers for the mapping relation must be secured.

One might claim that the mapping relation is an internal relation and hence not in need of independent truthmakers. There are two problems with this approach. First, it is not obvious that the mapping relation is an internal relation. Merely stipulating that it is does not settle the matter—stipulations by themselves cannot make it so. Second, an internal relation requires the existence of its relata. Therefore, in order for a mapping relation to obtain, *both* the physical state and the abstract mathematical object that is the CSA need

[20] Provided one restricts attention to architectures with finite storage.

to exist. This response therefore commits one to the existence of mathematical objects, and such commitments are far from uncontroversial.

The third problem is that it is not clear that CSA machines ultimately escape Putnam's anti-realist argument. Assume, without loss of generality, that the substates of a CSA take numerical values. Now define an FSA with states $S = 2^{S_1}3^{S_2}\ldots p_n{}^{S_n}$ for every possible substate S_i, where p_n is the nth prime; and transitions $S \rightarrow S'$ iff $[S_1, \ldots, S_n] \rightarrow [S_1', \ldots, S_n']$. The defined FSA is just like a CSA: it is a state-based automaton with exactly the same states and transitions. The only difference is that FSA states are characterised by scalars and CSA states are characterised by vectors. If this difference does not matter to the computational identity of the system, then every CSA is an FSA. By Putnam's result, any open physical system implements any FSA. Therefore, any open physical system implements any CSA.

Chalmers is aware of this problem. He acknowledges that an extra constraint needs to be added in order to avoid collapsing the CSA case to the FSA case. Chalmers suggests that this constraint is that 'each element of the vector corresponds to an independent element of the physical system' (Chalmers, 1996, 325). Unfortunately, the intended interpretation of this requirement is not obvious. It is not obvious how 'independent' should be understood so as to exclude anti-realism about computation. For this reason, Chalmers refines his suggestion: he claims that each component of the state vector of a CSA should correspond to a distinct *physical region* of the implementing system. Unlike the initial proposal, this suggestion is clear, but unfortunately it is neither a necessary nor sufficient condition for implementing a CSA.

Chalmers' condition is not necessary because it is conceivable that a system can implement a CSA even if its physical substates occupy the same spatial regions. There are a number of ways in which this could happen. First, a system could use properties to encode different substates, and different properties can be instantiated at the same spatial location. Second, as Chalmers admits, the substates of a CSA could change their implemented region over time. It is conceivable that two substates could gradually swap over so as to entirely occupy each other's regions. A real-world example of this would be the use of pointers in PCs: pointers allow the physical memory location of data to be changed without affecting the computation.

Chalmers' requirement is not sufficient because, even with his requirement, Putnam's result still applies. This can be shown as follows. Assume that the proper parts of an open system are themselves open systems. Pick an open physical system S and divide it into regions $r_1, \ldots, r_n$. By Putnam's result, each of these regions implement any FSA. Each region r_i therefore can be associated with the state S_i of an implemented FSA at any moment in time. Since $r_1, \ldots, r_n$

form part of the same system, they are in contact with each other—they share a common border along their edge. Define these bordering edges as the input and output to each FSA (if necessary, $r_1, \ldots, r_n$ can be chosen so that each r_i has an edge with every other r_j). Since the regions implement any FSA, they implement any FSA with input and output so defined. Therefore, they implement FSAs such that if the FSAs are in states $S_1, \ldots, S_n$ at one moment, they will be in states $S_1', \ldots, S_n'$ at the next moment, with the transition potentially governed by all the previous S_i via the inter-region inputs and outputs. Therefore, the system as a whole implements the state transitions $[S_1, \ldots, S_n] \rightarrow [S_1', \ldots, S_n']$. Hence, the system implements a CSA with those transitions. We have made no assumptions about the detailed nature of this CSA: it could be any CSA. Therefore, we can conclude that any open system implements any CSA. Modifying the system so that it accepts overall input and output, and its transitions have modal force is not hard.[21] Therefore, Chalmers' condition does not place the desired constraint on the notion of computation. For both CSA and non-CSA architectures, Putnam's anti-realist result is preserved.

2.3.5 Copeland's realist approach

Copeland (1996) advocates an alternative realist approach. Copeland's claim is that performing a computation is a matter of executing an algorithm. A system executes an algorithm just in case there exists a modelling relationship between that system and a formal specification of the algorithm and its supporting architecture. Therefore, a system performs a computation just in case it satisfies, in a model-theoretic sense, a formal specification. Copeland argues that if the modelling relationship is restricted to being *of a certain kind*, then a non-trivial form of realism about computation can be achieved.

Suppose that an entity e performs a computation. Let f denote the function that e computes, and let α denote the algorithm that e uses to compute f. According to Copeland, an algorithm α is a finite list of instructions such that anyone or anything that follows the instructions in the specified order is certain, if given the arguments of f as input, to yield the values of f as output. Copeland argues that algorithms are architecture-dependent: each instruction of α calls for the performance of an operation, and that operation is specific to the architecture on which α runs. For example, if α contains the instruction 'increment register', then the architecture on which α runs must at least support registers and their incrementing. An algorithm cannot run on a machine if the instructions that the algorithm employs are not supported on that machine.

[21] Note that it is not necessary to have a separate dial and clock for each region. All regions could share the same dial and clock.

Suppose that one has a formal specification both of the list of instructions that comprise α, and of the architecture on which α runs. Call this formal specification, SPEC. Copeland asks us to assume that SPEC takes the form of a set of axioms. Copeland gives an example of a SPEC. Suppose that a machine M consists of three eight-bit registers: an instruction register I, a data buffer D, and an accumulator A. Machine M performs various operations such as addition, multiplication, and transferring the contents of one register to another. Copeland gives a formal specification of the architecture of the machine M:

$Ax1$: If $\bar{I}$ = 00000001 ACTION-IS ($\bar{A} \Rightarrow \bar{D}$)
$Ax2$: If $\bar{I}$ = 00000010 ACTION-IS ($\bar{A} \Rightarrow \bar{A} + \bar{D}$)
$Ax3$: If $\bar{I}$ = 00000011 ACTION-IS ($\bar{A} \Rightarrow \bar{A} \times \bar{D}$)
$Ax4$: If $\bar{I}$ = 00000100 ACTION-IS ($\bar{A} \Rightarrow \bar{A} + \bar{\bar{D}}$)

In this specification, 'ACTION-IS' is intended to be a conditional with modal force, $\bar{X}$ is intended to be 'the contents of register X', and '$\Rightarrow$' is intended to be 'becomes'.

Axiom $Ax1$ states that the instruction 00000001 is the instruction to wipe the accumulator and transfer the contents of D to the accumulator. $Ax2$ states that the instruction 00000010 is the instruction to add the contents of D to the contents of the accumulator and store the result in the accumulator. $Ax3$ states that the instruction 00000011 is the instruction to multiply the contents of D by the contents of the accumulator and store the result in the accumulator. $Ax4$ states that the instruction 00000100 is the instruction to add the contents of the register whose address is stored in D to the contents of the accumulator and store the result in the accumulator. Setting aside the details of input, output, data control between CPU and memory, and program storage and control, these axioms specify an architecture for machine M. A SPEC consists of these axioms plus an algorithm. An algorithm in this case would be a sequence of instructions, such as '00000010, 00000001, 00000010'.

How does a formal specification, a SPEC, relate to a real-world entity, e? In order to answer this question, Copeland introduces the notion of a labelling scheme. A labelling scheme assigns representational content to spatial and temporal parts of an entity. For example, a labelling scheme might assign 0's and 1's to different spatiotemporal parts of an entity. For example, if the voltage across a certain logic gate is 5 V and the voltage across another logic gate is 0 V, then the pair might be labelled (High, Low) or $(1, 0)$.

We can now say what it means for a entity, e to perform a computation. When the formal axioms of SPEC *are true of* an entity e under a labelling scheme L, let us say that the ordered pair $\langle e, L \rangle$ is a *model* of SPEC. According to Copeland,

an entity e performs a computation just in case there exists a labelling scheme L and a SPEC such that $\langle e, L\rangle$ is a model of SPEC. In other words, an entity e performs a computation just in case a set of axioms SPEC are true of e under some labelling scheme.

As Copeland notes, this simple theory of computation is vulnerable to the kind of *reductio ad absurdum* arguments given by Searle and Putnam.[22] If we are to avoid the conclusion that the notion of computation is trivial, then some additional constraints are required. Copeland argues that these additional constraints can be introduced by requiring that the modelling relation be *of a certain kind*.

Honest models

There seems to be an intuitive distinction between standard and nonstandard interpretations of a set of axioms. According to Copeland, a nonstandard interpretation is a interpretation that does not respect the intended meaning of the axioms. A standard interpretation is an interpretation that does respect the intended meaning.[23] A nonstandard interpretation of a set of axioms concerning European geography might assign the number 1 as the referent of the symbol 'London', the number 16 as the referent of the symbol 'Moscow', and sentences of the form 'a is north of b', the truth conditions 'the referent of "b" < the referent of "a"'. On this interpretation, the sentence 'Moscow is north of London' is true, but it is no longer about Moscow and London. Similarly, the sentence 'Abraham Lincoln is American' is true under the interpretation that 'Abraham Lincoln' refers to the Queen of England, and 'American' to the property of being English, but it is no longer about Lincoln or being American. More generally, the Löwenheim–Skolem theorem entails that if a first-order theory is true under any interpretation at all, then it is true under an interpretation whose domain consists of at most the natural numbers.[24]

Copeland claims what is wrong with Searle's and Putnam's arguments is that they exploit nonstandard interpretations to show that any physical system can perform any computation. (A labelling system is here to be understood as an interpretation). When Searle claims that a brick wall performs a computation, what he means is that the axioms that describe that computation are true of the brick wall. But, argues Copeland, those axioms are true of the brick wall only under a nonstandard interpretation. Brick walls do not perform com-

[22] For the argument, see Copeland (1996), 343–346.

[23] In mathematical logic, a nonstandard interpretation means an interpretation which is not isomorphic to the intended interpretation. Copeland's use of the term is broader: a nonstandard interpretation may or may not be isomorphic to the standard interpretation. See Copeland (1996), 346.

[24] Provided the theory is countable and contains the identity predicate.

putations under standard interpretations. Call a model based on a standard interpretation an *honest* model. Copeland claims that a non-trivial notion of computation can be achieved by requiring that a system perform a computation only under an honest model. Formally, an entity e performs a computation just in case there exists a labelling scheme L and a SPEC such that $\langle e, L\rangle$ is an honest model of SPEC.

What makes a model honest? The answer cannot be our *intentions* in interpreting the axioms—that would concede the game to the anti-realist. There must be some mind-independent conditions that determine whether a model is honest or not. Copeland puts forward three conditions that he claims are necessary for honesty.

Copeland's first condition is that the model must interpret the SPEC axioms as being about action. Copeland claims that in Searle's example the brick wall is a passive 'scoreboard', whereas it should be an active participant in the computation. In order for an entity to perform a computation, it is not enough that entity display a certain behaviour, it must also *act* in bringing about its behaviour. Copeland explains the notion of action he has in mind in terms of Belnap's *stit* and Segerberg's δ operator.[25] However, the *stit* and δ operator are intended to formalise actions concerning *agents*. They formalise 'A sees to it that p comes about', where A is an agent and p is an event.[26] The implied notion of action is intentional action. This seems too strong a notion for computation. First, many entities perform computations even though they lack intentional agency—for example, AND and OR logic gates. Second, we may wish to explain agency, or at least aspects of agency that concern decision theory, in terms of the performance of computations. Such accounts would be circular if the performance of a computation itself required agency.

Copeland's second condition is that the model must not interpret the SPEC axioms in a time-relative way. The interpretations of Putnam and Searle fail to meet this condition because they assign computational states to the wall over a time interval, but say nothing about the state of the wall before or after that interval. Copeland contrasts this with honest cases of computation in which an interpretation applies to an entity throughout its lifetime. Care must be taken in interpreting this condition, for two reasons. First, Copeland's point cannot be that the anti-realist is unable to interpret the wall before some particular t_1 or after some particular t_n, for the anti-realist can interpret the wall as performing a computation over *any* finite interval of time she likes. Second, his point cannot be that the anti-realist is restricted to *finite* intervals of time, since, as Copeland

[25] See Belnap (1996) and Segerberg (1996).

[26] There are a number of differences between *stit* and δ but those differences are not relevant here—both operators concern formalising actions peculiar to agents.

admits, honest cases of computation have finite lifetimes too.

Copeland explains his time-relativity condition in the following way. The time-relativity problem with nonstandard interpretations is, he says, that they are *ex post facto*: they are constructed after the event. In contrast, in honest cases the interpretation can be specified before the event, and one can thereby *predict* aspects of the entity's physical behaviour. However, there are two problems with this suggestion. First, there is no reason why the anti-realist is restricted to constructing her interpretation after the event. In the case of Putnam, knowledge of classical physics and the boundary conditions can be used to construct the interpretation before the event. Second, and more importantly, it is a metaphysical, not an epistemic point that is at issue here. In terms of the truthmakers of computation, it *does not matter* if an interpretation can only be constructed by us after the event: all that matters is that such an interpretation *exists*. (Plausibly, if an interpretation exists after the event, then it also existed before the event.) Questions about prediction are largely irrelevant in this context. It is the existence of interpretations, not our ability to find them (which may be limited *ex post facto*), that is at issue when considering the facts of computation.[27]

Copeland's third condition is that the model must interpret the SPEC axioms as having counterfactual force. However, we have already seen that this condition is easy to meet. Any system with a dial and a clock satisfies the relevant counterfactual conditionals . Although counterfactual dependency might be a necessary condition for computation, it cannot be a sufficient condition. Since the two other conditions that Copeland suggests—intentional action and lack of time-relativity—are not even necessary, let alone (taken together with the counterfactual condition) sufficient, we are left with the anti-realist's *reductio*. In terms of the mind-independent facts, nearly every physical system performs any computation. Therefore, Copeland's account does not achieve the desired result: a notion of computation that is both mind-independent and non-trivial.

Before closing, two further aspects of Copeland's account should be noted. The first is that Copeland says nothing about the truthmakers of the labelling scheme. This matters because if the truthmakers of the labelling scheme are mind-dependent, then realism about computation is false, and it would be unclear why realism about the other conditions is even worth fighting for. Second, even if Copeland's account is correct, it does not provide an account of algorithm or computation individuation. Apart from merely syntactic differ-

[27] It is worth noting that Copeland acknowledges that there is a difference between the existence of an interpretation, and our ability to construct it: 'A sentence of the form "There exists a function f such that ..." may be true irrespective of whether it is know to be true. I use the phrase "there exists a labelling scheme and a formal specification such that" in the same way.' (Copeland, 1996, 338).

ences between different axioms, the account provides no clue as to when two computations are the same or different.[28]

2.3.6 Mellor's realist approach

Mellor (1991a) presents a theory of computation that he developed independently of the anti-realist considerations discussed above. However, his theory can be used as a response to such arguments. Of all the theories considered so far, Mellor's position is closest to the view presented in this thesis. In short, I think that Mellor is essentially correct. The differences between his approach and my own concern the details, emphasis, and implications of the approach, not the fundamentals. Mellor outlines his theory in rapid and characteristically terse fashion. The position developed in the next two chapters, although happened upon independently, can be seen as an elaboration and defence of his position.

Mellor defines a computation is a causal process that maps certain inputs to certain outputs. The notion of a causal process is meant to be understood as having counterfactual force. In other words, to say that something is a 'causal process' implies not just what does happen, but also what would have happened. Not all causal processes are computations. In order to qualify as a computation, a causal process must be *semantic* and *syntactic*. A causal process is semantic just in case its inputs and outputs have representational content. A causal process is syntactic just in case the representational content of each input/output depends, in some well-defined way, on the representational content of that input/output's spatial or temporal parts. Not all causal processes satisfy these two conditions. Therefore, a non-trivial notion of computation is achieved.

Mellor does not explicitly consider realism about computation, but it is clear how such a position could be developed. His account makes computation dependent on representation. If one can be a realist about representation, then one can be a realist about computation. What are the prospects for realism about representation? In the case of electronic computers, those prospects are poor. Electronic computers generally represent only what we *choose* them to represent. However, for other causal processes, and in particular those in the brain, the prospects for realism about representation seem better. A number of philosophers have defended realism about representation in this context, including Dretske (1981), Fodor (1990b), and Millikan (1986). If, for example, our *beliefs* represent in a way that does not depend on attitudes towards them, then a causal process in which those beliefs take part could be a mind-independent

[28] See Section 2.3.1 for criticism of syntactic approaches to computation individuation.

computation. Mellor does not go so far as to claim realism about representation. He claims that beliefs represent, but he does not say what their truthmakers are.[29] However, if realism about representation can be secured, then Mellor's account secures realism about computation as well.

In light the argument of Chapter 1, it is worth noting that Mellor claims that the conditions for a causal process to be a computation—namely, that it be both semantic and syntactic—are interdependent. A causal process is syntactic just in case the representational content of each input/output depends, in some well-defined way, on the representational content of that input/output's spatial or temporal parts. Therefore, in order for a causal process to be syntactic, it must already have semantic content. Syntax, as defined above, is not independent of semantics; syntax *presupposes* semantics. Note also that if the first condition is satisfied—if the process is semantic—then the second condition is automatically satisfied. This is because even if a token has no interesting syntactic structure, it still qualifies as having a syntax, albeit a simple one: its representational content depends on a function, namely the identity function, of the representational content of its spatial and temporal non-proper part. Even if the *proper* parts of a token do not have distinctive representational content, that token can still qualify as having a syntax merely by having semantic content. Hence, if the semantic condition is satisfied, then the syntactic condition is automatically satisfied too.

Contrast with the PR-model

There are five respects in which the position developed below differs from that of Mellor.

First, Mellor restricts the representational content of tokens that participate in computations to *propositions*. A token may represent the proposition *the earth is round*, and a part of that token may represent *the earth*, but a whole input or output token of a computation cannot represent *the earth* or *roundness*. The inputs and outputs of a computation must represent propositions. The motivation for this requirement is the intuition, with which Mellor identifies, that to perform a computation is to process information. Information is either true or false. Therefore, the bearers of information must be truth-evaluable; they must be capable of truth or falsity. Whatever theory of propositions one favours, propositions are the paradigmatic entities that are truth-evaluable. Therefore, Mellor elects to call the bearers of information 'propositions'. Since computation is information processing, computation is the processing of propositions.[30]

[29] 'How beliefs represent states of affairs is another question, which I fortunately need not answer.' (Mellor, 1991a, 71).

[30] See Mellor (1991a), 62–63.

I am not so sure. I am doubtful that information processing should be taken as definitive of computation. First, the information processing model for explaining the mind pre-dates the computational model, and the computational model should not be taken as beholden to it.[31]Second, the sense in which computations are said to 'process information' is fairly loose and ill-understood. As Mellor notes, the relevant notion of information is not that of mathematical information theory. In my view, claims about information processing can be excised from cognitive science with little or no loss; typically, such claims carry rhetorical value rather than analytical weight. If they *do* carry analytical weight, then the intended notion of information is meant to be understood in terms of computation, not vice versa. Computation is regarded as the respectable notion in cognitive science, not information. Few cognitive scientists would be willing to rest the foundations of their discipline on their pre-theoretical intuitions about what is and what is not information. Third, there seem to be paradigmatic cases of computation in which inputs and outputs do *not* represent propositions. Mellor himself gives an example. Imagine a machine that computes the $f(x) = x^2$ function: it takes tokens that represent numbers (e.g. '5') as input, and yields tokens that represent numbers (e.g. '25') as output. Numbers are not propositions (for one thing, they do not have a truth value). Therefore, we appear to have a counterexample to Mellor's claim. In response, Mellor argues although the machine *by itself* is not a computer, a joint system comprising of the machine and someone using it to process propositions is a computer. For example, if I use the machine to verify that if the side of a square is 5 cm then the area of the square is 25 cm, then there is an overall causal process, of which the machine is a part, in which the overall input ('the side of a square is 5 cm') and the overall output ('the area of the square is 25 cm') represent propositions. Mellor says that this overall process qualifies as a computation, but the process inside *just the machine* does not. This seems wrong. The squaring machine is a paradigmatic case of computation. Based on our pre-theoretical intuitions about computation, there could not be a clearer case. Mellor is too strict. He ignores paradigmatic cases of computation in order to preserve an idea, to which many cognitive scientists would be reluctant to commit, of computation as information processing.

Second, the only tokens that Mellor considers as genuinely representational are beliefs (and perhaps other propositional attitudes). All other tokens get their content in a derivative fashion from beliefs.[32] Mellor does not say much

[31]See Lachman et al. (1979), 121 for a discussion of how the information processing model was superseded by the computational model.

[32]For example, discussing a mental representation '$c > b$' that takes part in a computation but is not a belief: 'Like my computer token '$C >> B$', ['$c > b$'] represents $c > b$ only *via* the token beliefs that are its causes and effects.' (Mellor, 1991a, 75).

about the motivation for this assumption, but it seems too strong. First, there might be tokens that represent, and represent in a realist way according to Dretske (1981), Fodor (1990b), and Millikan (1986), but which are not beliefs. For example, the activity in a small cluster of neurons might *represent* a line in the visual field without being a full-blown belief—the representation might not be conscious, and it may not connect with desires in the right ways to cause action. Cognitive science is full of such representations. *Prima facie*, there seems no reason why these states cannot be genuinely representational. It is both reasonable, and in the spirit of cognitive science, to posit sub-personal representational states without belief.

Third, the only processes that Mellor considers are causal processes. This is unduly restrictive. In Section 3.2.7, I argue that *any* class of relations that supports the relevant counterfactuals suffice to implement a computation. Therefore, computations need not be causal; other dependency relations can be used. Of course, the force of this point depends on how broadly one understands the notion of 'causal process'. Unfortunately, Mellor does not say anything about how this notion should be understood. This is doubly unfortunate, because it also obscures the truthmakers of his theory. It is likely that Mellor intends talk of causal processes to be made true by more basic facts, such as combinations of events, individuals, properties, and relations. However, he gives no indication of how this is to be done. In the account in the next two chapters, process talk is explicitly reduced to talk of standard metaphysical entities.

Fourth, like Copeland, Mellor does not provide an account of the individuation conditions of computations. This is understandable, given that Mellor is interested in different problems concerning computation. However, as we have seen, the individuation conditions of computations are important. The conditions under which two computations are the same or different determine whether a given cognitive science theory is true or false.

A final point of difference between my position and Mellor's is that he draws a conclusion about the extent to which the mind is computational with which I disagree. Mellor claims that it is only possible for two kinds of mental process—inference and perception—to be computations. All other mental processes must be non-computational. Mellor's argument for this conclusion is as follows.

First, mental processes whose inputs and outputs *do not represent* can be ruled out immediately. Mellor claims that mental processes involving pains and other sensations fall into this category. These mental processes cannot be computations. Second, mental processes that represent, but do not represent propositions, can be ruled out for similar reasons. Third, many of the remaining aspects of the mind that *do* represent propositions cannot participate in computation either. This is because, although they represent propositions, their

representational content does not convey information. For example, consider desires. Desires have propositional content, but, according to Mellor, desires do not *embody information*—they do not represent any proposition as true. A desire, such as the desire to close a door, represents the proposition *that the door is closed*, but it does not represent that proposition as true. The same is true of other propositional attitudes such as hoping, fearing, and wishing. Belief is the exception because, as Moore's paradox indicates, to have a belief is to take the represented proposition as true. Therefore, beliefs embody information. Beliefs—and perceptions, for which a similar argument can be made—are legitimate candidates for information processing, and hence for computation. Other propositional attitudes are not.[33]

Unsurprisingly, I do not accept this conclusion. As already discussed, information processing, at least in the context of the CTM, should not be taken as analytic of computation. Our intuitions about information processing should be guided by our intuitions about computation, not vice versa. Furthermore, computation need not, as Mellor also claims, aim at preserving truth. Computation may serve other functions as we saw in the case of the squaring machine, such as systematically transforming one class of representations into another class of representations. On the view developed in the next two chapters, there is no reason why computations cannot involve desires as well as other non-propositional kinds of representation.

2.4 Conclusion

This concludes the survey of existing problems for, and solutions to, realism about computation. We have seen a number of anti-realist arguments: (1) *reductiones ad absurdum* of the notion of mind-independent computation; (2) the claim that an interpreter is necessary to interpret the input, output, and intermediate states of a computation; (3) the claim that an interpreter is necessary to interpret a computation as having an infinite domain; (4) the claim that an interpreter is necessary to make sense of correct functioning. In the next two chapters, a view is developed that will enable us to respond to these objections. This view will show that realism about computation is possible. It will also enable us to provide an account of the individuation conditions of computations in terms of the individuation conditions of standard metaphysical entities. It is to this positive position that we now turn.

[33] See Mellor (1991a), 77–81.

Chapter 3

Preliminaries

This chapter and the subsequent chapter provide a semantics for our computation talk. The proposed semantic model is called the 'process and representation model' (PR-model). Once this semantic model is in place, we shall be in a position to identify the truthmakers of that semantic content. In particular, we shall be in a position to identify whether those truthmakers can be mind-independent. The PR-model provides a semantics for: (1) claims that a system performs a computation; and (2) claims that a system performs one computation rather than another. This chapter begins with an overview of the CTM and two key intuitions concerning computation. The aim of the PR-model is to formalise this account and these two intuitions. The rest of the chapter introduces the basic concepts of the PR-model and the kinds of metaphysical commitments they entail. This prepares the way for the task of the next chapter: the description of the PR-model proper.

3.1 The discourse to be formalised

3.1.1 The computational theory of mind

The computational theory of mind attempts to answer the question: 'Given a cognitive process P that systematically transforms inputs, Φ, into outputs, Ψ, *how does P work?*' In particular, the task is to give some explanation of how process P works that fits with general assumptions about the finite nature of the human mind and its relationship to the physical world. Examples of cognitive processes that psychologists have been interested in explaining include: syntax parsing, simple shape recognition, and deductive inference.

If the I/O behavior of a process is particularly simple, then it is easy to hypothesise how that process might work. For example, suppose that for any

input, ϕ, presented to a process R, that process yields ϕ again as output. Process R does no more than reproduce its input as output. The I/O behaviour of process R is so simple that it is easy to come up a mechanism for how it could work. For example, R could be a noiseless wire connecting input to output: if an input ϕ is presented at one end of the wire, then an output ϕ is yielded at the other end. Of course, R might *in fact* work in a more complex way: a Rube Goldberg machine might connect its input to its output. However, the I/O behaviour of R is simple enough that it is not hard to come up with a mechanism for how R *might* work. As we shall see, this condition is not met for many processes with more complex I/O behaviour. Here is another example: a process S always yields a fixed output ψ_c for any input ϕ. The I/O behaviour of S, like that of R, is simple enough that it is easy to come up with a mechanism for how S could work. One such mechanism would be a noisy wire: a wire that ignores its input. For every input ϕ presented at one end of the wire, the same output ψ_c is yielded at the other end.

The I/O behaviour of processes R and S is such that we can easily hypothesise how they might work. If all psychological processes were like R and S, then there would be little motivation for computational theories of mind—cognitive processes could be explained in more direct ways. However, psychologists are often interested in processes that have much more complex I/O behaviour than either R or S. For these processes, theories about how they might work are not so easy to come by. It is here that the notion of computation can help.

Consider the I/O characteristics of our syntax parsing processes for natural language. A syntax parsing process takes strings of phonemes as input and yields a grammatically structured representation as output. Any one of a vast number of strings of phonemes can be paired with any one of a vast number of grammatical representations. The I/O pattern of the process is incredibly complex. One struggles to find a simple mechanism of the kind described above to explain how it could work.[1] Generally speaking, if a cognitive process has complex I/O behaviour—and many cognitive processes that psychologists are interested in do—then it is difficult to explain how such processes could work. It is not just difficult to explain how such processes *in fact* work, it is difficult to explain how they *might* work.

Complicated I/O behaviour is one source of problems for explaining how cognitive processes work. Another source of problems is that some cognitive processes appear to be sensitive to the semantic content of their input. *Prima facie*, it is difficult to know how such behaviour is possible. For example,

[1] Historically, associationism has been a popular mechanism for explaining how complex cognitive processes work. Associationist mechanisms have been advocated by theorists from Hume to Skinner. See Chomsky (1957, 1959) for an argument that associationist mechanisms cannot account for our syntax parsing I/O capabilities.

consider the cognitive process of deductive inference. This cognitive process tends to preserve the semantic property of truth: if one already believes truths, then the cognitive process of deductive inference tends to take one to more truths. How is this possible? Again, it is unclear how to explain how such a process might work.[2]

Computational theories of mind can potentially solve both problems. Computation provides a mechanism for how complex, semantically sensitive, processes could work. How does the notion of computation do this? In order to answer this question we need to discuss what it means for a process to be a computation. If a process is a computation then, among other things, it must satisfy two conditions: (1) it must be made up of a finite number of simple steps; and (2) it must be sensitive to the formal structure of its input. Let us discuss each condition in turn.

Computations are made up from simple steps

A computational process P is made up from a finite number of simple steps just in case it is made up from a finite number of simple subprocesses $p_1, \ldots, p_n$. A process is made up from a finite number of subprocesses just in case it is constituted from start to finish by p_i, and no part of it is not constituted by p_i. The p_i's can be arranged in a variety of ways. Two p_i can be connected together by the output of one p_i being the input of another. However, the process need not be a linear sequence of p_i's: the p_i's can form a complex network. A process is made up from *simple* subprocesses if, in addition to the above condition, it is not mysterious how the individual p_i work. For this condition to be satisfied, the I/O behaviour of the individual p_i must be so simple that it is possible to directly explain how they might operate. Like processes R and S, the I/O behaviour of the simple p_i must be simple enough that is clear how those patterns could be achieved.

Computations are sensitive to formal structure

Each input of a process's p_i must have a formal or syntactic structure. This condition can be satisfied in a number of mundane ways.[3] For example, an input can have formal structure by consisting of eight ink-marks, each of which is '0' or '1', separated by a time delay. Such an input has a temporal formal

[2] Associationist mechanisms again are not adequate. Associations between beliefs need not follow logical entailment. Human subjects generally have just as strong associations between truths and falsehoods as they do between truths and truths. It is hard to see how an associationist account can explain our ability to systematically preserve truth.

[3] It is worth emphasising that the notion of computation under consideration here is that of implementation in real-world systems. The sense in which computation in mathematics is formal or syntactic will not be considered.

structure. (This eight-fold structure is called a byte). Another possible structure is spatial. The eight signals could be separated by spatial displacement, like the ink-marks in '01101110'. Or, instead of being made up from ink-marks, the input could be made up from electrical signals. For example, the input could be a time-delayed sequence of electrical pulses, each of which is 0 V or 5 V. Alternatively, the components of the electrical signal could occur at different frequencies but at the same time. Another possibility is that the input is made out of distinct interlocking parts, and thereby have a part–whole structure. One such input is a Lego model, which is built out a finite number of distinct interlocking blocks. Another is a DNA molecule, which is built out of a sequence of distinct interlocking nucleotides. In each case, formal structure is just an aspect of an input's physical structure.[4] A computational process can be sensitive to the formal structure of its inputs by being sensitive to the physical features that constitute that structure. There are many examples of processes that are sensitive to the temporal structure, spatial structure, and part–whole relations of their inputs.

These two conditions that a computational process must satisfy—being made up from a finite number of simple steps and being sensitive to formal structure—enable us to explain how complex semantically sensitive I/O behaviour is possible. First, let us consider complex I/O behaviour.

Explaining complex behaviour

Complex I/O behaviour can be achieved if simple subprocesses are chained together in right way. A few simple p_i are sufficient to produce extremely complex I/O behaviour. Indeed, *any* I/O behaviour that can be produced by a computer can be reproduced by a computational process consisting of just AND gates and OR gates.[5] AND and OR gates are good candidates for simple p_i: their I/O behaviour is simple enough that, like processes R and S, it is easy to explain how they might work. In the case of electronic computers, AND and OR gates are usually explained in terms of simple electrical principles. These gates are usually built out of transistors, but they could be built in other ways.[6]

Computation provides us with a way of explaining how cognitive processes with complex I/O behaviour could work. By breaking up a process into simple parts, each of which it is obvious how it could work, we provide an explanation

[4] If the computation is implemented in a non-physical medium, then the formal structure of the input is one of the structures in the medium in which that computation is implemented.

[5] Strictly speaking, any computable behaviour achievable with finite memory.

[6] An AND gate consists of two switches (wires that can be open or closed) in serial: the output of the gate is *on* only if both switches are closed. An OR gate consists of two switches in parallel: the output of the gate is *on* if either one of the switches is closed. AND and OR gates are usually implemented using transistors because transistors are fast switches.

of how a cognitive process with complex I/O behaviour could be achieved. Such explanations may turn out to be wrong—we may make a mistake and attribute a mechanism that a process does not in fact possess. However, at least we have something that we did not have before: an explanation of how such a process *might* work.

Explaining semantically sensitive behaviour

Computation also provides a mechanism for how a process could have semantically sensitive I/O behaviour. Semantic sensitivity can be achieved if the formal structure of an input covaries with its semantic structure. Covaration between formal structure and semantic structure is common. For example, the shape of natural language words often covaries with their meaning within their language. We have already discussed how computational processes are sensitive to the formal structure of their input, namely, by being sensitive to the physical features that constitute that structure. If formal structure covaries with semantic structure, then, by being sensitive to formal structure, a computational process will also be sensitive to semantic structure. Computation therefore provides a mechanism for explaining how semantically sensitive I/O behaviour is possible.

For example, consider the cognitive process of deductive inference. This cognitive process tends to preserve truth. The problem is to explain how this is possible: how can a physical process be sensitive to a semantic property like truth? Applying the strategy above, a computational process can track truth-preservation by tracking a formal counterpart of truth-preservation: proof-theoretic entailment. Suppose that a computational process, if presented with an input ϕ that represents the proposition *snow is white and grass is green*, yields an output ψ that represents the proposition *grass is green*. The inference from ϕ to ψ is truth-preserving: if ϕ is true, then ψ cannot help but be true. As discussed above, there are a variety of ways in which the structure of ϕ could reflect its truth-functional semantics. The input ϕ might be a composite of three components: one, ϕ_1, representing the first conjunct, one, $\wedge$, representing the conjunction operator, and one, ϕ_2, representing the second conjunct. The three members of the composite could be ink-marks on the page, e.g. ϕ = '$\phi_1 \wedge \phi_2$'. Alternatively, the three members of the composite could be three electrical signals. Whatever the details of the formal structure of ϕ, if that formal structure mirrors ϕ's truth-functional structure, then the process can track truth-preserving inferences.

The same strategy can be used to explain apparent I/O sensitivity to any other semantic property. All that is required is that the formal structure of

the input covary with *all* the semantic features relevant to the processing. For example, some word processors have an 'Auto Summarise' feature that takes sections of text as input and yields a short summary of the text as output. The I/O pattern of this process aims to be semantically sensitive. The computer program does not achieve this I/O by somehow 'knowing' the meaning of the text. Instead, it relies on covaration between formal features of the text, such as the frequency of words and phrases, with the meaning of the text. By responding to formal features, the program can have an I/O behaviour that approximates that it would have if it were responding directly to semantic features.

It is not obvious to what extent this strategy can be used to explain semantic sensitivity in human cognition. Putnam (1975a, 1988) and Burge (1986) argue that many important semantic features of human cognition do *not* covary with formal properties available to an individual's cognitive processes. If psychological laws employ these external semantic features, then the mechanism through which those laws work cannot be computational. Therefore, there might be limits to how far computation can be used to explain psychological processes. For computational responses, see Block (1986); Fodor (1980a, 1994); Stich (1983).

Explanatory value and truth

The role that computation plays in cognitive science is to explain how complex semantically-sensitive cognitive processes are possible. Without the notion of computation, it is not clear how these processes work, or how they are possible at all. Therefore, the point of computational theories of mind is *that they provide a way to explain how complex semantically-sensitive cognitive processes could work*.

An explanation can have many virtues. An explanation can be good because it unifies previously diverse phenomena, because it is useful in creating new technology, or because it provides a perspicacious way of arranging existing data. One of the principal virtues of a good explanation—some would say an *essential* quality of a good explanation—is that it be true.[7] The question that concerns this thesis is what is it for a computational theory of mind to be true. The answer will come in two parts: first, an account of the semantic content of a computational theory of mind—what do we mean when we say that the mind or brain performs a particular computation? Second, an account of the kind of facts that make that semantic content true or false—the truthmakers of that theory.

This chapter and the next concern the semantic part of the project. We

[7] For example, see Hempel (1965), 248–249.

already have an outline of what we mean when we say that a system performs a computation. This outline is that a system performs a computation just in case the process involved is made up from simple parts and those parts are sensitive to the formal structure of their inputs. However, the content of these claims is still far from clear. In the rest of this chapter, and the subsequent chapter, a more detailed account is given.

3.1.2 Two key intuitions

The account above needs to be supplemented by two intuitions. These two intuitions are essential to our notion of computation.

(i) Computation involves representation

The first intuition is that computation involves representation. For the moment, the version of this claim that I wish to defend is that the inputs and outputs of a computation must have representational content. The claim will be extended to apply to the inputs and outputs of the intermediate steps of a computation in Section 4.2.1.

A computation maps certain inputs to certain outputs. A computation, in the sense considered by this thesis, is a mapping between real-world *stuff*: it takes stuff (ink-marks, electrical signals, etc.) as input and yields other stuff, or other arrangements of stuff, as output. The claim made above is that computations are not just mappings between any kind of stuff, but mappings between *stuff that represents*.

This claim does not place any restriction on the type of representation relation involved. Nothing is said, for example, to require that the inputs and outputs of computational processes must *intrinsically* or *naturally* represent. In some cases this may be true—the inputs and outputs may naturally represent—in other cases it may not—the inputs and outputs only represent because we, as humans, interpret them as doing so. Nevertheless, whatever kinds of facts underlie representation, the inputs and outputs of a computation must represent.

Before defending *this* claim, it is worth noting that it is already widely, although not universally, held. Fodor uses the claim as part of his argument for the existence of a language of thought. Fodor's argument consists of three steps. The first step claims that the only plausible psychological accounts of many cognitive processes are computational. The second step claims that computation requires representation. The third step claims that the representations involved in cognition have a language-like nature.[8] Many philosophers have

[8]See Fodor (1975), 27–29, 34, 31–32 for the first, second, and third steps respectively.

disagreed with Fodor's conclusion, but criticism has usually focused on the first or third steps, not the second step. Philosophers with views as divergent as Dennett (1971), Churchland (1986), and Cummins (1989), agree that computation involves representation. Even Searle (1992) agrees that computation involves representation.

Here are three arguments for why computation has to involve representation:

Argument 1. *Paradigmatic cases of computation involve representation.*

Many paradigmatic cases of computation involve representation. For example, Turing's clerk, who performs computations by hand, performs a mapping between representations. The clerk maps representations (ink-marks on the page) to other representations (other ink-marks on the page). The clerk's ink-marks can be interpreted as representing either numerals or numbers. This ambiguity in representational content is not unusual. The following ink-marks, 1, can represent either the numeral '1' or the number 1, depending on context. The context can be partially specified by adding quotation marks. However, this convention is not always decisive. In many cases there is opportunity to interpret the ink-marks either way. As we will see later, it is not unusual for the same physical stuff to have multiple representational contents associated with it, and therefore for the same physical process to have multiple computational identities.

Another paradigmatic case, electronic computation, also involves representation. An electronic computer takes electrical signals as input and yields electrical signals as output. The input and output electrical signals of a computer are not just any electrical signals, but electrical signals that represent. Typically, the electrical signals of a computer represent 0's and 1's. Again, there is possibility of multiple representation. A given signal may represent *both* a long sequence of 0's and 1's, *and* the text of a new e-mail message. Or, a given signal may represent *both* a sequence of 0's and 1's, *and* a picture to display on the screen.

Argument 2. *Representation is involved in the notion of I/O equivalence.*

We saw in Chapter 1 that I/O equivalence is a necessary, but not sufficient, condition for performing the same computation. There is more to the notion of computational identity than I/O equivalence, but I/O equivalence is at least an essential component of that notion. I wish to claim that it is hard to make sense of I/O equivalence without assuming that computation involves representation.

Imagine two I/O equivalent systems that are made out of different materials. One system may be made out of silicon and take electrical signals as inputs

and outputs, the other system may be made out of tin-cans and string and take marbles as inputs and outputs. Suppose that the two systems are I/O equivalent. What could their I/O equivalence consist in? The respective inputs and outputs of the two systems may be so different so as to not have any physical or structural properties in common. The only plausible answer seems to be that their respective inputs and outputs *represent the same thing*.

Another example would be two computational systems that perform the same numerical calculation. Suppose that one system takes ink-marks shaped like Roman numerals (I, II, III, IV, ...) as input and yields ink-marks shaped like Roman numerals as output. Suppose that the other system takes ink-marks shaped like Arabic numerals (1, 2, 3, 4, ...) as input and yields ink-marks shaped like Arabic numerals as output. Suppose that we wish to say that the two systems are I/O equivalent. What could their I/O equivalence consist in? There need be no physical or structural similarity between their respective inputs and outputs. The only respect in which the two systems are I/O equivalent seems to be that their inputs and outputs represent the same thing.

Argument 3. *Representation is needed to make some basic distinctions.*

Any plausible notion of computation needs to be able to make certain basic distinctions. One such distinction is that between AND gates and OR gates—the building blocks of many electronic computers. AND and OR gates have the following characteristics. The output of an AND gate is *1* just in case both inputs are *1*, otherwise it is *0*. The output of an OR gate is *0* just in case both inputs are *0*, otherwise it is *1*.

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

Table 3.1: AND and OR gates

Consider an electrical system with following characteristics. The system gives an output of 5 V if both its inputs are 5 V, otherwise it gives an output of 0 V. Does this system implement an AND gate or an OR gate? At first glance, the system appears to implement an AND gate: it gives an output with 5 V just in case both its first *and* its second inputs are 5 V. But why should 5 V be associated with *1*, and 0 V with *0*, rather than the other way around? If 5 V is associated with *0*, and 0 V with *1*, then according to the tables above, the

system implements an OR gate. So which gate does the system implement? As the system has been described so far, there is nothing to decide between the two options. No physical or structural property decides whether 5 V should be paired with *1*, and 0 V with *0*, or 5 V with *0*, and 0 V with *1*. The situation is symmetrical with respect to both assignments.

in_1	in_2	*out*
0 V	0 V	0 V
0 V	5 V	0 V
5 V	0 V	0 V
5 V	5 V	5 V

Table 3.2: An implementation of an AND gate or an OR gate?

The notion of representation allows us to decide between these two options. We can say that *if* an electrical signal of 5 V represents *1*, and *if* an electrical signal of 0 V represents *0*, *then* the system implements an AND gate. Alternatively, *if* an electrical signal of 0 V represents *1*, and *if* an electrical signal of 5 V represents *0*, *then* the system implements an OR gate. It seems that the difference between an implementation of an AND gate and an OR gate is a difference in representational content.

The representational nature of real-world computation is sometimes obscured by the widely accepted claim that computation is syntactic. Computation is syntactic in at least two senses. First, as we saw in Section 3.1.1, computation is sensitive to the formal, syntactic, structure of its input. This requirement is, of course, compatible with the claim that such input has representational content. The second sense in which computation is syntactic is that the inputs and outputs of a computation often *represent* syntactic entities. We often take an input to a computational process to represent a numeral ('0' or '1') rather than a number (*0* or *1*). Thus, one often finds the inputs and outputs of an AND gate labelled with the numeral '0' or '1', and this numeral called its 'syntactic content'. Such content may be a syntactic, but it is representational nevertheless.

Another possible source of confusion about syntax arises from the conflation of the notion of real-world computation with the notion of computation in mathematics. Real-world computation and computation in mathematics are different. A Turing machine employs the mathematical notion of computation. A Turing machine is not a physical object, it is an abstract mathematical object; it does not 'perform' a computation in the same sense as a physical system. A Turing machine can be identified with the quintuple $M = (Q, \Sigma, \Gamma, \delta, q_0)$, where Q is a finite set of state symbols, Γ is a finite set of numerals that can be used

on the tape, B a special symbol that represents a blank, Σ is a subset of $\Gamma - \{B\}$ called the input numerals, δ is a partial function from $Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$ called the transition table, $q_0 \in Q$ is a special state called the start state, and the symbols L, R the direction of movement along the tape (conventionally labelled 'left' or 'right').[9] A Turing machine so defined is a mathematical object, a set of sets of symbols. It is not dissimilar in ontological status to a mathematical function, such as $f(x) = x^2$.

A Turing machine, at least in the first instance, operates on numerals instead of numbers. It takes numerals (symbols) as input and yields numerals (symbols) as output. A Turing machine therefore operates on syntactic entities.[10] Two points should be made about these syntactic entities. First, syntactic entities such as numerals are themselves abstract objects—they are not identical to ink-marks on the page, although ink-marks may represent them. Second, syntactic entities are commonly thought of as uninterpreted in this context, i.e. as lacking representational content. Hence, mathematical computation does not need to operate on entities that represent. This may lead one to think that the computations performed by real-world systems do not need to operate on entities that represent either. Unfortunately, this is not true. Although symbolic entities, such as numerals, provide a way of individuating Turing machines, these objects are not available in the real-world. In real-world computation, we are stuck with physical stuff, such as ink-marks and electrical impulses. As we have seen, the only way for such stuff to support a plausible notion of computational identity is to employ the notion of representation. The non-representational nature of mathematical computation does not transfer to real-world computation. The two notions of computation have different problems and metaphysical commitments.

(ii) Computations are the sum of their parts

The second intuition to which I wish to draw attention is that computations are the sum of their parts. It seems intuitively plausible that computations, at least in some sense, have parts. Computations are not undifferentiated wholes: they consist of parts that are connected in ways distinctive to that computation. The parts of a computation are its steps, or subprocesses. For example, a computation that calculates a JPEG compression of an image consists in a series of steps: divide the image into blocks of 8×8 pixels, apply a discrete cosine transformation to each block, quantise the coefficients to reduce the amount of data, perform a Huffman compression on the resulting coefficients, and so on.

[9] See Sudkamp (1998), 259–260 for more on the definition of a Turing machine.
[10] See Boolos et al. (2002), 24–25, for more on this point.

Each of these steps can potentially be broken down further into smaller steps, which themselves can be broken down, and so on, until one reaches the steps that one considers simple.

We saw that I/O equivalence is a necessary but not sufficient condition for computational identity. In order for two systems to perform the same computation, the two systems must be I/O equivalent *and they must achieve their I/O pattern in the same way*. One of the problems raised in Chapter 1 was that it is not clear how to spell out the content of this latter requirement. If computations are made up of parts, then a provisional answer can be given. We can say that two computations work in the same way just in case *they have the same parts and those parts are connected in the same ways*. In other words, two systems perform the same computation just in case they are made up from the same subprocesses, and those subprocesses are connected in the same ways. For example, a system performs a JPEG computation just in case it has the subprocesses described above, and those subprocesses are connected in the same way. I believe that this intuition—that computations are the sum of their parts—is the key to our notion of computational identity.

3.1.3 Outline of problem and solution

The discussion so far has glided over some important issues. Among these are: What is a process? When are two processes the same? What is it for a process to be made up from subprocesses? What is it for two subprocesses to be connected? How does a representation have formal structure? How does representational content contribute to computational identity? The discussion of the previous sections did not answer these questions. Indeed, there are no generally accepted answers within the CTM community. The remainder of this chapter and the next argue for an account that provides answers.

There are many ways in which such an account could be developed. For example, one might attempt to fill in the notion of 'process' by positing *sui generis* entities in the world, processes. On this view, along with whatever else exists, there also exist processes. This approach provides answers to some of the questions above. For instance, one could say that two processes are the same just in case the two corresponding entities in the world are in fact identical, and that a process is made up from subprocesses just in case it is literally the mereological sum of those entities. However, whatever the virtues of such an approach, it would be a Pyrrhic victory for the computationalist. Few philosophers independently believe in processes as *sui generis* entities. An account of computation that posited such entities would be inherently implausible. The aim of this thesis is to develop a notion of computation

that has a broader appeal, a notion that relies only on standardly accepted metaphysical notions.

The account of computation developed below is called the 'process and representation' model (PR-model). The name reflects the fact that the model defines what we mean by our computation talk in terms of processes and representations. The PR-model is intended to formalise the two key intuitions described in Section 3.1.2, and the account of the CTM described in Section 3.1.1. The PR-model elaborates these intuitions and this account into a more substantial account of computation. The PR-model is designed to be customisable. If one objects to a component of the model, say, its analysis of process talk, then one can swap out that component and slide in a different analysis. However, I shall argue that the model is an adequate analysis as it stands. The end products of the PR-model are: (1) an account of what we mean when we say that a system performs a computation; and (2) an account of what we mean when we say that a system performs one computation rather than another.

The PR-model is introduced in two stages. Section 3.2 introduces the basic terms used by the model and discusses their metaphysical commitments. The next chapter defines the model itself in a step-by-step fashion using these basic terms. Two points should be borne in mind.

First, the PR-model specifies what we mean by our computation talk in quasi-formal terms. One might question why such an account need be formal at all. Why can't we say what we mean in plain English? The reason for adopting a formal approach is that an account in English is likely to be misleading. For one thing, the scope of quantifiers in English is often ambiguous. A formal approach avoids these problems and, on balance, is likely to be more readable. However, glosses in English are provided below.

Second, the PR-model is not the only possible account of the semantics of computation talk. Other accounts are possible, and they could save the same intuitions in different ways. The success of the PR-model, like that of any semantic account, has to be judged on its intuitive appeal and its pay-offs for other projects, such as the project of giving an account of the metaphysics of computation. It is worth reiterating that the PR-model is only a semantic account of talk of *implementation* of computation. The PR-model is not a semantic account of all computation talk. Furthermore, the PR-model is only intended as an account of *CTM-style* implementation talk. We use implementation talk in many contexts for many different ends; a general notion of implementation may prove too heterogeneous to be captured by a single account. The PR-model aims only to capture the semantic content of computation talk relevant to the

CTM.[11]

3.2 Preliminary definitions

The PR-model defines the performance of a computation in terms of processes and representations. In this section, accounts are given of both processes and representations.

3.2.1 The relata of representation

Representation involves the obtaining of a relation between two things: that which represents and that which is represented. For example, an oil painting represents Westminster Cathedral just in case a representation relation obtains between two things: a collection of oil marks on canvas and Westminster Cathedral. The collection of oil marks on the canvas is *that which represents*, Westminster Cathedral is *that which is represented*. In what follows, that which represents shall be called a 'representation'. Representations shall be denoted by schematic lower case Greek letters, such as ϕ. In the example above, the oil marks on canvas are a representation. The other side of the representation relation, that which is represented, shall be called the 'referent' or 'content' of a representation. Non-actual contents are allowed—these cases are discussed in Section 3.2.6. The content of a representation, ϕ, shall be denoted as $[\![\phi]\!]$. In the example above, the content of the representation (the oil marks on canvas) is the entity Westminster Cathedral.

3.2.2 Representation tokens

In the PR-model, a representation should be understood as a representation *token*, not a representation type. Potentially anything can count as a representation token. Representations could be, depending on one's views, objects, events, or states of affairs. Whatever theory of representing entities one favours, that theory can be plugged into the PR-model. The PR-model uses tokens rather than types for two reasons. First, many philosophers regard token entities as metaphysically less mysterious than types. Second, criteria of identity for token entities are usually easier to come by than those for types.

The PR-model only requires two assumptions about representation tokens: (1) representation tokens are possible; and (2) there are determinate facts about numerical identity for representation tokens. Let us consider each assumption

[11] However, as we will see in Chapter 5, the PR-model can be applied to some areas of computation talk outside the CTM.

in turn. The first assumption is widely accepted. Most philosophers agree that there *can be* entities in the world that represent, even if they disagree about the nature of those entities and the nature of the representation relation. For the most part of this discussion, I shall simply assume that this assumption is true. For those philosophers who disagree, Section 3.2.8 suggests a way in which the PR-model might still be of use.

The second assumption is that there are determinate facts about numerical identity for representation tokens. This again is widely accepted. An argument for this assumption is given in Section 3.2.5. It is sufficient here to note that although there is often disagreement about the *conditions* of numerical identity for token entities, there is often agreement that there *are* determinate facts about such identity. No matter whether one favours objects, events, or states of affairs, one will almost certainly wish to say that there are determinate facts about numerical identity for those entities. The concern here is broader than the notion of computation. If one does not admit facts about numerical identity for token entities, then it is hard to see how they could be put to useful metaphysical work. It would not be determinate how many such entities existed, and this would have consequences for such metaphysical features as persisting through time, having parts, or being a cause. Each of these metaphysical features is sensitive to facts about the numerical identity of the underlying entities.

One should distinguish strict numerical identity from looser and more idiomatic notions of identity. The definitions below are intended to employ only strict numerical identity. One might wish to weaken the position by allowing weaker identity relations. For example, one might wish to say that two photographs are identical just in case the two photographs look exactly alike. This fits with our everyday talk: one might say that there are two identical photographs on different pages of a photograph album. However, this weaker kind of identity relation, *being the same photograph*, is not strict numerical identity. No matter how similar two photographs are, they are still two numerically distinct objects. It is only the notion of strict and unrelativised numerical identity that is at issue in the PR-model.[12]

[12]Geach (1980) denies that a strict notion of unrelativised identity is possible. I assume, along with Armstrong (1997), 14–16, that Geach is wrong. However, even if Geach is correct, a notion of computation could perhaps still be developed along the lines below. This could be done if an appropriate notion of relative identity can be selected. Alternatively, if as seems likely, there is no single correct notion of relative identity for representation tokens, then the entire account of computation could be indexed to a notion of relative identity, determinable by context or other factors.

3.2.3 The representation relation

We have seen that the PR-model is neutral about the nature of representation tokens. The PR-model is also neutral about the nature of representation relations. There may be more than one kind of metaphysical relation that underlies representation. Some theories of representation claim that there are naturalistic representation relations (e.g. Fodor (1990a)), others claim that representation relations are non-naturalistic (e.g. Davidson (1984)). When we come to consider the metaphysics of computation, these differences will matter a great deal. However, in the current context they can be ignored. All that matters to the PR-model is that a representation relation obtains, i.e. that a representation token is paired with a referent. The details of how this relation obtains are not important.

The PR-model involves the assumption that each representation token represents exactly one thing. There are two worries one might have about this assumption. The first is that there seem to be cases in which a token *does* represent more than one thing. For example, a photograph of Westminster Cathedral might represent *Westminster Cathedral*, *the seat of an archbishop*, and *cathedrals* in general. The second worry is that it is sometimes indeterminate *which* thing a token represents. For example, a fuzzy photograph may not represent any one particular thing determinately at all. Let us consider each worry in turn.

The first worry—that a representation token can represent more than one thing—-can be addressed in the following way. Instead of defining computation *simpliciter*, we can define computation indexed to unambiguous representational content. Collect those things that a representation token represents into a set.[13] Call an 'interpretation' of a token, a selection of a single member from that set. Call an 'interpretation function' of a set of representation tokens, an interpretation of each token in that set. Relative to an interpretation function, a single referent is associated with each representation token. Therefore, if an interpretation function is applied to a set of tokens, then we are guaranteed that each token represents exactly one thing. Hence, we can talk of whether that process qualifies as computational *under an interpretation function*.

Definition 3.1. I is an interpretation function for a set of representation tokens $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ iff for each $\phi_i \in \Phi$, I associates exactly one referent with each ϕ_i. The referent of ϕ_i under I is denoted $[\![\phi_i]\!]_I$.

It should be noted that interpretation functions do not carry a commitment to an 'interpreter' or introduce representation relations where there were none

[13] That the possible referents form a set is guaranteed by the assumption, discussed in Section 3.2.4, that there are determinate facts about numerical identity for referents.

before. An interpretation function is a way of talking about patterns of pre-existing representation relations. Interpretation functions do not 'produce' new representation relations. When we say that a system performs a computation under an interpretation function, all that we mean is that given that each token involved, ϕ_i represents $[\![\phi_i]\!]_I$—which, *inter alia*, it does—then that system performs a computation.

Computations can be realised in many different physical systems. The same computation can be performed by a system made out of silicon and a system made out of tin-cans and string. Conversely, the same system can perform many different computations. As mentioned in Section 3.1.2, one way for this to happen is if the tokens involved represent more than one thing. If a system has multiple representational content then it has the potential for multiple computational identities. The notion of an interpretation function provides a useful way to talk about such cases. We can say that a single system performs different computations under different interpretation functions. This seems a more natural way to treat these cases than to say, for example, that such systems perform a single computation with ambiguous representational content.

What about everyday talk of computation, which does not contain any reference to interpretation functions? A notion of computation *simpliciter* can be defined in terms of the notion of computation relative to an interpretation function. A system performs a computation *simpliciter* just in case it performs a computation under at least one interpretation function. In what follows, we shall only focus on cases of computation relative to an interpretation function. The extension of the results to the notion of computation *simpliciter* is trivial.

The second worry is that it is not determinate whether certain representation tokens represent certain entities or not. Here is an example. An OR gate takes voltages as input and yields voltages as output. Suppose that a voltage signal of 5 V represents *1*, and a voltage signal of 0 V represents *0*. Real-world electrical signals rarely take exactly integer values; it is possible for a signal to deviate from 5 V in many ways. Therefore, we should modify our representational conventions by, for example, requiring that signals of 5 ± 0.4 V represent *1*, and signals of 0 ± 0.4 V represent *0*. However, this assumes that we are able to choose a sharp cut-off point, such as ±0.4 V. Typically, an OR gate will not have a sharp cut-off point in its behaviour. Do we wish to introduce a sharp cut-off in its representational content? What if the OR gate responds correctly to inputs slightly outside the ±0.4 V range? Do we wish to say that a signal of 4.599999999 V does not represent *1* even though it differs from a signal that does represent *1* by a minute amount?

Two options are possible. One is to insist on a sharp-cut off. The other is to accept vagueness in whether signals represent or not. On the latter view, it is

indeterminate whether a signal of 4.599999999 V represents *1* or not. Electrical engineers generally employ only the first option in their conventions. The alternative—that signals with indeterminate content could occur—would make it difficult to predict the function computed.[14] Nevertheless, it is conceivable that tokens could occur for which it is vague whether they represent an entity $[\![\phi_i]\!]_I$ or not: we might choose to adopt a representational scheme that satisfies the second option. What should we say about such cases?

I believe we should say that in such cases the identity of the computation itself is vague, and it is vague to the extent that its underlying representational content is vague. The respect in which representational vagueness transmits to computational identity will depend on the respects in which computational identity depends on representational content. A detailed description of this relation is given in the PR-model. For the most part of the discussion below however, like the electrical engineers, we shall assume that representation tokens either clearly represent or do not represent a referent. Extending the discussion to the vague cases is not hard.[15]

3.2.4 Representational contents

As already mentioned, the PR-model is neutral concerning the nature of representation tokens. Potentially any entity can qualify as a representation token; the ultimate nature of representation tokens depends on one's general metaphysical views. The PR-model is similarly neutral about the nature of the referents. Potentially any entity can qualify as the referent of a representation token. Only one restriction is placed on referents: that there are determinate facts about numerical identity for such entities. For any two referents, $[\![\phi]\!]_I$, $[\![\psi]\!]_I$, it must be determinate whether $[\![\phi]\!]_I = [\![\psi]\!]_I$ or $[\![\phi]\!]_I \neq [\![\psi]\!]_I$. Provided that this assumption is met, potentially anything can count as a referent. Representation tokens are allowed to represent anything, including themselves.

The notion of representational content is intended to be understood in this context purely extensionally. Whether two representation tokens have the same content depends only whether the referents of those tokens are numerically identical, *not* on the way in which those entities are referred to. For example, we can say that $[\![\text{'The morning star'}]\!]_I = [\![\text{'The evening star'}]\!]_I$, even though the two tokens represent their referent in different ways.

[14] See Horowitz and Hill (1989), 473.

[15] Note that although the preceding example of the OR gate deals with a case in which it is vague whether a token represents or not, the same argument applies to cases in which it is vague whether a token represents one entity rather than another.

3.2.5 Determinate numerical identity relations

The PR-model requires determinate numerical identity relations between representation tokens and between representational contents. For any two tokens ϕ and ψ, it must be determinate whether $\phi = \psi$ and whether $[\![\phi]\!]_I = [\![\psi]\!]_I$. If ϕ and ψ have clear numerical identity conditions, then it will be clear whether such a numerical identity relation obtains. For example, if ϕ and ψ represent numbers, then it will be clear whether $[\![\phi]\!]_I = [\![\psi]\!]_I$ or $[\![\phi]\!]_I \neq [\![\psi]\!]_I$, since numbers have clear numerical identity conditions. Similarly, if ϕ and ψ represent numerals, then it will be clear whether $[\![\phi]\!]_I = [\![\psi]\!]_I$ or $[\![\phi]\!]_I \neq [\![\psi]\!]_I$, since numerals have clear numerical identity conditions. In cases like these, the conditions above (at least regarding representational content) are clearly satisfied, and the PR-model can be applied without question. However, in cases in which the relevant entities *do not* have clear identity conditions, the determinacy of the identity relation, and therefore the applicability of the PR-model, may appear under threat. Many of the entities with which we are familiar have unclear numerical identity conditions. Therefore, a possible objection to the PR-model is that these entities cannot qualify as representation tokens or representational contents, and so cannot participate in real-world computation. Hence, according to the PR-model, many intuitive cases of computation do not count as computation. In this section, I argue that this objection is mistaken. The PR-model is compatible with computation with entities that have unclear identity conditions.

Lack of clarity about identity conditions can arise for one of three reasons. First, ignorance: the identity conditions of an entity may be unclear because we are ignorant of its identity conditions. Second, semantic indeterminacy: the identity conditions of an entity may be unclear because it is vague or indeterminate *which* entity we are referring to. Third, metaphysical indeterminacy: the identity conditions of an entity may be unclear because it is *in fact* indeterminate whether that entity is numerically identical to another entity or not. Only the final source of lack of clarity in identity conditions is a problem for the PR-model. In that case, the identity conditions are unclear *because there really are no determinate facts about numerical identity*. This violates the condition above for application of the PR-model. I suggest that there are good reasons for thinking that such cases do not occur. We can attribute lack of clarity in identity conditions to sources other than metaphysical indeterminacy. If lack of clarity in identity conditions is due only to ignorance or semantic indeterminacy, then such lack of clarity poses no threat to the applicability of the PR-model to real-world computation.

Let us consider each potential source of lack of clarity in turn.

First, ignorance. We are ignorant of the identity conditions of many entities. Conditions of numerical identity are often a matter of philosophical or natural scientific enquiry.[16] An example of an entity whose identity conditions have been a matter of philosophical enquiry is the ship of Theseus. Let the representation token ϕ represent the original ship of Theseus. Let the representation token ψ_1 represent the ship that is the result of gradually replacing every plank in the original ship. Let the representation token ψ_2 represent the ship that is constructed out of the planks that were removed from the original ship. Does $[\![\phi]\!]_I = [\![\psi_1]\!]_I$ or $[\![\phi]\!]_I = [\![\psi_2]\!]_I$?[17] The answer is unclear. One might argue that this lack of clarity is not due to any indeterminacy in the identity relation itself, or in the reference of the term 'ship', but to ignorance on our part. A possible, albeit controversial, position on such identity statements is that such statements are unclear simply because we lack epistemic access to their truth value.

Second, semantic indeterminacy. Quine (1980a) claims that identity statements such as those concerning the ship of Theseus have unclear truth value, not because of any epistemic limitation on our part, but because of semantic indeterminacy inherent in our talk of ships. According to Quine, ships are aggregates of atoms. The representation ϕ could refer to any one of two distinct aggregates, each of which is broadly ship-shaped. As far as the problem has been stated, it is indeterminate to which of these two aggregates the tokens refer. This semantic indeterminacy need not be accompanied by metaphysical indeterminacy. Quine, for one, claims that whether two entities in the world are numerically identical is determinate, even though it may be indeterminate which of these entities we refer to.

Finally, metaphysical indeterminacy. On this view, our terms refer, but they refer to vague objects. Vague objects are objects for which it is indeterminate whether or not they are identical to each other or to anything else. Evans (1978) and Salmon (1981) have argued against the possibility of vague objects.[18] Briefly, their argument is as follows. Suppose that an entity, a is b. Trivially, b is such that it is determinately true that it is b. By Leibniz's Law, a has every property b has, so a is such that it is determinately true that it is b. Therefore, if our original assumption is correct, i.e. if a is b, then it is determinately true that a is b. Taking the contrapositive of the above conditional, if it is not determinately true that a is b then a is not b. The defender of vague objects believes that for some a and b, it is not determinately true that a is b. However, she refuses to assert the consequent of the conditional: that a is not b. Therefore, her position

[16] See Noonan (2003) on the conditions of personal identity as a matter of philosophical enquiry. See Putnam (1975a) on the identity conditions of natural kinds as a matter of scientific enquiry.

[17] Molesworth (1845), Part 2, Ch. 2, 135.

[18] The argument is clarified and defended in Lewis (1988); Salmon (1986); Stalnaker (1988); Wiggins (1986).

is logically incoherent.[19]

Vague objects are also objectionable for a methodological reason. A philosopher who posits vague objects not only needs to show that they are possible, but also that they are necessary for doing useful metaphysical work. A plausible default assumption is that vague objects do *not* exist. A defender of vague objects has to overturn this assumption; she has to show that vague objects are needed to accommodate features of the world that cannot be accommodated in any other way. She faces an uphill struggle in this task. Even if Evans and Salmon are incorrect, there seems no compelling reason for positing vague objects.

Let us assume that Evans and Salmon are correct. Therefore, ignorance and semantic indeterminacy are the only sources of unclear identity conditions. Under this situation, as discussed above, entities with unclear identity conditions pose no threat to the applicability of the PR-model.

3.2.6 Non-actual tokens and contents

Another possible objection to the PR-model concerns non-actual tokens and content. The problem is that it is not clear how there could be determinate facts about numerical identity for non-actual entities. Quine expresses this worry:

> Take, for instance, the possible fat man in that doorway; and, again, the possible bald man in that doorway. Are they the same possible man, or two possible men? How do we decide? How many possible men are there in that doorway? Are there more possible thin ones than fat ones? How many of them are alike? Or would their being alike make them one? Are no *two* possible things alike? Is this the same as saying that it is impossible for two things to be alike? Or, finally, is the concept of identity simply inapplicable to unactualized possibles? (Quine, 1980b, 4)

If there are no determinate facts about numerical identity for non-actual entities then, according to the PR-model, such entities cannot participate in computations.

A possible response to this worry is to limit the PR-model to cover only actual entities. Unfortunately, this does serious damage to our intuitions about computation. This damage occurs in two ways. First, it seems essential to our notion of computation that even if, as a matter of contingent fact, certain tokens never occur, a process *would* have behaved in a suitable way if those tokens *were* to have occurred. Second, there seems to be no reason why there cannot be

[19] This version of the argument is adapted from the helpful summary of Williamson (1990).

computations with tokens that represent non-actual entities—there seems no reason why there cannot be computations about Sherlock Holmes or a talking donkey.

A different way to deal with the problem is to claim that non-actual entities are ontologically on a par with actual entities, and hence have identity conditions that are just as determinate. Lewis (1986b) claims that non-actual entities are real in this way. For Lewis, the problem described above does not arise: relations of numerical identity for non-actual entities are just as determinate as those for actual entities. On Lewis's view, the indeterminacy that Quine indicates above is only semantic indeterminacy: the identity of the fat man in the doorway is unclear because it is unclear *which* non-actual fat man Quine is referring to. The referent of Quine's expression is indeterminate in the same way as the referent of 'the British person in the room' is indeterminate if used without further information in a room full of British people.

Many philosophers try to steer an intermediate course between Quine's actualism and Lewis's realism about non-actual entities. Typically, these positions claim that non-actual entities are real, and therefore have determinate identity conditions, but that their nature is different from that of actual entities. If these theories can be made to work, then they would allow the PR-model to employ non-actual entities without incurring Lewis's metaphysical commitments. Here is a summary of current approaches.

Plantinga (1974) claims that possible worlds are abstract objects called maximally consistent states of affairs. All possible states of affairs exist, but only one, the actual world, obtains. Non-actual objects can be accommodated by adding individual essences. Individual essences are properties, such as *being identical to Socrates*, that all and only objects that are numerically identical share. According to Plantinga, individual essences can exist uninstantiated. Therefore, individual essences can play the role of non-actual objects. Stalnaker (1976) claims that possible worlds are uninstantiated properties. On Stalnaker's view, properties can be structured, and so properties can again play the role of non-actual objects. Another theory is that possible worlds are maximally consistent sets of sentences. On this approach, non-actual objects are names in the language in which the sentences are expressed. These names may themselves be abstract objects, such as sets. Another approach is modal fictionalism. On Rosen's (1990) fictionalist theory, modal claims are true or false depending on whether they are true or false according to Lewis's theory, although Lewis's theory is assumed to be false. If modal fictionalism is correct, then identity statements in the PR-model are disguised statements about Lewis's fiction. A statement about two non-actual entities being numerically identical is true just in case it is true in the fiction of Lewis's theory that the two non-actual entities

are numerically identical. The success of these various approaches is unclear. Non-actual entities are an outstanding problem in metaphysics. In what follows, it will be assumed that determinate facts about numerical identity for non-actual entities can be secured somehow.

It is worth emphasising that facts about numerical identity for non-actual entities is not just a problem for the PR-model. Any theory that quantifies over non-actual entities faces the same problem. For example, possible world semantics often assume the existence of a set, D, the domain of all individuals. This set contains all possible individuals in all worlds.[20] The members of a set must, by definition, be numerically distinct. Therefore, the possibility of forming such a set D depends on there being determinate facts about numerical identity for non-actual individuals. Determinate facts about numerical identity are also required by metaphysical theories. Current theories of causation, supervenience, and persistence, depend on such numerical identity relations.[21]

3.2.7 Processes

As well as using the notion of representation, the PR-model also uses the notion of a process. For the purposes of the PR-model, what meant by a 'process' can be captured by a list of counterfactual conditionals concerning tokens. Note that this model of processes is intended only to apply to processes in the context of computation talk. There are many other contexts in which the word 'process' is used that cannot be treated in this way.

Domains and ranges of processes

A process has an associated domain $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$, and range $\Psi = \{\psi_1, \psi_2, \ldots, \psi_m\}$. The domain and range of a process are sets of representation tokens. These sets of tokens may be finite or infinite. The domain and range may overlap, and may even be the same set. A process should be understood as performing a mapping from tokens in its domain to tokens in its range: if a certain token ϕ from its domain is presented to a process, then after a finite amount of time the process yields a token ψ from its range. The conditional underlying the mapping should be understood as having counterfactual force: if a token ϕ from the domain *were* to be presented to a process, then after a finite amount of time the process *would* yield a token ψ from its range. As discussed in Section 3.2.2, tokens can be anything: objects, events, states of affairs—whatever one thinks exists. An example of a real-world process is a

[20] See Lewis (1970b); Montague (1974).

[21] See Lewis (1986b), Ch. 1 for an overview of areas in which determinate numerical identity relations are required for non-actual entities.

NOT gate. A NOT gate maps signals of 0 V to signals of 5 V, and signals of 5 V to signals of 0 V.[22]

Definition of process—first attempt

If we wish to define a process, it is not enough to specify that process's domain and range. We also need to specify the dependency relations between the members of its domain and range. We need to say *which* inputs yield which outputs. In order to characterise this pattern of dependency relations, let us associate a set of ordered pairs, Ω, with each process. For every pair of tokens $(\phi, \psi) \in \Omega$, the following is true: if ϕ were presented as input to the process, then the process would yield ψ as output.

Let us make a first attempt at an account of process talk. Let us say that $P(\Omega)$ is a process just in case for each (ϕ_i, ψ_i) in Ω:

> If ϕ_1 were presented as input, then ψ_1 would be yielded as output.
>
> If ϕ_2 were presented as input, then ψ_2 would be yielded as output.
>
> If ϕ_3 were presented as input, then ψ_3 would be yielded as output.
>
> .
>
> .
>
> .

Perhaps additional counterfactuals are needed to characterise a process. The above series stipulates that if ϕ_1 were presented, then ψ_1 would be yielded as output. Should we also require that if ϕ_1 were *not* presented, then ψ_1 would *not* be yielded as output? According to Lewis (1970a), a pair of positive and negative counterfactuals is needed in order to characterise causal dependence. In order for B to causally depend on A, it must be that: (i) if A were to occur then B would occur, and (ii) if A were not to occur then B would not occur. However, a pair of positive and negative counterfactuals is not needed to characterise what we mean by a process. The series of positive counterfactuals given above is sufficient.

Consider a NOT gate. If 5 V is presented as input to the NOT gate, then it yields 0 V as output. If 0 V is presented as input to the NOT gate, then it yields 5 V as output. Suppose that the input to the NOT gate is *neither* 0 V *nor* 5 V. Why should we intend anything about its output? Typically, we not do not. These situations are given the technical name of 'don't cares' by electrical engineers.[23] The negative counterfactual conditional can also be violated in another way. A

[22] Whether these inputs and outputs are events or instantiations of properties depends on the correct account of the metaphysics of electrical signals.

[23] Hayes (1993), 308–309.

process may associate two different input tokens ϕ_1, ϕ_2 with the same output token ψ. Therefore, if ϕ_1 were presented as input, then ψ would be yielded as output. However, the corresponding negative counterfactual conditional—if ϕ_1 were not presented, then ψ would not be yielded as output—is false, since ϕ_2 might have been presented and ψ yielded. Therefore, a negative series of counterfactuals is not needed.

We can therefore conclude that process-like dependence is weaker than causal dependence—at least on Lewis's (1970a) analysis of causal dependence. For Lewis, lack of causal dependence is no bar to a relation being causal.[24] However, even this condition need not be true of processes. In some cases, the input to a process may cause its output. In other cases, the input may not cause the output but the two may counterfactually covary due to a common cause. In yet other cases, the relation linking input to output may be entirely non-causal. For example, the relation could be that of part to whole, or a functional law-like relation such as that relating pressure and volume of a gas at constant temperature. Potentially any counterfactual supporting regularity will do.

Presenting input and yielding output

The definition above used the terms 'presented' and 'yielded'. What does it mean for a token to be presented to a process or yielded by a process? One way of fleshing out these notions is in terms of spatiotemporal regions. If a token is presented to a process, then that token occurs at a certain time and at a certain location. Similarly, if a token is yielded by a process, then that token occurs at a certain time and at a location. The converse is also true. If a token occurs at a certain time and location associated with a process, then that token is presented to that process. Similarly, if a token occurs at a time and location associated with a process, then that token is yielded by that process. I suggest that talk of presentation and yielding can be captured in terms of talk of spatiotemporal regions. We can say that a token is 'presented' to a process just in case that token occurs in a certain spatiotemporal region A associated with the process—call this the 'input region' of the process. We can say that a token is 'yielded' by a process just in case that token occurs in a certain spatiotemporal region B associated with the process—call this the 'output region' of the process.[25]

In addition to providing an account what we mean by presentation and yielding, this approach also allows us to make sense of the idea that processes are spatiotemporally located. For example, it is conceivable that two AND gates could be perfect physical duplicates. These AND gates can be distinguished

[24] On Lewis's view, A causes B provided there is a chain of causally dependent events between A and B, and even if B is not itself causally dependent on A. See Lewis (1970a).

[25] All conditions should be understood as stated tenselessly.

by their spatial location. One AND gate may be located inside my desktop computer and be sensitive to a certain spatiotemporal region. The other AND gate may be located inside the Mars Pathfinder probe and be sensitive to a different spatiotemporal region. Both AND gates share the same I/O behaviour Ω, but they operate on different spatiotemporal regions.

There is no restriction on the shape of the input or output regions. The input and output regions may overlap with each other, either partially or entirely, and they may not be internally connected or continuous. The input and output regions of one process may overlap, either partially or entirely, with the input and output regions of another process.

We can revise the definition above by replacing 'presented' and 'yielded' with conditions involving spatiotemporal regions. On the revised definition, a process is characterised by a triple (Ω, A, B): a set of I/O pairs Ω, an input region A, and an output region B. Let us say that P (Ω, A, B) is a process just in case for each (ϕ_i, ψ_i) in Ω:

> If ϕ_1 were to occur in region A, then ψ_1 would occur in region B.
>
> If ϕ_2 were to occur in region A, then ψ_2 would occur in region B.
>
> If ϕ_3 were to occur in region A, then ψ_3 would occur in region B.
>
> .
>
> .
>
> .

The revised definition employs the term 'occurs' as a primitive predicate. This term should be understood as a placeholder: it is to be replaced by the term appropriate to the tokens in question. That term will vary depending on one's view of tokens. If one believes that tokens are objects, then 'occurs', should be replaced by 'is present'; if states of affairs, then 'obtains'; if events, then 'occurs' need not be changed; and so on. The predicate 'occurs' marks the place of whatever predicate is most appropriate to the metaphysics of the tokens in question.

Definition of process—second attempt

The definition is still not unsatisfactory. We need to add three more conditions in order to capture the required notion of a process.

First, we need to add the requirement that the output of a process occurs some finite time after its input. The output should not occur before, or simultaneously with, the input, and it should not be delayed indefinitely. There should be a characteristic time delay associated with each I/O pair. If an input

token ϕ is presented to a process P, then an output token ψ should be yielded after some finite time t. It is conceivable that two processes could have the same set of I/O pairs, be sensitive to the same spatiotemporal regions, and yet differ in their time delays—one process could be fast and the other slow. The definition above needs to be modified to accommodate such time delays. This can be done by redefining the set Ω. Instead of defining Ω as the set of I/O pairs (ϕ, ψ), we can define Ω as a set of ordered triples (ϕ, ψ, t), where t is the time delay associated with ϕ and ψ. The definition above can then be modified: P (Ω, A, B) is a process just in case for each (ϕ_i, ψ_i, t_i) in Ω, if ϕ_i were to occur in region A, then ψ_i would occur in region B time t_i after ϕ_i.

Second, the definition fails to capture the correct modal relations between input and output. This is easiest to see in the case of actual tokens. Imagine that two tokens ϕ_1, ϕ_2 actually occur in the input region A of a process at times t_1 and t_2 respectively, and that ψ actually occurs in the output region B at time t_3. For the sake of argument, assume that Lewis (1973) is correct that a counterfactual is true if both its antecedent and consequent are true. Therefore, the following counterfactuals are true: (i) If ϕ_1 were to occur in region A, then ψ would occur in region B; and (ii) If ϕ_2 were to occur in region A, then ψ would occur in region B. Hence, according to the definition above, we have a process P $(\{(\phi_1, \psi, t_3 - t_1), (\phi_2, \psi, t_3 - t_2)\}, A, B)$.

Now suppose that additional information is provided. The additional information is that if ϕ_2 were to occur alone (without ϕ_1), then ψ would not have occurred. The token ψ only occurred because ϕ_1 was present. Therefore, although the counterfactual 'If ϕ_2 were given as input, then ψ would be yielded as output' is true, it is only true because of the extraneous presence of ϕ_1 in the actual world. If ϕ_1 were absent, then there would have been no ψ. Intuitively, even though the definition above is satisfied, the link between ϕ_2 with ψ is not process-like.

The definition can be modified to handle these cases. This can be done by requiring that a process-like dependency between ϕ and ψ obtain only if the counterfactual holds *no matter what other tokens occur in the input spatiotemporal region*. A process-like dependency between ϕ and ψ only obtains if, given that ϕ occurs in region A, then no matter what other tokens might be in A before or after ϕ, ψ would occur in region B a time t after ϕ. This clause need only consider tokens that occur before or after ϕ, because the behaviour of processes is typically undefined for input tokens that are presented simultaneously.

One might wonder whether a similar problem arises for entities that are *not* tokens of the domain. For example, suppose that ϕ_1 were to yield ψ only if an extra entity α were present too. If α were not present, then ϕ_1 would not yield ψ. Such cases can be dealt in one of two ways. Either α and ϕ_1 can be

classified together as a joint input token. Or, if α is needed for a wide variety of input tokens, then α can be treated as part of the background conditions for the process to exist at all. Certain background conditions need to be satisfied in order for a process to exist. For example, the Big Bang needs to have occurred, and the local environment needs to be favourable for the dependencies involved in the process. What is involved in satisfying these conditions is not rightly speaking part of that process's input. Background conditions are preconditions for the counterfactual dependencies involved in the process to obtain at all. Sensitivity to background conditions is a feature common to all counterfactual conditionals; it is not problem specific to processes. As such, it should be dealt with by one's account of counterfactual conditionals, not by a theory of processes.

The final condition is that a process should be deterministic. A process is deterministic just in case each token in the domain is paired with a single token in the range. Determinacy is not a conceptual necessity for processes: it is easy to imagine processes that are not deterministic. However, for the sake of simplicity in what follows we shall restrict discussion to deterministic processes. The definition above can be modified to include this restriction. This can be done by specifying that the same input token always yields the same output token with the same time delay: $\forall(\phi, \psi, t), (\phi, \psi', t') \in \Omega$, $\psi' = \psi$ and $t' = t$.

We can now give the final definition of a process. $P\ (\Omega, A, B)$ is a process just in case $\forall(\phi, \psi, t), (\phi, \psi', t') \in \Omega$, $\psi' = \psi$ and $t' = t$, and for each (ϕ_i, ψ_i, t_i) in Ω:

> If ϕ_1 were to occur in region A, then no matter what other ϕ might be in A before or after ϕ_1, ψ_1 would occur in region B time t_1 after ϕ_1.
>
> If ϕ_2 were to occur in region A, then no matter what other ϕ might be in A before or after ϕ_2, ψ_2 would occur in region B time t_2 after ϕ_2.
>
> If ϕ_3 were to occur in region A, then no matter what other ϕ might be in A before or after ϕ_3, ψ_3 would occur in region B time t_3 after ϕ_3.
>
> .
>
> .
>
> .

This definition can be collapsed to a single condition:

Definition 3.2. $P\ (\Omega, A, B)$ is a process iff $\forall(\phi, \psi, t) \in \Omega$, if ϕ were to occur in region A, then no matter what other ϕ' might be in A before or after ϕ, ψ would occur in region B time t after ϕ. Furthermore, we shall assume that all processes are deterministic: $\forall(\phi, \psi, t), (\phi, \psi', t') \in \Omega$, $\psi' = \psi$ and $t' = t$.

This definition provides an account of processes in terms of tokens, counterfactual dependence, and spatiotemporal regions. This is the definition that we shall work with in the PR-model.

Final remarks

The domain and the range of a process can be defined in terms of Ω:

Definition 3.3. If P (Ω, A, B) is a process then:

- The domain Φ of $P = \{\phi : (\phi, \psi, t) \in \Omega\}$
- The range Ψ of $P = \{\psi : (\phi, \psi, t) \in \Omega\}$

Two extra conventions are adopted in subsequent chapters. First, if P (Ω, A, B) is a process then we shall say that P is a 'mapping' from a domain Φ to a range Ψ. This shall be symbolised: $\Phi \mapsto \Psi$. Second, if a process pairs a particular input token ϕ with a particular output token ψ, then we shall say that the process 'yields ψ if presented with ϕ'. This shall be symbolised: $\phi \vdash_P \psi$. The $\vdash$-relation is defined: if P (Ω, A, B) is a process then $\phi \vdash_P \psi$ iff $\exists t$ such that $(\phi, \psi, t) \in \Omega$.[26]

3.2.8 Extreme positions

The preceding sections have assumed: (1) representation tokens and relations are possible; (2) numerical identity relations between representation tokens and their content are determinate; (3) non-actual entities can be representation tokens or contents; and (4) counterfactual dependence relations exist. These are all widely tolerated assumptions, at least within a broadly realist framework. But what if one denies them, can one still make sense of computation in terms of the PR-model? An initial reaction might be that computation must be understood in some other way. However, this would be too hasty. Even if an anti-realist denies these assumptions, she still has to make sense of the fact that we often talk *as if* they were true. For example, an anti-realist who denies that representation is possible still has to make sense of the fact that we have a rich and varied representation talk. Provided she acknowledges this challenge, the anti-realist can make use of the PR-model suitably interpreted. She can appeal to the PR-model to give a semantic account of our computation talk, but instead of treating the terms of the PR-model as basic, she can give a further account of what is meant by representation, identity, and counterfactual dependence

[26] The turnstile symbol ('$\vdash$') is intended to be suggestive of proof-theoretic entailment. However, the $\vdash$-relation is not proof-theoretic entailment. Proof-theoretic entailment is an abstract relation between expressions in a formal language. The $\vdash$-relation is a counterfactual dependence relation between real-world entities.

that is compatible with her anti-realist metaphysics. Although the PR-model comports well with realism about computation, PR-model semantics need not be barred to the anti-realist.

Chapter 4

The PR-model

The PR-model provides an account of what we mean by our computation talk in terms of processes and representations. According to the PR-model, when we say that something is a computation, what we mean that a particular kind of process is in operation. That kind of process is *representational*. Representational processes take representation tokens as input, yield representation tokens as output, and their I/O pattern preserves representational content. Computational processes are a type of representational process: not all representational processes are computational, but all computational processes are representational. For convenience, I shall use the term 'computation' interchangeably with 'computational process'. Sections 4.2 to 4.3 specify the conditions for a representational process to qualify as a computational process.

4.1 Representational processes

A representational process P is defined as follows:

Definition 4.1. P (Ω, A, B): $\Phi \mapsto \Psi$ is a representational process under interpretation function I iff P is a process and:

1. Each $\phi \in \Phi$ and $\psi \in \Psi$ represents at least one thing, and out of those things, interpretation function I picks out exactly one referent $[\![\phi]\!]_I$ or $[\![\psi]\!]_I$ for each $\phi \in \Phi$ and $\psi \in \Psi$.

2. $\forall \phi, \phi' \in \Phi$, if $\phi \vdash_P \psi$ and $\phi' \vdash_P \psi'$ and $[\![\phi']\!]_I = [\![\phi]\!]_I$, then $[\![\psi']\!]_I = [\![\psi]\!]_I$.

A representational process maps representation tokens to representation tokens in a way that preserves relations between representational content. Condition (1) of Definition 4.1 states that the input and output tokens of a representational process have representational content. An interpretation function guarantees

unambiguous content for each token. Condition (2) of Definition 4.1 states that a representational process preserves relations between representational content. If two input tokens ϕ, ϕ' represent the same thing, then their two corresponding output tokens also share their respective representational content.

Many cognitive processes are representational processes. For example, the process of syntax parsing is a representational process. Plausibly, the inputs and outputs of a syntax parsing process are electrical or chemical signals inside the brain. However, whatever the physical nature of those signals they must have representational content: the input must represent a stream of phonemes, and the output must represent a grammatically parsed sentence. If the inputs and outputs do not represent, then it is difficult to see how the elements of syntax parsing—phonemes and grammatical structures—can even be introduced into a system of neurons and chemicals. The physical nature of the inputs and output signals may vary slightly on different occasions. For example, it might be that the firing rate of certain neurons can vary within certain limits, or the concentration of a neurotransmitter can vary within certain tolerances. However, it is generally assumed that whatever physical variation there is, if two input signals represent the same thing—the same stream of phonemes—then the two output signals of the process must represent the same thing—the same grammatical structure.[1] If this condition is not met, then the process does not perform syntax parsing as conventionally understood. Therefore, syntax parsing satisfies the conditions above for being a representational process. The same argument applies to other cognitive processes.

One might wonder how it is possible for a physical process to satisfy condition (2) of Definition 4.1. How can a physical process be sensitive to the representational content of its input? As we saw in Section 3.1.1, computation provides an answer. Computational processes are sensitive to the representational content of their input by being sensitive to formal features that covary with that representational content. Therefore, computation provides a way in which condition (2) can be satisfied. In other words, computation provides a way in which representational processes are possible. The rest of this chapter specifies what it means for a process to be computational.

4.2 Unary computational processes

For the moment, let us restrict attention to processes that take *single* tokens as input and yield *single* tokens as output. Call these processes 'unary processes'.

[1] Provided that the cognitive process is operating correctly and the nature of the process itself is not modified across the two inputs.

In Section 4.3, the model is generalised to processes with multiple inputs and outputs. A unary computational process is defined as follows:

Definition 4.2. $P\ (\Omega, A, B)$ is a unary computational process under interpretation function I iff there are a finite number of unary representational processes $p_1, \ldots, p_n$ under I such that:

1. $p_1, \ldots, p_n$ span process P.
2. $p_1, \ldots, p_n$ are simple processes.
3. Each p_i operates on structured representations under I.

Definition 4.2 employs three special terms: 'span', 'simple process', and 'structured representation'. Each of these terms requires further definition. Sections 4.2.1 to 4.2.4 provide these definitions.

4.2.1 Spanning

Processes $p_1, \ldots, p_n$ span a process P just in case P is 'made up' from $p_1, \ldots, p_n$ in the intuitive sense suggested in Section 3.1.1. The notion of spanning is intended to provide a definition for the notion of being 'made up' from. A collection of unary process $p_1, \ldots, p_n$ span a process P just in case four requirements are met. First, the input region of p_1 coincides with the input region of P. Second, the output region of p_n coincides with the output region of P. Third, the output region of each p_i (except p_n) is contained within the input region of p_{i+1}. Fourth, if a process p_i (except p_n) yields a token ψ as output, then that token is the input token to the next process p_{i+1}, with the time delays of the overall process respecting the time delays of the intermediate processes. A number of minor extra conditions are also required. These conditions are fully described by Definitions 4.3 to 4.4 below.

Let us define an operator, denoted '$\circ$', that joins two processes to form a larger process. This joining operator is defined as follows:

Definition 4.3. $P = Q \circ R$ iff P, Q, R are processes and:

1. $P\ (\Omega, A, D) : \Phi \mapsto \Psi$, $Q\ (\Omega_1, A, B) : \Phi \mapsto \Theta$, $R\ (\Omega_2, C, D) : \Gamma \mapsto \Psi$
2. $\Theta \subset \Gamma$ and region B is entirely contained in region C
3. $\forall(\phi, \psi, t) \in \Omega$, $\exists(\phi, \theta, t_1) \in \Omega_1, \exists(\theta, \psi, t_2) \in \Omega_2$, such that $t = t_1 + t_2$.

The first condition of Definition 4.3 states that the input region of Q coincides with the input region of P, and that the output region of R coincides with the output region of P. The second condition states that the range of Q is a subset

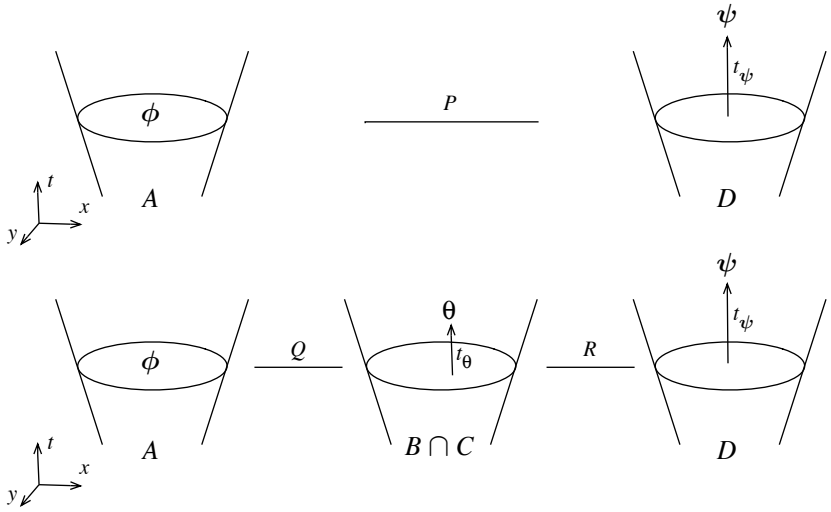

Figure 4.1: An illustration of a process, P, being spanned by two subprocesses, Q, R. The x, y axes represent spatial dimensions and the t axis represents time. Input and output spatiotemporal regions are marked as ringed sections. The upper and lower diagrams describe the same block of spacetime, showing $P = Q \circ R$. An example input token ϕ, output token ψ, and intermediary token θ are shown, along with their respect time delays, t_θ, t_ψ.

of the domain of R, and that the output region of Q is contained within the input region of R. The third condition states that for any I/O triple (ϕ, ψ, t) of P: if ϕ were presented to Q, then an intermediate token θ would be yielded by Q, which would then result in an output token ψ being yielded in the output region of P. The time delays match: the time delay for the overall process P is the sum of the time delays of Q and R. Only if all three conditions are met does $P = Q \circ R$.

It is worth noting that the joining operator is not commutative: if $P = Q \circ R$ then it may not be that $P = R \circ Q$. However, it can be shown that the joining operator is associative: if $P = (p_1 \circ p_2) \circ p_3$, then $P = p_1 \circ (p_2 \circ p_3)$, and vice versa. Since the operator is associative, brackets are not needed in specifying multiple joins, e.g. $P = p_1 \circ p_2 \circ p_3$.

We can now define spanning using the joining operator:

Definition 4.4. A sequence of unary processes $p_1, \ldots, p_n$ span a process P iff $P = p_1 \circ \ldots \circ p_n$.

Three remarks should be made about Definition 4.4.

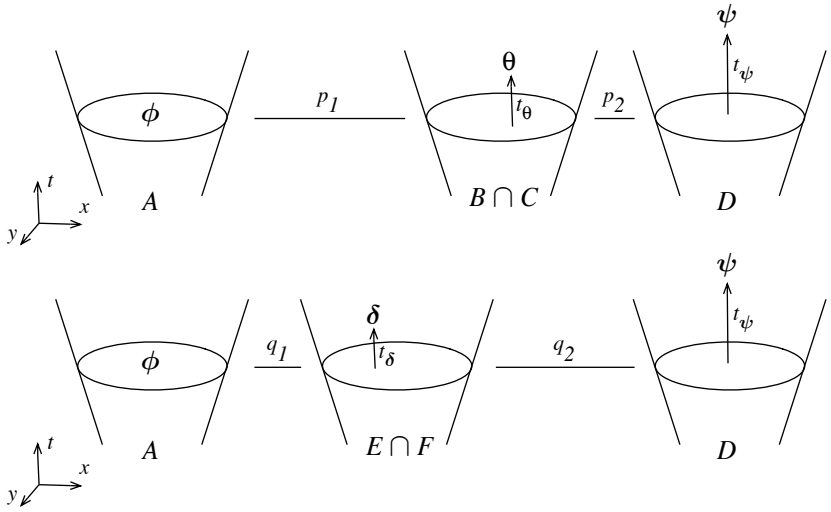

Figure 4.2: Two distinct sequences of processes may span P. Process P is spanned by p_1, p_2, and q_1, q_2, such that it is both true that $P = p_1 \circ p_2$ and $P = q_1 \circ q_2$. The upper and lower diagrams describe the same block of spacetime, showing that distinct intermediate processes can coexist without conflict. In this case, the two processes involve different spatiotemporal regions, $B \cap C$ and $E \cap F$, different tokens, θ and δ, and different time delays, t_θ and t_δ.

First, it is consistent with Definition 4.4 that a process be spanned by more than one sequence of subprocesses. For a process P, there may be two distinct sequences $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$ such that $p_1, \ldots, p_n$ span P and $q_1, \ldots, q_m$ span P. Different sequences of intermediate tokens and dependency relations can coexist without conflict. In other words, there can be more than one pattern of intermediate tokens and counterfactual relations that obtain between a process's overall input and overall output. An example is shown in Figure 4.2. As discussed in Section 5.1.8, this is one way in which a system can perform more than one computation at a time.

Second, although a process can be spanned by more than one sequence of subprocesses, it is not true that a process is spanned by *any* sequence of subprocesses. The conditions on spanning are non-trivial. Among other things, the appropriate intermediate tokens must exist and the appropriate dependency relations must obtain. Furthermore, in order for a process to qualify as computational, its spanning subprocesses must be *representational*. Therefore, the intermediate tokens must represent, and their dependency relations must preserve representational content. There are not many subprocesses that satisfy

this condition.

Third, an important property of the joining operator is that it preserves the representational character of processes. If two representational processes Q, R are joined to form a process P, then process P is *ipso facto* representational. This result can be shown as follows:

Result 4.1. If Q, R are representational processes under I, and $P = Q \circ R$, then P is a representational process under I.

If $Q\ (\Omega_1, A, B) : \Phi \mapsto \Theta$ and $R\ (\Omega_2, C, D) : \Gamma \mapsto \Psi$ are representational processes under I, then I associates a single referent $[\![\phi]\!]_I$ with each $\phi \in \Phi$, a single referent $[\![\theta]\!]_I$ with each $\theta \in \Theta$, a single referent $[\![\gamma]\!]_I$ with each $\gamma \in \Gamma$, and a single referent $[\![\psi]\!]_I$ with each $\psi \in \Psi$. If $P = Q \circ R$, then according to Definition 4.3, $P\ (\Omega, A, D) : \Phi \mapsto \Psi$. The domain Φ of P is, by definition, identical to the domain Φ of Q. The range Ψ of P is, by definition, identical to the range Ψ of R. This means that we can be sure that each element of P's domain Φ, and range Ψ, represents exactly one thing under I. Therefore, condition (1) of Definition 4.1 is satisfied. To show that condition (2) of Definition 4.1 is satisfied, we need to show that if $\phi \vdash_P \psi$ and $\phi' \vdash_P \psi'$ and $[\![\phi']\!]_I = [\![\phi]\!]_I$, then $[\![\psi']\!]_I = [\![\psi]\!]_I$. This can be shown as follows. If $\phi \vdash_P \psi$ then, by Definition 4.3, $\phi \vdash_Q \theta$ and $\theta \vdash_R \psi$. Similarly, if $\phi' \vdash_P \psi'$, then $\phi' \vdash_Q \theta'$ and $\theta' \vdash_R \psi'$. Since Q is a representational process, if $[\![\phi']\!]_I = [\![\phi]\!]_I$, then $[\![\theta']\!]_I = [\![\theta]\!]_I$. Since R is a representational process, if $[\![\theta']\!]_I = [\![\theta]\!]_I$, then $[\![\psi']\!]_I = [\![\psi]\!]_I$. Therefore, if $[\![\phi']\!]_I = [\![\phi]\!]_I$ then $[\![\psi']\!]_I = [\![\psi]\!]_I$. Hence, P fulfills condition (2) of Definition 4.1.

Result 4.1 underwrites part of the explanatory strategy discussed in Section 3.1.1. The problem we faced was to explain how complex semantically-sensitive cognitive processes are possible. Part of the solution was to treat such processes as being made up from smaller processes whose I/O behaviour and semantic sensitivity are easy to explain. Result 4.1 establishes the link between the easy-to-explain semantic sensitivity of the subprocesses and the potentially mysterious semantic sensitivity of the larger process. If the spanning subprocesses of a process are representational, and then the larger process is *ipso facto* semantically sensitive—there should be no mystery about it. In other words, being a composite of simple representational processes is a straightforward way in which a representational process is possible.

There are other ways in which representational processes are possible. A representational process could be a spanning sequence of *non-representational* subprocesses. However, I wish to claim that such processes are not thereby computational. A process is computational only if it is spanned by a sequence of *representational* subprocesses. It is compatible with this requirement

that a computational process be spanned by both representational and non-representational subprocesses. However, there must be *at least one* sequence of representational processes that span a process in order for that process to qualify as computational. Let us consider why.[2]

4.2.2 The representation requirement

There are two questions one might ask. First, why do computational processes need to involve representation at all? I gave three arguments to answer this question in Section 3.1.2. Second, why must all of the *subprocesses* of a computational process be representational? This section provides an answer.

Computational identity of subprocesses

Systems with different physical properties can perform the same computation. For example, a system made out of silicon and a system made out of tin-cans and string can perform the same computation. How can we make sense of this? Anticipating the discussion of Section 5.2.2, we can say that two systems perform the same computation just in case they are spanned by sequences of representational processes that are I/O equivalent. Two representational processes are I/O equivalent just in case their respective I/O pairs represent the same things. Therefore, the notion of computational identity requires that the *subprocesses* of a computation be representational.

The problem for the non-representationalist is that it is unclear how to make sense of computational identity any other way. Let us return to intuition (ii) of Section 3.1.2. This intuition was that two computations are the same just in case they have the same parts and those parts are connected in the same ways. The parts of a computation are its component subprocesses. Therefore, two computations are the same just in case they have the same subprocesses and those subprocesses are connected in the same ways. I wish to focus on what is meant by subprocesses being 'the same'. I shall argue that in order to make sense of this notion, we must assume that those subprocesses are representational. Hence, the subprocesses of a computational processes must be representational. One might say: there is no computation without representational subprocesses.

[2]Here is an example of a representational process that is not spanned by any representational subprocesses. Suppose that a process P takes pictures as input and yields pictures as output. Suppose that process P preserves representational content by cutting up its input pictures into little pieces, moving the pieces around, and glueing them back together to form the same output picture. Suppose that the intermediate pieces are so small that they do not represent anything. Process P is representational, it satisfies Definition 4.1, but it is not spanned by any representational subprocesses: none of its intermediate tokens represent. I claim that such processes, although possible, are not computational.

First, let us consider non-representationalist accounts of what it means for two subprocesses to be the same.

Shared first-order physical properties

One suggestion is that two subprocesses are the same just in case they share some, or all, of their first-order physical properties. This theory can be quickly dismissed. Identity of first-order physical properties is neither necessary nor sufficient for computational identity of subprocesses.

First, computationally identical systems need not share any first-order physical properties. They can be made out of entirely different things: one system may be made out of silicon, the other out of tin-cans and string. Therefore, shared first-order physical properties is not necessary for computational identity.

Second, two systems can share all their physical properties and yet be computationally distinct. The same physical system could, in one context, be interpreted as making predictions about the stock market, and in another context, be interpreted as making chess moves. Therefore, shared physical properties are not sufficient for computational identity either.

Shared pattern of relations between physical properties

Another suggestion is that two subprocesses are computationally identical just in case they share the same *pattern of relations* between their first-order physical properties. On this view, two subprocesses are computationally identical just in case their first-order physical properties are isomorphic in some suitable sense. For example, suppose that a subprocess E maps an input voltage $0\,\mathrm{V} \leq v \leq 5\,\mathrm{V}$ to an output voltage v^2, and a subprocess W maps an input water pressure $0\,\mathrm{Pa} \leq p \leq 5\,\mathrm{Pa}$ to an output water pressure p^2. Subprocesses E and W involve different properties and individuals—electrons in one case, and water in the other. However, the pattern of relations between those properties and individuals is the same: both subprocesses relate the magnitude of their input to their output in a quadratic fashion. Therefore, the two subprocesses E and W count as computationally identical.[3]

However, isomorphism between first-order physical properties is not sufficient for computational identity. As we saw in Section 3.1.2, such isomorphism cannot distinguish between a subprocess that implements an AND gate and a

[3]This means that E and W can count as computationally equivalent parts in different overall systems (for example, one that operates with electricity, and another that operates with water). Note that the above argument, although directed against physical isomorphism as an account of identity of computational *sub*processes, works equally well against physical isomorphism as an account of identity of overall computational systems.

subprocess that implements an OR gate. The subprocess described in Table 3.2 is equally physically isomorphic to both gates.

Physical isomorphism is not necessary for computational identity either. A striking feature of computational systems is that they can, in principle, use any system of coding one likes. For example, one might stipulate that an electrical signal of 0 V represents *0*, and a signal of 5 V represents *1*. Or, one might stipulate that 0 V represents *0*, 1 V represents *1*, 2 V represents *2*, and so on. Or, one might stipulate that 0 V represents *5*, 1 V represents *2*, 2 V represents *4*, and 3 V represents *1*. Or, one might stipulate *any system of coding one likes*. The underlying representational relations are easy to set up: all one has to do is adopt the appropriate conventions. A representational relationship between my left sock and Tony Blair can be set up simply by adopting the convention that my left sock represents Tony Blair. Similarly, all that it is required for a system to compute using any of the coding systems above is for us to adopt the convention that the system's tokens represent what the coding system says they represent. For any voltage v, that voltage can be associated with any value $f(v)$ one pleases.[4,5]

Given this flexibility, one can show that subprocesses that are not physically isomorphic can be computationally identical. Consider two subprocesses A and B that are not physically isomorphic. For example, suppose that process A takes signals of voltage $0\,\text{V} \leq v \leq 10\,\text{V}$ as input and yields signals of voltage v as output, and process B takes water pressure $0\,\text{Pa} \leq p \leq 10\,\text{Pa}$ as input and yields water pressure $p^2 + 100$ as output. The two subprocesses relate their first-order properties in different ways: subprocess A relates the magnitude of its input to output linearly, subprocess B relates the magnitude of its input and output quadratically. (A different example can be chosen if one's concept of isomorphism differs).

Let S be a system of coding such that a voltage v represents the number v if $0\,\text{V} \leq v \leq 10\,\text{V}$, and water pressure p represents the number p if $0\,\text{Pa} \leq p \leq 10\,\text{Pa}$ and $\sqrt{p - 100}$ if $100\,\text{Pa} \leq p \leq 200\,\text{Pa}$. Suppose that we adopt the representational convention that the tokens of subprocesses A and B represent according to S. It now looks like subprocesses A and B perform the same computation: they both compute the function $f(x) = x$. The two subprocesses, under this system of representation, play the same computational role: they are computationally the same in the sense described above. Hence, the two

[4]Within limits: a coding must be consistent, and it can assign at most one referent to each token.

[5]Note that a system of coding and an interpretation function are different. A system of coding is a *stipulation*, for each token, of one thing that token represents. An interpretation function is a *selection*, for each token, of one thing from among the things that token represents. A system of coding sets up new representational relationships, an interpretation function selects already existing representational relationships.

subprocesses are computationally identical. Therefore, physical isomorphism of subprocesses is not a necessary condition for computational identity.[6]

Shared representational content

Shared physical properties and isomorphism of physical properties are poor guides to computational identity. The correct guide to computational identity is representational content. Representational content allows us to make sense of two physically distinct systems performing the same computation. It also allows us to make sense of two physically non-isomorphic systems performing the same computation.

The intuition above was that two computations are the same just in case they have the same subprocesses and those subprocesses are connected in the same ways. The question was how to make sense of the notion of 'the same' subprocess. The first suggestion was that 'the same' means a subprocess with shared first-order physical properties. The second suggestion was that 'the same' means a subprocess with a shared pattern of relations between physical properties. These suggestions were shown to be flawed. The current suggestion is that 'the same' subprocess means a subprocess that maps the same representational content to the same representational content. Two subprocesses are 'the same' just in case they perform the same mapping between representational content. This notion of sameness will later be called 'I/O equivalence' and is defined in Section 5.2.2. For the moment, note that computational identity presupposes representational content at the subprocess level.

The definition of a representational process, Definition 4.1, has two requirements: that a representational process map representations to representations, and that the mapping be representation-preserving. I have argued that the subprocesses of a computation map representations to representations. It is not hard to show that those subprocesses must also be representation-preserving. For the sake of brevity, the argument is given in Appendix A. Granted that this argument is correct, we can conclude that the subprocesses of a computation must be representational.

4.2.3 Simple processes

Definition 4.2 requires that the spanning subprocesses of a computation be simple. Subprocesses often regarded as simple include: logic gates such as AND and OR gates, processes that detect if two representation tokens are the

[6]Note that a restriction to properties that are 'natural', as defined by Lewis (1983), is of no help. Even among natural properties, isomorphism is neither necessary nor sufficient for computational identity. To take one example, *being 0 V* and *being 5 V* (at least plausibly) are natural properties. Therefore, this view still does not secure facts about whether a system is an AND or an OR gate.

same or different, processes that output the first/last elements of a sequence of representation tokens, processes that add a representation token to the start/end of a sequence of representation tokens, processes that generate the successor of a numerical representation, and processes that generate the sum of two numerical representations. Although it is easy to give examples of computationally simple processes, it is not easy to say what their computational simplicity consists in.

A Turing-style definition

Intuitively, simple processes are those that we wish to be the building blocks of computations. What determines whether a given process is computationally simple? One might consider Turing's definition of computation. For Turing, the building blocks of computation are processes that do not require any insight or ingenuity on the part of the human being executing them. So one might define a simple process as a process that, if executed by a human being, would not require any insight or ingenuity.

Unfortunately, this definition is not appropriate for our purposes. First, there are processes that humans beings can execute without insight or ingenuity but which, from the point of view of the CTM, we do not wish to classify as computationally simple. Human beings can recognise three-dimensional objects, play musical tunes, and parse the syntax of their native language without insight or ingenuity. These processes are at least as free from insight and ingenuity as the simple operations described above. However, in each case—objection recognition, music playing, and syntax parsing—we do not wish to claim, in the context of the CTM, that the relevant process is computationally simple. Considerable effort is spent in the CTM in showing that these processes are made up from smaller processes. Therefore, Turing's criterion is not a sufficient condition for a process to count as computationally simple in this context.

Turing's condition is not a necessary condition in this context either. There are processes that are treated as computationally simple by the CTM but which humans, if they can perform them at all, cannot do so without insight or ingenuity. Consider a process with the same I/O behaviour as a Purkinje cell. A Purkinje cell may have as many as $200,000$ inputs and extremely complex I/O behaviour. It seems unlikely that a human clerk could replicate its action as a single operation without using insight or ingenuity. Yet, from the point of view of the CTM, we wish to treat such processes as computational simples. Several theories of cerebellar motor function treat Purkinje cells as computational simples (Albus, 1971; Houk et al., 1996; Ito, 1984; Marr, 1969; Pellionisz et al., 1977).

A single-step definition

An alternative definition of computational simplicity is that simple processes are those that can be executed in a single step. This definition differs from the insight-and-ingenuity proposal: it is conceivable that a process can take multiple steps and be free from insight and ingenuity, and it is conceivable that a process can involve insight and ingenuity but only take a single step. However, the single-step intuition fits with the way in which we understand many electronic computers. Simple processes are those that cannot be broken down into a further sequence of instructions, e.g. those operations that are atomic.

The main problem with the single-step definition of computational simplicity is that it presupposes that we have an independent grasp of the notion of a single computational step, and it is not clear how we could have this. Appeal to single computational steps only constitutes progress if that notion has more content than just its link to the notion of computational simplicity. There appears to be two options: either define the notion of a single computational step in terms of independent notions, or argue that the notion of a single computational step should be left as an undefined primitive. Let us see why neither option is acceptable.

First, consider how one might define the notion of a single computational step in independent terms. One apparently plausible strategy is to define a single computational step in terms of clock-cycles. Many electronic computers have a clock and an associated notion of a clock-cycle. In these 'synchronous' architectures, the length of a clock-cycle determines the time taken for the shortest unit of computation. Therefore, it seems plausible to define a single computational step as a computational operation that lasts for exactly one clock-cycle.

There are three problems with this definition. First, not every computation has a clock. Synchronous architectures are common, but there is no reason why a computer should have a clock or an associated notion of clock-cycle.[7] For computations that lack a clock, the definition above leaves us with no clue as to how to understand single computational steps, and hence computational simplicity. Second, some single-step operations take longer than others. One single-step operation (e.g. NOT) may take one clock-cycle, while another single-step operation (e.g. MOVE) may take two clock-cycles. Therefore, processes that are single-step may last for longer than one clock-cycle.[8] Third, even setting aside previous worries, the principle of linking computational simplicity

[7] For more on this point, see Section 5.1.1.

[8] Hennessy and Patterson (1998), 438–439.

with single clock-cycles processes seems dubious. Such a move prevents high-level operations being 'black-boxed' and treated as new computational simples. Black-boxing seems a key part of our computation talk. There appears to be no reason to privilege single clock-cycle instructions as computationally simple over black-boxed candidates.

If defining single steps in terms of clock-cycles fails, how should one then define that notion? I cannot think of any plausible alternatives. The only notion that seems adequate to define the notion of a single computational step is the notion of a computationally simple process, and we have already seen that that is unacceptable. Can the notion of a single step be left as an undefined primitive? Unfortunately, it cannot. The notion of a single step raises too many questions to be left as an undefined primitive. Single computational steps are just too curious to leave questions about their nature unanswered.

One aspect of single computational steps that needs to explained is that the number of steps that a process takes seems to be, to some extent, in the eye of the beholder. A process that counts as single-step in one context can count as multiple-step in another. Imagine a computational process that adds numbers and which is made out of many smaller computational processes. As the context is currently described, the addition process is multiple-step—it is made out of smaller steps. However, in the context of a larger computation, this addition process may be black-boxed and treated as a single computational step, a single addition operation. Our intuitions seems to allow for this to be possible. Indeed, most addition processes inside real-world computers *are* made up of smaller steps, even though they are often regarded as paradigms of single-step processes.

This change in number of steps seems to be interest-relative. While for a low-level hardware designer, the addition process would count as multiple-step, for a high-level programmer the same addition process would count as single-step. It does not seem that either the hardware engineer or the programmer has made a mistake in this respect. Rather, there does not seem to be a mind-independent fact of the matter to capture. The number of computational steps that a process takes depends on the interests of observers.

An alternative, non-interest-relative, theory of single computational steps is hard to believe. Suppose that there really is an absolute answer to whether an addition process is single or multiple step. What possible metaphysical facts could underlie this answer? It cannot be facts about clock-cycles, since we have already rejected that analysis. Any facts that distinguish single steps from multiple steps have to be mind-independent. But it is hard to think of *any* mind-independent difference between single-step processes and their black-boxed counterparts. Short of positing bizarre *sui generis* facts that distinguish a

privileged basic level computation, I cannot see of any way to do it.

For the moment, let us accept that what counts as a single step is, to some extent, in the eye of the beholder. If this is correct, then it is not plausible to take the notion of a single computational step as an unanalysed primitive. The problem here is not so much that the notion of a single step allows interest-relativity, but that it would leave that interest-relativity unexplained. Taking the notion of a single step as an unanalysed primitive gives no clue as to what makes a process single-step and how our interests come into that decision. Indeed, the mention of interest-relativity may give the unjustified impression that the decision is entirely subjective. Therefore, a minimal requirement on a definition of single computational steps is that it give some account of *how* our interests affect whether a process is single-step. Otherwise, the nature of single steps just seems either entirely subjective or plain mysterious. The interest-relativity of single steps needs an explanation, and the approach above is no help at providing it.

To summarise, the advocate of a single-step definition of computationally simple processes faces a dilemma. Either she can explain the notion of a single step in terms of other notions, or she can treat that notion as an undefined primitive. If she chooses the former option, then she faces the difficulty of finding notions in which to explain the notion of a single step. If she chooses the latter option, then she is open to the charge that she leaves central features of single steps, such as interest-relativity, unexplained.

An explanation-based definition

I have argued that computationally simple processes should not be defined in terms of either insight and ingenuity or single computational steps. Instead, I suggest the following definition:

Definition 4.5. A representational process p is simple just in case its mechanism does not call for a computational explanation.

We saw in Section 3.1.1 that certain processes have I/O patterns that are so simple that they do not call for computational explanation. According to the definition above, such processes are computational simples. If a process *does* calls for computational explanation—if it is not clear how it works—then that process is not a computational simple. A process is a computational simple just in case it is not mysterious how it could achieve its I/O pattern. Processes that are not computational simples raise the kinds of questions discussed in Section 3.1.1 about how such processes are possible and how they work.

Let us discuss the features of the definition point by point.

First, the definition is vague, since it may not be clear-cut whether a particular process calls for computational explanation or not. However, as Lewis (1973) points out, a vague definition is acceptable provided both *definiens* and *definiendum* are vague in the same respects. I believe that this true of our concepts of computational explanation and computational simplicity. Imagine that in a certain computation an addition operation is always followed by a complementation operation. It is hard to say whether this process counts as a simple process or as two distinct processes. Our intuitions about simplicity do not provide a clear answer. Similarly, whether the mechanism of the joint process calls for computational explanation is also unclear: maybe the joint process has sufficiently complex I/O behaviour to call for computational explanation, or maybe not. Without more information it is difficult to say. However, if in a certain context we decide that the joint addition–complementation operation does *not* need a computational explanation, then it seems reasonable to treat the joint process as a computational simple. Similarly, if we decide that the joint process *does* need a computational explanation, then that means we believe it to be made up from smaller computational simples, and so cannot consistently classify it as a computational simple. Correspondingly if, for whatever reason, we decide that the joint process *does* count as a simple, then it seems inconsistent to insist that a computational explanation of it is required. Similarly, if we decide that the joint process does *not* count as a simple, then its mechanism will require an explanation in terms of simpler processes. Therefore, it seems that once the question of whether a process is computationally simple has been settled, then the question of whether that process needs computational explanation has also been settled, and vice versa. The boundaries of both vague concepts align.

Second, the definition above can accommodate the intuition that made the single-step definition so attractive. The single-step definition defined computational simplicity in terms of single computational steps. This equivalence can be retained in Definition 4.5 by reversing the direction of the single-step definition. Instead of defining computational simplicity in terms of single steps, we can define single steps in terms of computational simplicity. This preserves the intuition that a computationally simple process is one that is executed in a single step. Since we have an independent account of computational simplicity, this avoids the criticism above that such a definition would be circular.

Third, Definition 4.5 defines simple processes in terms of specifically computational explanations because it is conceivable that there could be calls for other types of explanations of the mechanism of computational simples. For example, there might be a call to explain why the mechanism of a particular process is implemented in silicon rather than germanium. The explanation might be cost, or availability of materials—nothing that would require one to

consider the process as computationally non-simple. The mechanism of a process can raise many questions, but provided those questions do not involve a call for a specifically computational explanation, then that process is simple.

Fourth, Definition 4.5 makes computational simplicity depend, to some extent, on our interests. It is dependent on our interests to the extent that our explanatory standards are dependent on our interests. This does not mean that computational simplicity is a matter of arbitrary choice. Our explanatory standards cannot be arbitrarily adjusted in a purely subjective way. Definition 4.5 does not entail that 'anything goes', or that what counts as a simple process is insensitive to that process's mind-independent nature. Computational simplicity depends on some combination of the intrinsic nature of the process, the context in which the process takes place, and our explanatory interests and goals. The details of this mix depend on the correct theory of explanation.[9]

Fifth, Definition 4.5 does better than the previous definitions because it can explain how and why the simplicity of a process is interest-relative. According to Definition 4.5, the simplicity of a process is interest-relative because our explanatory standards are interest-relative. We had a specialised notion—computational simplicity—that we found was interest-relative. The current definition explains this interest-relativity in terms of a more general form of interest-relativity: the interest-relativity of explanation. This phenomenon may itself not be understood, but it at least has the virtue of being wide-spread: the interest-relativity of explanation is not specific to computation. Though not a solution, this at least is a promising start in explaining the interest-relativity of computational simplicity.

Sixth, black-boxing was mentioned as a procedure by which processes can be grouped into new computational simples. An attractive feature of Definition 4.5 is that it explains what black-boxing is, and when black-boxing is and is not legitimate. If one accepts Definition 4.5, then black-boxing can be understood as a switch in our explanatory standards. Black-boxing involves a switch from treating a group of processes as in need of a computational explanation to treating that group as a single process whose workings need no computational explanation. Black-boxing is legitimate just in case it is legitimate to apply the associated explanatory standards in that particular situation. This fits with many of our intuitions about black-boxing. A group of processes that is black-boxed and treated as a computational simple in one explanatory context may be broken down and considered as made up from components in another. If one decides, in a particular context, to black-box a process, then

[9] In Section 6.1.3, I argue that the apparent mind-dependence of computational simplicity does not entail that computation must also be mind-dependent. There are a variety of ways in which a realist about computation can defend her position in this regard.

effectively one denies that a computational explanation of that process is owed in that context. Similarly, if one sets one's explanatory standards such that a certain computational process calls for explanation, then it would be untrue to those explanatory standards to then black-box that process and treat it as a computational simple.

Seventh, Definition 4.5 fits with the account given above of the CTM. As described in Section 3.1.1, the purpose of the CTM is to provide an explanation of how complex semantically sensitive cognitive processes could work. One way of explaining how a cognitive process could work is to show that it is made up from other processes. For this to be a good explanation however, those other processes must not themselves be in need of explanation. These are the computational simples. Therefore, just as Definition 4.5 entails, our explanatory standards must coincide with our standards of computational simplicity. A second point is that if our explanatory standards change, then the processes that count as simple change too. This is also reflected in the practice of cognitive science. In some areas, the cognitive processes of attention, memory, and syntax parsing are black-boxed and used to explain other processes. In other areas, those processes themselves come under scrutiny and are explained in terms of other computational simples. As Definition 4.5 claims, cognitive scientists treat different processes as computationally simple in different explanatory contexts.

Three objections

Definition 4.5 appears to be open to three objections.

Objection 1. *Definition 4.5 uses the notion of computation on both sides. In Definition 4.2, computational processes were defined in terms of simple processes. Now in Definition 4.5, simple processes are defined in terms of 'call for computational explanation', and hence in terms of computation.*

The response to this objection is that 'call for computational explanation' is not meant to be understood in a way that presupposes the notion of computation. What is meant by 'call for computational explanation' has already been indicated. A mechanism of a process calls for computational explanation if that mechanism is mysterious, if questions arise about how that process is possible, how it works, or how it could be sensitive to the representational contents of its tokens. The notion 'call for computational explanation' is intended to be understood as independent of, and prior to, the notion of computation.

Objection 2. *Definition 4.5 presupposes simplicity rather than defining it. Definition 4.5 defines computational simplicity in terms of need for explanation. However, the*

notion of 'need for explanation' arguably presupposes the notion of 'not-simple', which itself presupposes the notion of 'simple', which is the notion that we are trying to define.

This objection would be correct if Definition 4.5 attempted to define a general notion of explanatory simplicity. However, this is not the case. The notion that Definition 4.5 attempts to define is a specialised type of simplicity: the notion of computational simplicity. Even if the notion of 'need for explanation' presupposes our general notion of simplicity, that does not mean that it presupposes this specialised notion. It is worth noting that the term 'computational simplicity' in the context of this specialised notion is largely a term of art. It is not clear what connections the notion of computational simplicity has to our general notion of simplicity.

Objection 3. *Reduction to simple components is only one form of explanation, but Definition 4.5 presupposes that it is the only form of explanation.*

This objection has already be dealt with by the third point discussed above. Definition 4.5 does concern computational explanation, but it does not require that this is the only kind of explanation that can be given. Reduction to simples is not the only form of explanation, but it is the only form of explanation relevant when deciding whether a process is computationally simple.

Turing-equivalence of simple processes

A representational process is Turing-equivalent just in case it is I/O equivalent to some Turing machine. An example of a process that is not Turing-equivalent is a process that can correctly predict whether an arbitrary Turing machine halts or not on arbitrary input (*viz.* that solves the halting problem). Let us call processes that are not Turing-equivalent 'incomputable'. Can incomputable processes qualify as computational simples? An immediate reaction is that they should not. It is generally regarded as a minimal condition on computation that a process should be computable and that each of its steps should be computable.

A potential objection to Definition 4.5 is that it does not obviously entail this condition. Definition 4.5 defines computational simplicity in terms of the need for computational explanation. The need for computational explanation is sensitive to our explanatory interests and those interests can vary. There appears to be no guarantee that variation in our explanatory interests will always respect the boundary between Turing-equivalent and incomputable processes. In other words, it is conceivable that there could be an incomputable process which, for some reason, *we feel no need to explain*. According to Definition 4.5, such a process would qualify as a computational simple.

There is a strong temptation to legislate against such cases. However, we should not be too hasty. Sometimes we might wish to relax the limits on computability described by Turing and consider computation in a broader sense. For example, consider Turing's own discussion of oracle machines. Turing defines an oracle machine as a Turing machine with an extra atomic operation *O*, which solves the halting problem.[10] Even though operation *O* is not Turing-computable, Turing still asks what kinds of computation (in a broader sense) can be performed by such a machine. More recently, some philosophers have claimed that there could be physical systems which intuitively seem to be performing computations and yet fail to satisfy Turing-equivalence.[11] The possibility of such cases appears to show that our intuitions about computation apply to a wider class of phenomena than Turing-equivalence alone would suggest.

These cases are handled nicely by Definition 4.5. Definition 4.5 sets the requirements for the simple processes of a computation to be explanatory rather than Turing-equivalence. This explains how oracle machines can be introduced. When an oracle machine is defined it is stipulated that operation *O* is possible without enquiring into its workings. When one talks about the 'computations' that oracle machines perform, one stifles any desire to know how process *O* works: one black-boxes process *O*. In this respect, process *O* is like any other computational simple. Similarly, we may choose to black-box certain physical processes and treat them as computational simples. As described above, it is conceivable that this could happen even if those physical processes are not Turing-equivalent.

Definition 4.5 gives an attractively unified treatment to Turing-equivalent and non-Turing-equivalent computation. If one wishes to restrict attention to Turing-equivalent cases, then Turing-equivalence can be added as an explicit requirement to Definition 4.5. However, Definition 4.5 as it currently stands shows how Turing-equivalent and non-Turing-equivalent computation fall under the same general intuitions, and it explains how the two notions are continuous. Definition 4.5 also explains why we even *have* intuitions about computation in a broader sense at all. Contrary to our initial reaction, it is a positive virtue of Definition 4.5 that the requirement of Turing-equivalence is not entailed.

[10] See Turing (1939). See also discussion in Copeland (1998).

[11] For example, see Cotogno (2003); Hogarth (1994); Ord (2002); Welch (2004).

Stateful processes

A stateful process is a process whose output depends, not just on its current input, but also on its previous input. In other words, a stateful process is a process with a memory. Stateful processes are common in electronic computers. Such processes have not yet been considered. Processes, as they are defined in Definition 3.2, are non-stateful: they are sensitive only to their current input. However, there are a number of ways in which stateful processes can be accommodated.

First, one might try to reduce any stateful process to a spanning sequence of non-stateful processes. This strategy should be familiar from other processes with mysterious I/O behaviour: treat a process with mysterious I/O behaviour as made up from components with non-mysterious I/O behaviour. Many real-world stateful processes are made up from entirely non-stateful subprocesses. It is surprisingly easy to create such processes. Examples of how this is done are given in Section 5.1.6.

However, as a general strategy, a reductionist approach to stateful processes is not attractive. First, it is not clear whether it is possible to reduce *every* stateful process to non-stateful subprocesses. Second, on such an approach, a stateful process cannot itself qualify as a computational simple, and this seems to clash with our intuitions. In many cases we *do* wish to treat a stateful process as a computational simple. We black-box stateful processes, resist asking questions about their workings, and treat them as simple components when building larger computations. It seems wrong to insist that simple processes must always be non-stateful.

Fortunately, there is a more attractive way in which stateful processes can be accommodated: Definition 3.2 can be modified to allow processes to be stateful. Instead of associating a set of triples of I/O pairs and time delays with each process, we can associate a set of inputs crossed with all possible past inputs. Therefore, instead of the set Ω consisting of triples (ϕ, ψ, t), it should instead consist of tuples $(\phi_1, \ldots, \phi_n, \psi, t)$ where $\phi_1, \ldots \phi_{n-1}$ are the previous inputs to the process, i.e. the inputs prior to ϕ_n. For the sake of simplicity in what follows, we shall stick to the original, non-stateful, definition of a process. However, extending the definitions to cover the stateful cases is not hard.

4.2.4 Structured representation

Definition 4.2 requires that the inputs to each subprocess of a computation be structured. The notion of 'structured representation' is intended to define the notion of 'formal' or 'syntactic' structure introduced in Section 3.1.1. In Section 3.1.1, it was described how a key requirement on computation is that the rel-

evant processes be sensitive to the formal structure of their input. Examples of inputs with formal structure include: a sequence of electrical pulses, a sequence of ink-marks (e.g. '01101110'), and an input made out of interlocking parts (e.g. a Lego model). This section aims to give a general account of what it means for a representation to have formal structure.

Representation types

Before defining the notion of a structured representation, we need to define the notion of a representation type. So far, by 'representation' we have meant a representation token. For some purposes however, it is useful to consider groups of representation tokens. Representation tokens can be grouped together in many ways. One important way of grouping representation tokens is by the property of representing the same thing. Let us define the 'representation type' of a token ϕ on a set Φ as the set of tokens in Φ that represent the same thing as ϕ.

Definition 4.6. The representation type of a token ϕ on a set Φ under interpretation function I is $\{\phi' \in \Phi : [\![\phi]\!]_I = [\![\phi']\!]_I\}$.

Two points should be made. First, representation types are not *sui generis* features of the PR-model. Reference to representation types can be eliminated, if one so wishes, in favour of conditions involving representation tokens. Second, by Definition 3.1, each $\phi \in \Phi$ represents exactly one thing under I. Therefore, the set of representation types on Φ under I are equivalence classes on Φ. In other words, those representation types divide Φ up without overlap or remainder.

Structured representation

A *structured representation* is a representation that is a member of an alphabet of a finite number of basic representation types, or a composition of members of those basic types. Examples of composition operators are: spatial composition, temporal composition, and chemical bonding (e.g. the bonding between nucleotides in DNA). A computation is *sensitive* to the formal structure of an input representation just in case the output of that computation depends only on the basic types that make up that input and the way in which those types have been composed together. A representational process *operates on structured representations* just in case its inputs are structured representations and it is sensitive to their formal structure. This can be formalised as follows:

Definition 4.7. A representational process $P : \Phi \mapsto \Psi$ under I operates on structured representations just in case there is a basic alphabet set Θ, and a finite number of composition operators, $\diamond_1, \ldots, \diamond_m$, such that $\forall \phi \in \Phi$, either:

- ϕ is a composite, or
- $\phi \in \Theta$.

Definition 4.8. ϕ is a composite iff $\phi = x_1 \diamond x_2$, where $\diamond \in \{\diamond_1, \ldots, \diamond_m\}$ associated with the process, $\diamond$ is a valid composition operator on x_1 and x_2, and for $i \in \{1, 2\}$, either:

- $x_i \in \Theta$, or
- x_i is a composite.

Definition 4.9. $\diamond$ is valid composition operator on x_1 and x_2 under I just in case if $[\![x_1]\!]_I = [\![y_1]\!]_I = \alpha$, $[\![x_2]\!]_I = [\![y_2]\!]_I = \beta$, and $[\![x_1 \diamond x_2]\!]_I = \gamma$, then $[\![y_1 \diamond y_2]\!]_I = \gamma$.

Let us discuss the features of the definition point by point.

First, Definition 4.7 allows a representation token to be made up from multiple representations and multiple composition operations, e.g. $\phi = (x_1 \diamond_1 (x_2 \diamond_2 x_3)) \diamond_3 x_4$. In many cases, the number of basic representation types and the number of different composition operators is small, but in principle, their numbers could be as large as one likes, so long they are finite. The possibility of nesting composition operators provides a reply to a possible objection. The objection is that there might be composition operators that operate on more than two representations at a time. For example, a composition operator might compose three representation tokens at a time to form a fourth token. Such cases are not covered by the definition above, which only consider binary composition operators. However, Definition 4.7 can accommodate n-ary composition operators by treating such composition operations as a sequence of binary compositions. The procedure is similar to that of treating an n-ary function $f(x_1, \ldots, x_n)$ as a nested sequence of binary functions: $f_1(x_1, f_2(x_2, f_3(x_3, \ldots f_{n-1}(x_{n-1}, x_n)))))$.

Second, the alphabet set Θ need not be part of the domain Φ of the process. To see why this condition is justified, consider the following case of operation on structured representation. Consider a process that operates on a temporal sequence of eight electrical signals, each of which can be either 0 V or 5 V—a process that operates on bytes. Suppose that the process performs some operation, such as checking whether the eighth signal represents *0* or *1*. The domain Φ of the process is a domain of *sequences* of signals—a domain of *bytes*. However, plausibly the alphabet set Θ consists only of *single* signals of 0 V or 5 V, which are temporally composed together to form a sequence. These basic representations are not valid inputs to the process. A lone 5 V signal would not be a valid input to the process. Only a sequence of eight signals is valid input. Hence, the alphabet set Θ need not be part of the domain Φ. Note that composite representations, such as $x_1 \diamond x_2$, need not be part of the domain Φ

either. A sequence of *two* signals is not a valid input to the process either, even though it is composite. Therefore, not all composite representations or basic representations need be part of the domain Φ.

Third, Definition 4.9 states that the output of a process is *at most* sensitive to the representational content of its input's constituents. If a process yields a value with representational content other than $[\![\psi]\!]_I$ for its output, it must be that either the representational content of x_1 or x_2 in the input token $x_1 \diamond x_2$ has changed, or the input is no longer in the $\diamond$-part of the domain. (Remember that by condition (2) of Definition 4.1, if a process yields a value with representational content other than $[\![\psi]\!]_I$, then it must be that the representational content of its input $[\![x_1 \diamond x_2]\!]_I$ has changed, and hence by Definition 4.9, that either the representational content of x_1 or x_2 have changed, or the input is no longer in the $\diamond$-part of the domain). Therefore, process P is sensitive, on the relevant $\diamond$-part of its domain, only to the representational content of x_1, x_2 and the way in which they are composed. This gives a precise definition of the way in which computations are sensitive to the formal structure of their input.

Definition 4.9 can be understood as an extension of Definition 4.1. Condition (2) of Definition 4.1 states that if two input tokens of a representational process share the same representational content, then they produce output tokens that also share the same representational content. Definition 4.9 applies the same principle to the component parts of representations. Definition 4.9 states that if two input tokens have *components* that share the same representational content, and they are composed in the same ways, then they should map to output tokens that share the same representational content.

Definition 4.9 places an upper bound on how the output of P can vary as the components of its input are changed. The output of P can be *at most* sensitive to the content of its input's components: no variation in representational content of the output is allowed without variation in the representational content of the input's components, or the way in which they are composed. However, Definition 4.9 places no lower bound on how the output of P can change with variations in representational structure. It is consistent with Definition 4.9 that process P completely ignore some components of its input, i.e. that changes in the component representations have *no effect at all* on the representational content of the output. This may seem odd. What is to stop one from positing all kinds of strange structures that have no effect on the output? What is to stop one from introducing as many composition operators as one likes?

There are two reasons why one should not worry about this problem.

First, there is a good reason why Definition 4.9 places no lower bound on how the output of P can vary with changes in representational structure. It turns out to be convenient to say that certain processes *operate on* structured

representations even if those processes do not respond to changes in that structure. This is especially true of simple processes. Even though a simple process may not *use* a certain aspect of the structure of a representation—it may not change its output if changes are made in the components of that input—it is nevertheless useful to say that the process operates on those structures. One might think of such a process as 'passing' such structures, operating on them without responding to all of their features. A liberal approach in this respect allows us to preserve the intuition that a representation can keep its structure throughout different subprocesses in a computation even though not all of them would be sensitive to all aspects of its structure. We can say that a representation can keep its structure throughout a computation, with some subprocesses operating on some parts of its structure and others on others. Even if not all of a computation's subprocesses respond to every feature of a representation's structure, we can still say that those subprocesses operate on a single univocal structure.[12]

Second, it is not true that one is free to introduce as many composition operators as one likes. Although Definition 4.9 does not explicitly forbid the introduction of strange and useless operators, other norms rule them out. In particular, the pragmatics of computation talk rules such operators out. It is a violation of norms of informativeness and relevance to talk about aspects of representational structure that play no role in a computation. One way of phrasing this norm would be to say that a composition operator should not be introduced into a computation if changes in the components of that operator have no effect on the output of *any* of the subprocesses of that computation. This statement of the norm may need to be refined, but the point remains that pragmatic factors rule out the introduction of strange and useless operators. It is not false to posit such operators, but it is inappropriate. The pragmatics of computation talk receives more discussion in Section 5.3.

A final point of clarification is that although the joining operator '$\circ$' of Section 4.2.1 and composition operators '$\diamond$' discussed above both compose things, they compose different things. A joining operator composes processes, a composition operator composes representations. Computations are therefore compositional in at least two respects: they are composed of subprocesses, and their input representations are composed of basic representations.

Examples of composition operators

A computational process can use the same composition operators throughout its computation, or it can use different composition operators at different

[12] A lower bound that could be added to Definition 4.9 without violating these intuitions would be desirable, but I cannot see how to do it.

stages. Commonly used composition operators include: spatial concatenation, temporal concatenation, superposition of waveform, and coinstantiation of properties. Here is a brief description of each.

Two representations are spatially concatenated just in case they are physically adjacent to each other in some specified way, the details of which are specified by the operator in question. For example, the ink-marks 'abc' consists of the individual ink-marks 'a', 'b', and 'c' spatially concatenated. The individual ink-marks represent letters, and a spatial concatenation of ink-marks represents a string of letters. A computational process might operate on such representations. For example, a *head* process might take a spatial concatenation of ink-marks as input and yield an ink-mark that represents the first letter of the input (the head of the string) as output: if a head process is presented with 'abc', it yields 'a' as output. As required by Definition 4.9, such a process is not sensitive to more than the representational content of the components of its input and the way in which those components have been composed together.

Two representations are temporally concatenated just in case they are temporally adjacent to each other in some specified way, the details of which are specified by the operator in question. For example, voltage signals can be temporally concatenated to form a temporal sequence of voltage signals. Such a sequence may be a single input to a computational process. This is true of many processes inside electronic computers. Instead of operating on single voltage signals, these processes operate on temporal sequences of voltages, e.g. bytes.

Two representations are waveform superimposed just in case they are both waves and have been added together by a superposition operation. Waveform superposition requires that the components of the representation be of different frequencies or phases, just as spatial and temporal concatenation require that their components occupy different spatial or temporal locations. Waveform superposition is used in electronic computers to perform computations across telephone wires.

Two representations are composed by coinstantiation of properties just in case a composite instantiates some subset of both their properties, the details of which are specified by the operator in question. For example, suppose that x_1 instantiates the property of having a green dot, and x_2 instantiates the property of having a red dot. A composite $x_1 \diamond x_2$ may instantiate both the property of having a green dot and the property of having a red dot. Suppose that the property of having a red dot represents a value on the x-axis, and the property of having a green dot represents a value on the y-axis. The composite $x_1 \diamond x_2$ could then represent a combination of the representational content of its components (e.g. a location on the xy-plane). A process could be sensitive to this representational content in a straightforward way by responding differently to

the two different properties.

Three consequences should be noted. First, a composite representation need not have its components as proper spatial or temporal parts. In the examples above, this was true of waveform superposition and coinstantiation of properties. Second, a composite may have parts and properties that neither of its components have. These extra parts and properties do not count as internal structure if they do not have an associated representational content. One might think of these extra parts and properties as 'junk'—a non-representational background against which changes in representational structure are noticed. Third, in some cases the input representations of a computation may not have any interesting structure at all. For example, the inputs of a NOT gate are signals of 0 V and 5 V that represent *0* and *1*. There are no composition operators or composite representations, just two basic representation types. However, it is convenient to extend the notion of structured representation to these cases and to call them 'structured'. It is clear that they satisfy Definition 4.7, albeit only the base clause of the definition.

Analogue computation

An analogue representational process is a representational process whose domain or range represents a continuum, such as the real number line.

Definition 4.10. Representational process P (Ω, A, B): $\Phi \mapsto \Psi$ under interpretation function I is an analogue representational process iff either of the sets $\{[\![\phi]\!]_I : \phi \in \Phi\}$ or $\{[\![\psi]\!]_I : \psi \in \Psi\}$ has a subset that is a continuum.

An analogue computational process is a computational process at least one of whose spanning subprocesses is an analogue representational process. Analogue computation allows for calculation with an unlimited degree of accuracy. An analogue adding process can add real numbers exactly, without approximation. However, this ideal is rarely realised in the real world. Few physical processes, and perhaps none, are sensitive to infinitesimal differences in representational content. Usually there are limits to how far individual tokens can reliably be discriminated. This is the main reason why analogue computation is rarely used in electronic computers.

Analogue computation requires that *represented entities* form a continuum; it does not require that the representation tokens form a continuum. These two conditions can come apart in the following ways.

First, a set of tokens can form a continuum even if their representational content does not. Consider a process that takes input voltages that lie in the continuum between 0 V and 5 V. Suppose that signals of magnitude v (0 V $\leq v \leq$ 2.5 V) represent *0*, and signals of magnitude v (2.5 V $< v \leq$ 5 V) represent

1. The set of input tokens forms a continuum. However, the representational content of those tokens does not: it consists of only the two values *0* and *1*.[13]

Second, the set of representational content can form a continuum even if the representation tokens do not. The set of representational content forms a continuum just in case there are *enough* tokens, and the right representation relations obtain. There are enough tokens provided those tokens can be put into a one-to-one correspondence with the members of a continuum.[14] Notably, this condition can be satisfied without the tokens themselves forming a continuum. All that is required is that the set of tokens be uncountable. This condition can be satisfied without the tokens forming a continuum by, for example, the tokens failing to be linearly ordered.

Some cases of analogue computation appear to violate Definition 4.7. According to Definition 4.7, each input token must either be a member of a finite number of representation types, or the result of a finite number of compositions of members of those types. Consider a representational process in which a continuum of input voltage signals represents a continuum of real numbers. This process appears to violate Definition 4.7 because it is not clear how each input could either be a member of a finite number of representation types, or the result of a finite number of compositions of members of those types. Definition 4.7 can be weakened to accommodate these cases. However, this possibility will not be considered. In what follows, we shall ignore such cases of analogue computation and focus on the core cases of computation relevant to the CTM.

4.3 General computational processes

So far we have only considered unary processes: processes that take single representations as input and yield single representations as output. In this section, we shall consider general processes: processes that can take multiple representations as input and yield multiple representations as output. General processes offer the possibility of much more complex computational structures than their unary counterparts. General processes can be used to create much more complex computational architectures (for example, see Figure 4.3). Unlike unary processes, general processes are not restricted to being made up from a linear chain of simple processes.

Definition 3.2, the definition of a process, does not support the possibility of processes with multiple input and output. Therefore, Definition 3.2 needs to be modified before we can continue. Let us consider how we should revise

[13] Many real-world logic gates accept voltages in continuous ranges (e.g. 0 ± 0.4 V and 5 ± 0.4 V), even though such processes are prime examples of non-analogue processes.

[14] Strictly speaking, only a surjection between the tokens and a continuum is required.

Definition 3.2.

General processes part I—from tokens to sequences

Definition 3.2 defined a process as a mapping between single tokens. As it currently stands, Definition 3.2 does not allow for multiple input and output. However, we can modify Definition 3.2 by, instead of defining a process as a mapping between single tokens, defining a process as a mapping between *sequences* of tokens. On this view, a process with n inputs and m outputs is modelled as a mapping from n-sequences of tokens $(\phi_1, \ldots, \phi_n)$ to m-sequences of tokens $(\psi_1, \ldots, \psi_m)$. The individual members of the sequences correspond to the individual inputs and outputs of the process. We can rephrase the counterfactuals underlying the process as follows: if $(\phi_1, \ldots, \phi_n)$ were presented to the process, then $(\psi_1, \ldots, \psi_m)$ would be yielded by the process.

Talk of sequences being 'presented to' or 'yielded by' a process can be reduced to conditions involving their individual members being presented to or yielded by the individual inputs and outputs of that process. Let us say that if an n-sequence $(\phi_1, \ldots, \phi_n)$ is 'presented to' a process, then ϕ_1 is presented to the first input, ϕ_2 to the second input, ..., and ϕ_n to the nth input. Similarly, let us say that if an m-sequence $(\psi_1, \ldots, \psi_m)$ is 'yielded by' a process, then ψ_1 is yielded by the first output, ψ_2 by the second, ..., and ψ_m by the mth output.

Three complications have not yet been considered. First, a general process may receive some of its inputs at different times. Second, a general process may yield some of its outputs at different times. Third, even if a general process

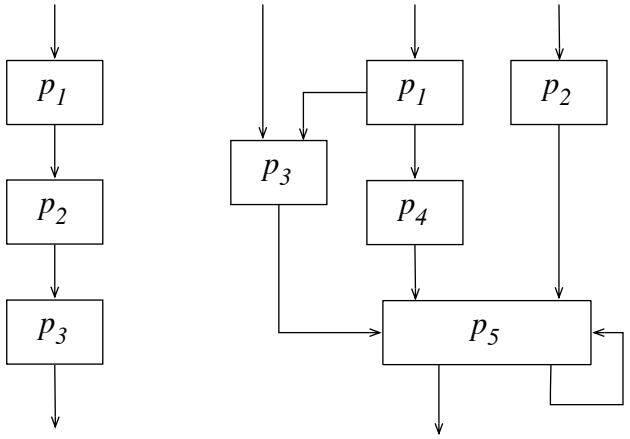

Figure 4.3: Unary vs. general computational processes.

never receives all of its inputs, those that it does receive may still be sufficient for it to yield an output. It takes some care to integrate these features into the model. This integration is done in two stages.

First, we need to introduce a placeholder that is not itself a representation token to represent the absence of an input or output token. This placeholder signifies an empty member in an input or output sequence. It does not matter what entity we choose as a placeholder, so long as that entity is not also needed as a representation token. Assume that the empty set $\emptyset$ satisfies this condition. We shall then say that if the placeholder $\emptyset$ is the ith member of an input or output sequence, then the ith input or output to that process is empty. For example, if we say that $(\phi_1, \emptyset, \phi_3)$ is presented to a process, then we mean that the token ϕ_1 is presented to the first input of the process, nothing is presented to the second input, and the token ϕ_3 is presented to the third input. If we say that $(\psi_1, \psi_2, \emptyset)$ is yielded by a process, then we mean that ψ_1 is yielded by the first output of the process, ψ_2 is yielded by the second output, and nothing is yielded by the third output.

Second, we need to associate a sequence of time delays with each pairing of I/O sequences. These time delays measure the time from the presentation of the input to the yielding of each one of the outputs. If an input $(\phi_1, \ldots, \phi_n)$ is presented to a process, then each output $\psi_i \in (\psi_1, \ldots, \psi_m)$ is yielded time $t_i \in (t_1, \ldots, t_m)$ after the input. These time delays are measured from the onset of the *whole* of the input sequence because it is the whole of the input sequence that is treated as the input to the process. If one wishes to consider cases in which only a partial input is present, then one should consider I/O pairings that contain the placeholder $\emptyset$ in the relevant places. Therefore, the counterfactuals underlying a process can be rewritten as follows: if $(\phi_1, \ldots, \phi_n)$ were presented to a process, then $\psi_i \in (\psi_1, \ldots, \psi_m)$ would be yielded time $t_i \in (t_1, \ldots, t_m)$ after the entire input was presented.

A possible objection to this account is that inputs and outputs with empty members are not inputs or outputs at all. If some of the members of an input are absent, then we do not have a partial input, we have an ill-formed input—in other words, no input at all. If this objection were correct, then the model above could be simplified. However, there are good reasons for thinking that the objection is wrong. Decisions about well-formedness should be based on our pre-existing intuitions about which inputs and outputs are valid inputs and outputs of a computational process. In many cases, we *do* wish to say that a computational process with n inputs performs a computation even if only a proper subset of those inputs is ever used. Legislating otherwise seems plain wrong. Judgements about well-formedness in natural language should be based on pre-existing intuitions of native speakers. Similarly, judgements about

well-formedness in computation should be based on pre-existing intuitions about computation, not just on convenience and simplicity.

General processes part II—presenting input and yielding output

In the case of unary processes, we made sense of the presentation of input and the yielding of output in terms of the presence of tokens in spatiotemporal regions. The same approach can be used for general processes. However, instead of associating a pair of spatiotemporal regions A, B with each process, we can now associate two sequences of spatiotemporal regions $(A_1, \ldots, A_n)$, $(B_1, \ldots, B_m)$ with each process. In this way, each input or output of the process has its own associated spatiotemporal region. Let us say that a token ϕ is 'presented' to the ith input of a process just in case that token occurs in a spatiotemporal A_i associated with that process. Let us say that a token ψ is 'yielded' by the jth input of a process just in case that token occurs in a spatiotemporal B_j associated with that process. As with unary processes, there is no restriction on the shape of these input and output regions: they can be disjoint, identical, or overlap to some degree.

In Definition 3.2, we characterised a process by the ordered triple (Ω, A, B). In the revised definition, we can still characterise a process by an ordered triple, but this time by an ordered triple that involves sequences rather than individuals. In Definition 3.2, the elements in the triple were: (i) the set of I/O pairs and their time delays, (ii) the input region, and (iii) the output region. The elements in the revised triple are: (i) the set of I/O sequences and their sequence of time delays, (ii) the sequence of input regions, and (iii) the sequence of output regions. Let us consider each element in turn.

Definition 3.2 stated that if a certain triple (ϕ, ψ, t) is a member of Ω then, if ϕ were presented to the process, then ψ would be yielded after time t. In the new definition, the set Ω is the set of I/O sequences and their sequence of time delays. If a certain triple $((\phi_1, \ldots, \phi_n), (\psi_1, \ldots, \psi_m), (t_1, \ldots, t_m))$ is a member of Ω, then if $(\phi_1, \ldots, \phi_n)$ were presented to a process, then $\psi_i \in (\psi_1, \ldots, \psi_m)$ would be yielded time $t_i \in (t_1, \ldots, t_m)$ after the entire input was presented. The two other members of the triple are A and B, the input and output regions. In the revised definition, we can replace and A and B with the sequences $(A_1, \ldots, A_n)$ and $(B_1, \ldots, B_m)$. Let $\mathbf{A}$ denote the sequence $(A_1, \ldots, A_n)$. Let $\mathbf{B}$ denote the sequence $(B_1, \ldots, B_m)$. We can now define a general process in terms of the triple $(\Omega, \mathbf{A}, \mathbf{B})$.

A triple $(\Omega, \mathbf{A}, \mathbf{B})$ characterises a process with n inputs and m outputs just in case for each $((\phi_1, \ldots, \phi_n), (\psi_1, \ldots, \psi_m)) \in \Omega$, if the token ϕ_1 were present in A_1, ϕ_2 were present in A_2, …, and ϕ_n were present in A_n, then, no matter what other input tokens are present in the input regions before or after $\phi_1, \ldots, \phi_n$, the

token ψ_1 would be present in B_1 after time t_1, ψ_2 would be present in B_2 after time t_2, ..., and ψ_m would be present in B_m after time t_m.

Definition 4.11. P $(\Omega, \mathbf{A}, \mathbf{B})$ is a process with n inputs and m outputs iff $\forall$ $((\phi_1, \ldots, \phi_n), (\psi_1, \ldots, \psi_m), (t_1, \ldots, t_m)) \in \Omega$: if

ϕ_1 were to occur in region A_1

ϕ_2 were to occur in region A_2

$\vdots$

ϕ_n were to occur in region A_n

then, no matter what other ϕ_i' occur in $A_1, \ldots, A_n$ before or after $\phi_1, \ldots, \phi_n$:

ψ_1 would occur in region B_1 a time t_1 after $(\phi_1, \ldots, \phi_n)$

ψ_2 would occur in region B_2 a time t_2 after $(\phi_1, \ldots, \phi_n)$

$\vdots$

ψ_m would occur in region B_m a time t_m after $(\phi_1, \ldots, \phi_n)$

furthermore, we shall assume that processes are deterministic:

$\forall((\phi_1, \ldots, \phi_n), (\psi_1, \ldots, \psi_m), (t_1, \ldots, t_m)), ((\phi_1, \ldots, \phi_n), (\psi_1', \ldots, \psi_m'), (t_1', \ldots, t_m')) \in \Omega$, $1 \leq i \leq m$, $\psi_i' = \psi_i$ and $t_i' = t_i$.

Note that in order to satisfy this definition a general process must have at least one input and at least one output (i.e. $n, m \geq 1$).

A shorthand for the statement that a process P yields an output $(\psi_1, \ldots, \psi_m)$ if presented with an input $(\phi_1, \ldots, \phi_n)$ is: $(\phi_1, \ldots, \phi_n) \vdash_P (\psi_1, \ldots, \psi_m)$.[15]

If the singleton sequence (α) is not distinguished from the entity α, then Definitions 3.2, 4.1, and 4.2 can be shown to be special cases of their general counterparts.

4.3.1 General representational processes

Now that general processes have been defined, it is relatively straightforward to define general representational processes. Unary representational processes were defined in Definition 4.1. General representational processes are defined in a similar way:

[15] In analogy with the definition above, the $\vdash$-relation is defined as follows: if P $(\Omega, \mathbf{A}, \mathbf{B})$ is a process with n inputs and m outputs then $(\phi_1, \ldots, \phi_n) \vdash_P (\psi_1, \ldots, \psi_m)$ iff $\exists(t_1, \ldots, t_m)$ such that $((\phi_1, \ldots, \phi_n), (\psi_1, \ldots, \psi_m), (t_1, \ldots, t_m)) \in \Omega$.

Definition 4.12. P $(\Omega, \mathbf{A}, \mathbf{B})$: $\Phi_1 \times \ldots \times \Phi_n \mapsto \Psi_1 \times \ldots \times \Psi_m$ is a representational process under interpretation function I iff P is a process and:

1. Each $\phi_i \in \Phi_i$ and $\psi_j \in \Psi_j$ represents at least one thing, and out of those things, interpretation function I picks out exactly one referent $[\![\phi_i]\!]_I$ or $[\![\psi_j]\!]_I$ for each $\phi_i \in \Phi_i$ and $\psi_j \in \Psi_j$ $(1 \leq i \leq n, 1 \leq j \leq m)$.

2. $\forall(\phi_1, \ldots, \phi_n)$ and $(\phi'_1, \ldots, \phi'_n) \in \Phi_1 \times \ldots \times \Phi_n$, if $(\phi_1, \ldots, \phi_n) \vdash_P (\psi_1, \ldots, \psi_m)$ and $(\phi'_1, \ldots, \phi'_n) \vdash_P (\psi'_1, \ldots, \psi'_n)$ and $1 \leq i \leq n$, $[\![\phi'_i]\!]_I = [\![\phi_i]\!]_I$, then $1 \leq j \leq m$, $[\![\psi'_j]\!]_I = [\![\psi_j]\!]_I$.

Definition 4.12, like Definition 4.1, has two conditions. Condition (1) of Definition 4.12 states that the input and output tokens of a representational process have representational content. Condition (2) of Definition 4.12 states that representational processes preserve relations between representational content: if two sequences of input tokens $(\phi_1, \ldots, \phi_n)$ and $(\phi'_1, \ldots, \phi'_n)$ have respective members that represent the same thing, then their two sequences of output tokens must also have respective members that represent the same thing. Intuitively, the representational content of the output of a general representational process cannot vary without some variation in the representational content of at least one of its inputs.

4.3.2 General computational processes

We are now in a position to define general computational processes. Unlike unary computational processes, a general computational process can have multiple inputs, multiple outputs, and be spanned by subprocesses that have multiple inputs and outputs. Nevertheless, the definition of a general computational process is structurally similar to the definition of a unary computational process, Definition 4.2. The difference lies in the way in which the terms 'process' and 'spanning' are interpreted. The definition of a general computational process is as follows:

Definition 4.13. P $(\Omega, \mathbf{A}, \mathbf{B})$ is a computational process under interpretation function I iff there are a finite number of representational processes $p_1, \ldots, p_n$ under I such that:

1. $p_1, \ldots, p_n$ span process P.

2. $p_1, \ldots, p_n$ are simple processes.

3. Each p_i operates on structured representations under I.

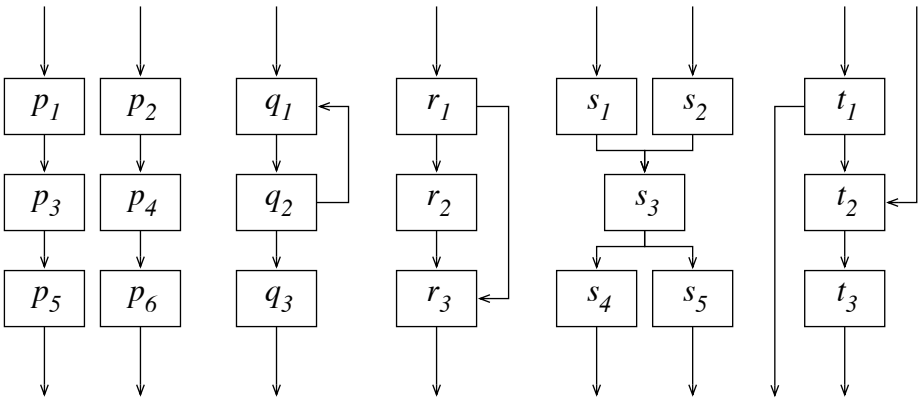

Figure 4.4: A few of the many ways in which a general process can be spanned by subprocesses: (1) the subprocesses can form parallel streams; (2) the output of one subprocess can be the input of a subprocess upstream (feedback); (3) the output of one subprocess can be the input of a subprocess downstream (feedforward); (4) two subprocesses can share the same input or the same output; (5) external inputs and outputs can be fed in or drawn out at various stages.

The notions of 'simplicity' and 'structured representation' are pretty much the same as those defined above. However, the notion of 'spanning' is not. The notion of spanning for general processes permits much more complex relations between subprocesses. This notion of spanning is defined below.

Spanning for general processes

Unary subprocesses can span a process in only one way, namely, by forming a chain. The situation is a great deal more complex for general processes. General processes have multiple inputs and outputs and can be spanned by subprocesses that also have multiple inputs and outputs. This vastly increases the number of spanning possibilities. Some of these possibilities are shown in Figure 4.4: (1) the subprocesses can form parallel streams; (2) the output of one subprocess can be the input of a subprocess upstream (feedback); (3) the output of one subprocess can be the input of a subprocess downstream (feedforward); (4) two subprocesses can share the same input or the same output; and (5) external inputs and outputs can be fed in or drawn out at various stages.

Before defining spanning, a number of preliminary definitions are required:

Definition 4.14. If P is a process with u inputs and v outputs, then:

- args(P) is the set of natural numbers $\{1, \ldots, u\}$

- vals(P) is the set of natural numbers $\{1, \ldots, v\}$
- $\text{in}_i(P)$ is the ith input to P
- $\text{out}_j(P)$ is the jth output of P

These functions provide a convenient way in which to talk about the inputs and outputs of processes. If we want to say that two subprocesses share an input such that the ith input of subprocess Q is the jth input of subprocess R, then we can say that $\text{in}_i(Q) = \text{in}_j(R)$. If we want to say that two processes share an output such that the ith output of subprocess Q is the jth output of subprocess R, then we can say $\text{out}_i(Q) = \text{out}_j(R)$. If we want to say that two processes are connected in series such that the ith output of subprocess Q is the jth input of subprocess R, then we can say $\text{out}_i(Q) = \text{in}_j(R)$. If we want to say that two processes are connected in a feedback relation such that the ith input of subprocess Q is the jth output of subprocess Q, then we can say $\text{in}_i(Q) = \text{out}_j(Q)$.

These functions are convenient, however they should be used with care. The ultimate constituents of the PR-model are the notions of representation, spatiotemporal region, and counterfactual dependence. We have not accepted, and will not accept, talk of inputs and outputs as basic. Therefore, such talk cannot be left unanalysed. Some account has to be given in terms of the basic constituents of the PR-model. This account is given in Appendix B. For the moment, let us assume that such an account can be given, i.e. let us assume that statements about inputs and outputs can be reduced to conditions only involving representation tokens, spatiotemporal regions, and counterfactual dependence. This allows us to focus on the main features of the definition of spanning, rather than on the details of how talk about inputs and outputs should be understood.

When we defined spanning for unary processes, we defined an operator '$\circ$' that joins two processes together. A similar operator can be defined for general processes. Let us denote the joining operator for general processes by '$\odot$'. This joining operator is defined as follows:

Definition 4.15. $P = Q \odot R$ iff P, Q, R are processes and:

1. $\forall i \in \text{vals}(Q)$ either:
   - $\exists j \in \text{args}(R)$ such that $\text{out}_i(Q) = \text{in}_j(R)$, or
   - $\exists j \in \text{vals}(P)$ such that $\text{out}_i(Q) = \text{out}_j(P)$, or
   - $\exists j \in \text{args}(Q)$ such that $\text{out}_i(Q) = \text{in}_j(Q)$.
2. $\forall i \in \text{vals}(R)$ either:

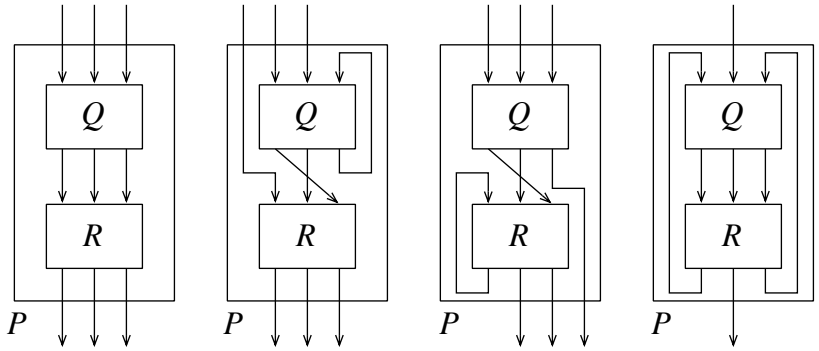

Figure 4.5: Four ways in which a process, P, can be composed of subprocesses, Q, R, such that $P = Q \odot R$ is true.

- $\exists j \in \text{args}(Q)$ such that $\text{out}_i(R) = \text{in}_j(Q)$, or
- $\exists j \in \text{vals}(P)$ such that $\text{out}_i(R) = \text{out}_j(P)$, or
- $\exists j \in \text{args}(R)$ such that $\text{out}_i(R) = \text{in}_j(R)$.

3. Every unattached input of Q, R is an input of P and every input of P is an input of either Q or R. Every unattached output of Q, R is an output of P, and every output of P is an output of either Q or R.

4. The time delays of P equal those of Q and R combined.

Let us consider each condition of Definition 4.15 in turn.

The first condition of Definition 4.15 concerns the outputs of Q, the first subprocess. It states that each output of Q can either be: (i) an input to R, (ii) an output of the overall process P, or (iii) fed back to Q as an input. These possibilities are illustrated in Figure 4.5. The second condition of Definition 4.15 concerns the outputs of R, the second subprocess. It states that each output of R can either be: (i) fed back to Q as an input; (ii) an output of the overall process P; (iii) or fed back to R as an input. These possibilities are also are illustrated in Figure 4.5. The third condition of Definition 4.15 concerns unattached inputs and outputs. It states every unattached input of Q, R is an input of P, and every unattached output of Q, R is an output of P. The final condition of Definition 4.15 states that the times delays of the overall process P match up with those of its component subprocesses. This is analogous to condition (3) of Definition 4.3, the definition of the unary joining operator, which required that $t = t_1 + t_2$. Unfortunately, a similarly simple analytic expression cannot be given for the time delays of general processes. However, by analogy with the unary case it

should be fairly clear what is meant. In Appendix B, this condition is explicitly spelt out in terms of the time delays of the individual processes.

In the unary case, if $P = Q \circ R$, then there is only one way in which P, Q and R can be connected: the output of Q must be the input of R, the input of P must be the input of Q, and the output of P must be the output of R. However, if $P = Q \odot R$, then not much can be inferred about the nature of the connections between P, Q, and R. Therefore, it is not possible to deduce the nature of the connections between P, Q, and R merely from the information that P is spanned by Q and R. The reason why this is the case is that there is *more than one way* in which P can be spanned by Q and R. The first three clauses of Definition 4.15 specify the ways in which Q and R can be connected so as to span P. The first two clauses of Definition 4.15 have three subclauses, *at least one* of which needs to be satisfied by each output of Q and R. However, there is no way of knowing *which* subclause is satisfied from the information that at least one subclause is satisfied. Therefore, there is no way of knowing the exact nature of the connections between the subprocesses from a statement about spanning. For general processes, the details of how subprocesses are connected—which input is connected to which output, and so on—cannot be inferred from spanning.

Note that although at least one subclause of the first two conditions of Definition 4.15 needs to be satisfied, there is nothing to prevent more than one subclause being satisfied. For example, an output of Q may *both* be an input to R *and* fed back as an input to Q. An output can be shared between more than one input/output, and an input can be shared by more than one input/output.

Definition 4.15 suggests at least two kinds of special case. First, it is possible for *all* of Q's outputs to satisfy only the second subclause of condition (1). In this case, the two subprocesses form two parallel streams. Instead of the outputs of Q feeding into R, they feed directly into the outputs of P. Second, it is conceivable that *all* of R's outputs satisfy only the third subclause of condition (2) in Definition 4.15. In this case, process R forms an 'island': it receives input, but does not contribute to the overall output of the process. Islands make no difference to the overall I/O behaviour of a process. Should such islands be allowed? Although islands make no difference to overall I/O behaviour, they *can* be important. For example, islands can mark aspects of structure that classify it as one of a type, they can mark vestiges of a previous structure, or they can mark sites that can be expanded into a new structure. It is worth remembering that it is a notion of computational *structure* that we wish to capture by the PR-model, not a notion of overall I/O behaviour.

Like '$\circ$', the joining operator for general processes is not commutative. If $P = Q \odot R$, then it may not be that $P = R \odot Q$. Also like '$\circ$', '$\odot$' is associative:

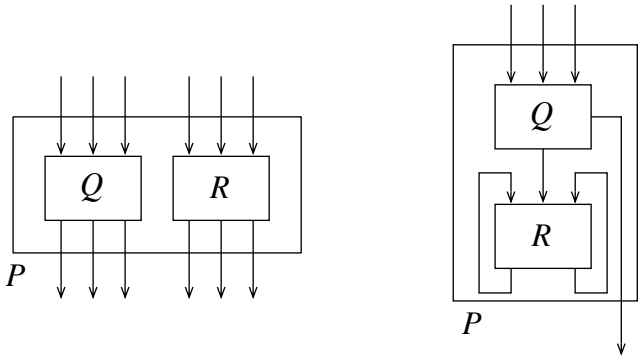

Figure 4.6: Examples of two special cases of $P = Q \odot R$. Both are legitimate cases of spanning. The left hand case is a parallel architecture. The right hand case is an architecture with an island.

if $P = (Q \odot R) \odot S$, then $P = Q \odot (R \odot S)$, and vice versa. This means that brackets are not needed in specifying multiple joins. There is no need to write $P = ((Q \odot R) \odot S) \odot \ldots \odot Z$, we can write $P = Q \odot \ldots \odot Z$.

Now that a joining operator for general processes has been defined, we can define the spanning relation for general processes. General processes $p_1, \ldots, p_n$ span P just in case the individual p_i are joined together by the general joining operator defined above. Therefore, the spanning relation for general processes is defined as follows:

Definition 4.16. Processes $p_1, \ldots, p_n$ span a process P iff $P = p_1 \odot \ldots \odot p_n$.

Definition 4.16 is structurally similar to the Definition 4.4, the definition of spanning for unary processes. However, as we saw above, for general processes, unlike for unary processes, not much can be inferred about the detailed nature of the connections between the individual processes from a statement about spanning.

Also unlike unary processes, the general processes $p_1, \ldots, p_n$ that satisfy a spanning relation need not form an ordered sequence. There may be no ordering relation between the subprocesses, but as long as they satisfy Definition 4.16, they count as spanning.

A final point is that if the singleton sequence (α) is not distinguished from the entity α, then Definition 4.4 can be shown to be a special case of Definition 4.16.

4.4 Conclusion

This concludes the construction of the PR-model. Definition 4.16 is the final piece needed to complete the definition of a computational process, Definition 4.13. We now have a semantic account of what we mean when we say that a process is computational. What remains to be done are two things, we need to: (1) justify the semantic model by showing how it can handle real-world examples of computation talk; and (2) extend the model to give an account of computational identity—an account what we mean when we say that two computations are the same or different. These two tasks occupy the next chapter.

Chapter 5

Applications

In the first part of this chapter, I show how the PR-model can be applied to real-world examples of computation talk. The examples discussed include some of the main types of computation used by cognitive science—asynchronous, connectionist, stateful, recursion-based, high-level, and so on. In the second part of this chapter, I show how the PR-model can be applied to claims about computations being the same or different. An account, based on the PR-model, is given of what it means for two computations to be the same or different. Finally, a brief discussion is given of the pragmatics of computation talk.

5.1 Examples of the PR-model

5.1.1 Synchronous computation

The definitions in the previous sections were silent about the relative timings of the subprocesses that make up a computation. The vast majority of computations with which we are familiar are synchronous: they are regulated by a central clock and each subprocess is executed, in either parallel or serial, in strict lock-step. However, it is perfectly possible to have an asynchronous computation: a computation in which subprocesses proceed at their own rate. Creating electronic computers with asynchronous architectures is an active research area.[1] It is unknown whether the computations performed by the brain are synchronous or asynchronous. There currently seems little evidence for a central clock that would regulate all computations across all parts of the brain.

In a synchronous design, each subprocess of a computation has an extra 'clock' input. The subprocesses are sensitive to the clock input so that they, for example, wait to produce their output until the clock state changes, or

[1] See Nowick et al. (1999) and references.

wait to accept their input until the clock state changes. Note that synchronous architectures can be either serial or parallel. Parallel synchronous computations are used inside modern CPUs where many operations can take place within a single clock-cycle.[2]

5.1.2 Clerk-style computation

One example of computation is the evaluation of a mathematical function using an electronic pocket calculator. The situation is familiar: one evaluates the function by performing a sequence of basic mathematical operations until an output representing the value of the function is obtained. The evaluation proceeds step-by-step using a sequence of basic operations. If the right basic operations are performed in the right order, then the function is calculated. An example is shown in Fig. 5.1 of a process that evaluates the function $f(x) = x^2 + 1$. The input to the process is a representation of the argument of the function, the output of the process is a representation of the value of the function. Intermediate tokens represent intermediate values of the calculation. All pocket-calculator-style computations are of this type. It should be clear that these processes satisfy Definition 4.13—in each case they are made up of a finite number of simple subprocesses connected in the ways described. So Definition 4.13 correctly classifies these processes as computational.

The example also shows that Turing's informal notion of computation satisfies Definition 4.13. Turing's notion of computation involves a human clerk calculating a mathematical function by hand without using undue insight or ingenuity. The situation is similar to that of the pocket calculator. The difference is that instead of performing the basic operations using an electronic calculator, the clerk performs the basic operations in his head. The clerk case, like the pocket calculator case, satisfies Definition 4.13. It may be objected that addition and multiplication are too complex to count as operations that can be performed without insight or ingenuity. These operations may, arguably, need to be broken down into simpler operations (e.g. unary addition). This can be done by replacing the processes above with an arrangement of simpler processes. Such a change means that the computational structure of the clerk case differ from the pocket calculator case, but it does not affect the overall status of what the clerk does as a computation.

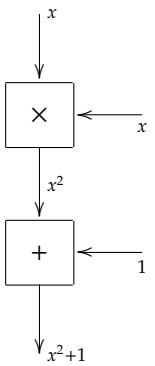

Figure 5.1: $x^2 + 1$

Note that although the clerk satisfies Definition 4.13, not all processes that satisfy Definition 4.13 need be clerks. There can be processes that count as

[2]See Hennessy and Patterson (1998), 436–440.

computational but which cannot be performed by a human clerk, e.g. massively parallel processes. This allows us to respect the claim of Chapter 1 that although Turing's definition extensionally captures the class of computational functions, it does not adequately capture all the ways in which those functions could be computed. Any method that the clerk uses to calculate the value of a mathematical function counts as computational, but there may be legitimate computational methods that elude him.

5.1.3 An adder

The previous example treated addition as a computational simple. However, one might legitimately ask how such a process could work: how can a physical process perform addition? Computation provides an answer.

Figure 5.2 shows one implementation of a 1-bit adder. A 1-bit adder is a process that computes the sum of two 1-bit numbers (e.g. 0 or 1). The process shown in Figure 5.2 has three inputs, a, b, c_{in}, representing the two input numbers and a carry flag respectively. The process has two outputs, z, c_{out}, representing the binary sum and the output carry flag. Each input or output can represent either 0 or 1. The process yields outputs that represent the sum of its inputs. Output z represents the first digit of the sum of a, b, and c_{in}, and output c_{out} represents the second (or carry) digit of the sum.

a	b	c_{in}	c_{out}	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 5.1: Truth table for a 1-bit adder.

The 1-bit adder shown in Figure 5.2 is implemented using XOR, AND and OR gates, but a 1-bit adder can be implemented in many other ways. For example, a 1-bit adder can be implemented using just AND, OR, and NOT gates. It is a general result that if a truth table characterises the I/O behaviour of a system, then an implementation of that system can be constructed that uses just AND, OR, and NOT gates. This result is a consequence of the general result that any Boolean formula can be put into a form that only uses conjunction, disjunction, and negation—disjunctive normal form. The truth table for a 1-bit

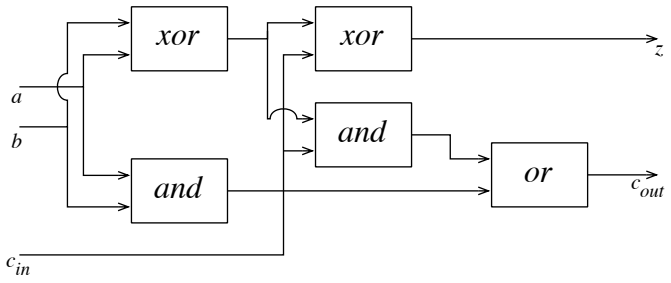

Figure 5.2: A 1-bit adder. a and b represent two 1-bit numbers, z represents their sum. c_{in} and c_{out} represent the input and output carry flags of the addition.

adder is given in Table 5.1. The corresponding disjunctive normal implementation of a 1-bit adder would involve, for each output, a collection of AND gates, some preceded by NOT gates, whose outputs feed into one big OR gate.[3]

There are a number of ways adding numbers larger than 1-bit. One way is to draw up a truth table for an n-bit adder, and then use the disjunctive normal method described above to deduce a collection of AND, OR, and NOT gates that implement that system. As with the 1-bit adder described above, this implementation would involve, for each output, a collection of AND gates, some preceded by NOT gates, whose outputs feed into one big OR gate.

Another way of making an n-bit adder is to connect a series of 1-bit adders together, so that the output carry flag of one 1-bit adder feeds into the input carry flag of another. This is called a 'ripple' adder. The individual 1-bit adders could be implemented in any way one pleases: they could be those pictured in Figure 5.2, or the disjunctive normal type described above, or yet another design. A ripple adder computes addition in a slow way: it has a minimum gate delay of $2n$. In contrast, the disjunctive normal implementation of an n-bit adder described above operates in parallel and has a small fixed gate delay of 3.[4]

Another way of building an n-bit adder is to construct a 'lookahead' adder. A lookahead adder consists of a series of modified 1-bit adders and a lookahead unit. The 1-bit adders have two additional outputs, 'generate' and 'propagate'. These outputs feed into the lookahead unit and are used to calculate the carry

[3]See Hayes (1993), 203–205. Note that even AND, OR, and NOT gates are not necessary. According to the result above, any process whose I/O behaviour can be characterised by a truth table can be implemented using just NAND gates. NAND gates are to the Sheffer stroke as AND, OR, and NOT gates are to conjunction, disjunction, and negation. Hence, we have another way in which a 1-bit adder can be implemented.

[4]See Hayes (1993), 367–368 for a comparison of the two adders.

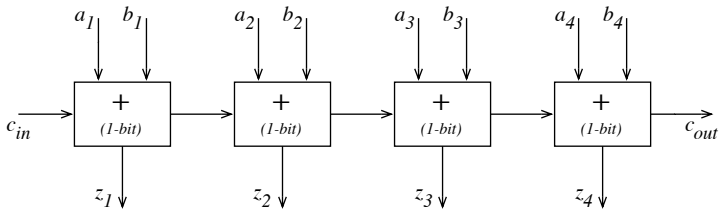

Figure 5.3: A 4-bit ripple adder. Each input and output represents either 0 or 1. $a_1, \ldots, a_4$ and $b_1, \ldots, b_4$ represent two 4-bit numbers, and $z_1, \ldots, z_4$ represents the 4-bit number which is their sum. c_{in} and c_{out} represent the input and output carry flags of the addition.

states as early as possible. The details of the implementation of the lookahead unit are complex, but the lookahead unit computes the following Boolean function:

$$\begin{aligned} c_1 &= G_0 + P_0.c_0 \\ c_2 &= G_1 + P_1.G_0 + P_1.P_0.c_0 \\ c_3 &= G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.c_0 \\ c_4 &= G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.c_0 \\ &\vdots \end{aligned}$$

An implementation of a lookahead adder is shown in Figure 5.4. The lookahead unit can be implemented in many different ways. One could, for example, use the disjunctive normal method described above to implement it using AND, OR, and NOT gates. However, there are other alternatives.[5]

Each of the n-bit adders described above illustrates a different way in which the same function, the addition function, can be computed. It should be clear that each of these ways satisfy Definition 4.13.

5.1.4 Connectionist computers

Connectionist computers consist of a collection of simple processes (neurons) connected together to form a network. The behaviour of the neurons, and the arrangement of their connections, can vary from one connectionist system to another. An example is shown in Figure 5.5.[6] The neurons have multiple inputs and multiple outputs, and compute two simple functions: their 'activa-

[5] See Hayes (1993), 369–371.
[6] See Rumelhart et al. (1986) for others.

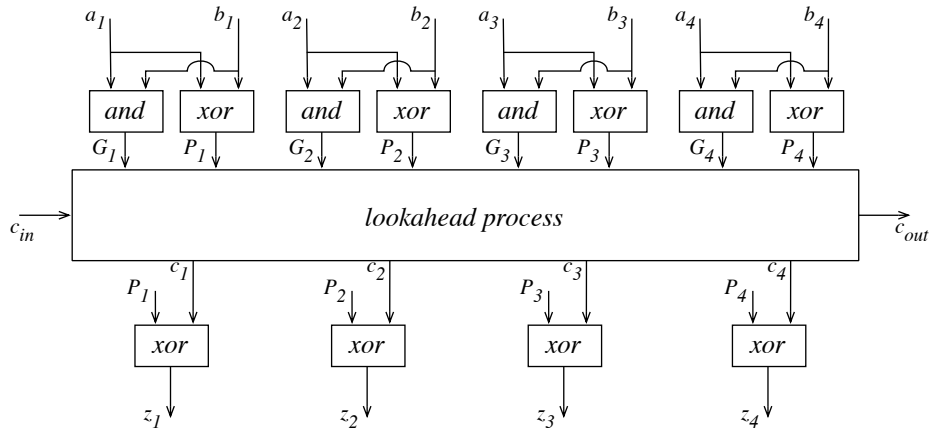

Figure 5.4: A 4-bit lookahead adder. The I/O behaviour of the lookahead process is described in the text. A possible implementation is given in Hayes (1993), 371.

tion' function and their 'output' function. The neurons also contain a stateful element, called their 'activation state' (see Section 5.1.6 for stateful elements). Neurons receive input, and based on a weighted sum of those inputs and their current activation state, they yield output. This is done in two stages: (i) the activation function maps a weighted sum of the values of their inputs and their current activation value to a new activation value, and (ii) the output function maps the current activation value to an output value that is yielded by all their outputs. The weighting given to each input is regarded as one of the most important features of the network. In learning systems, the weighting of each input is adjusted in a 'learning phase', until the network as a whole exhibits the right behaviour.

The details of how neurons are implemented are usually omitted from the specification of a connectionist system: all that matters is that a unit with the right I/O behaviour exists. The I/O behaviour of an individual neuron is typically very simple. Although their I/O behaviour is simple, if the neurons are connected in the right ways, and weigh their inputs in the right ways, then the I/O behaviour of the network as a whole can be complex. Connectionist networks appear to capable of providing a mechanism for how complex cognitive processes, such as pattern recognition, could work.[7]

Connectionist systems clearly satisfy the spanning requirement because as

[7]For a discussion of these kinds of explanations, see Bechtel and Abrahamsen (1991).

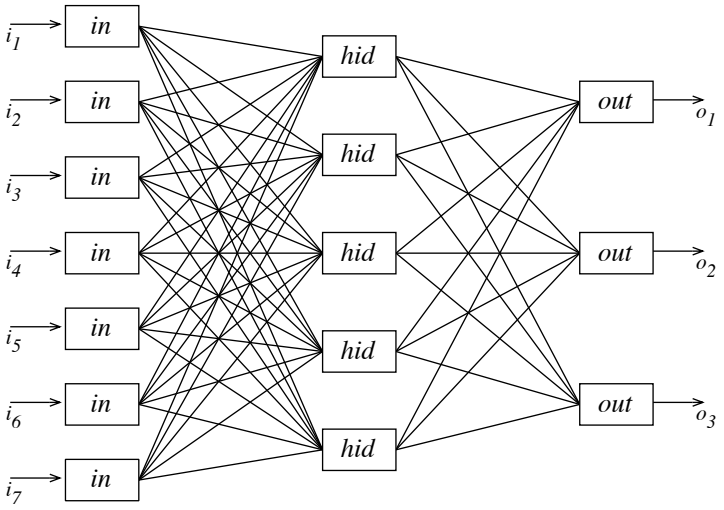

Figure 5.5: A simple connectionist network (intermediate arrowheads are omitted in order to aid legibility; the flow of the process is from left to right).

described above they are an interconnected collection of computational simples (neurons). However, in order to qualify as computational processes they *also* need to satisfy the requirement that the inputs and outputs of their computational simples are representations. This last point requires some discussion because it may appear to conflict with some descriptions of connectionist systems.

Connectionist systems are sometimes described as not requiring representations for the inputs or outputs of individual neurons. Representations, if they occur at all, appear only as features of the network activity as a whole. But this characterisation of connectionist computation is misleading. The inputs and outputs of individual neurons *do* represent. Typically, the input or output activity of a neuron represent either *activity off* ('0'), *activity on* ('1'), or an intermediate value of activity (e.g. '0.34'). The connectionist might object that the input or output of a neuron does not *represent* activity, it *is* the activity. This objection is based on a possible confusion. Normally, the question of whether an input or output represents or is the activity does not arise, because typically it does both: it both has activity, and it represents activity of a proportionate degree. However, these two properties can come apart, and it is only the representational content that matters to the system's status as performing a computation.

Suppose that a neuron can receive one of two types of signal as input: a 0 V

signal or a 5 V signal. Intuitively speaking, a 5 V signal has more 'activity' in it than a 0 V signal. Therefore, we might choose to adopt the representational convention that a 5 V signal represents *activity on* (or '1'), and a 0 V signal represents *activity off* (or '0'). However, there is nothing inherently necessary about this convention. We could just as well choose to interpret a signal of 5 V as representing *activity off*, and a signal of 0 V as representing *activity on*. Or, we could choose a more complex convention, such as one that maps voltages to levels of 'activity' in a non-linear fashion. No matter what convention we choose, it is representational content that matters both to the computational identity of the network, and to its status as performing a computation. As we saw in Section 4.2.2, unless the simple components of a computation have representations as their input and output, we cannot make sense of computation or computational identity. Therefore, if a connectionist network performs a computation, its simple processes (neurons) must operate on representations.

A connectionist might object that her model can explain how a cognitive process works in a different way. Connectionist models, interpreted as dynamical systems, provide a mechanism for how an input can be systematically transformed into an output. In this case, the network is described in terms of spreading patterns of activation, continuous dynamic relations, and 'relaxation' into stable states. This kind of explanation is not committed to representation at the level of individual neurons. However, neither is it a specifically *computational* explanation. A theory such as this—a dynamical systems theory—may be true, and it may be able to explain how certain aspects of the mind work, but it is not thereby computational.[8]

5.1.5 Stored program architectures

Inside a stored program computer, in addition to data representations, there are also program representations. The purpose of a program representation is to modify the overall function computed by the system. Depending on the program representation presented to the system, the system performs a different I/O function on its data. The mechanism via which program representations affect data representations is called the 'control' process. The control process takes programs as input and yields signals to the subprocesses in charge of data representations as output. These signals could be '0' or '1' messages that turn certain subprocesses on or off, and thus alter the processing of the data. An example of a stored program computer is shown in Figure 5.6.[9]

[8] See Hinton and Sejnowski (1986); Hopfield (1982). van Gelder (1995) advocates a purely dynamical systems approach to the mind.

[9] Real-world stored program computers are complex, but they satisfy the same general principles. For a discussion, see Hennessy and Patterson (1998), 338–388.

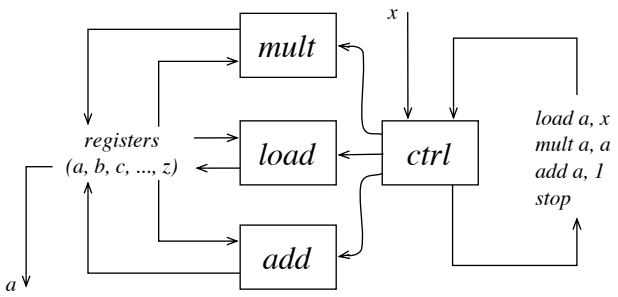

Figure 5.6: A stored program computer. A program representation is shown on the right hand side, and process that operate on data representations are shown on the left. The control unit, ctrl, mediates the interaction between the two.

The I/O behaviour of a control unit is complex, and its implementation is often non-trivial. In many electronic computers, a great deal of effort goes into the efficient implementation of a control unit. However, if the behaviour of the control unit can be specified using a truth table, then the disjunctive normal method described above can be used to find an implementation using AND, OR, and NOT gates.[10]

The way in which this discussion has been phrased may give the erroneous impression that processes are somehow 'stationary', and representations, such as programs, somehow 'move'. But nothing about the PR-model requires this. It need not be that program representations, for example, move around outside a control unit. Program representations could stay in the same spatial location, and the control process could operate on them in-place, perhaps modifying them to turn them into tokens with different representational content.

5.1.6 Memory, state, and constants

An example of a constant representation is the representation of 1 in the computation of $f(x) = x^2 + 1$ shown in Figure 5.1. This representation has the same content, and plays the same role in the computation, no matter what input is presented to the process. There are several ways in which constants can be implemented. One way is to treat constants as extra inputs to the overall process. This is what is done in the clerk-style computation shown in Figure 5.1. Another way is to build the constant into the operation performed by a computational

[10]For discussion of how to implement control units using logic gates and other computational simples, see Hennessy and Patterson (1998), 389–410, and Appendix C.

simple. This is shown in the left hand diagram in Figure 5.7. Another way is to add a memory to the system into which constants, and other representations, can be stored and recalled. This is shown in right hand diagram in Figure 5.7.

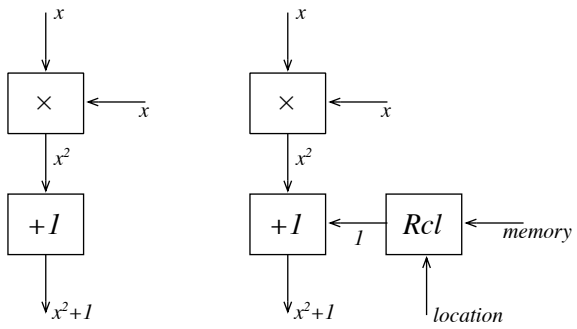

Figure 5.7: Two implementations of adding a constant.

Memory processes such as those shown in Figure 5.7 are stateful: their output depends, not just on their current input, but also on their past input. Another example of a stateful process is a neuron of a connectionist system: a neuron's current activation state depends on its past activation state as well as its current input. How can a stateful process be implemented? Stateful processes are often implemented as spanning collections of *non*-stateful processes. This is how memory is typically implemented in electronic computers. Here are some examples.

An RS-latch is a process that stores one 'bit' (a binary digit) of information. An RS-latch consists of two NAND gates, as shown in Figure 5.8. The bit of information stored by the RS-latch is represented by the state of voltage signal ϕ. The inputs to the process, S and R, are used for setting and resetting the state of ϕ. If 5 V is presented to S and 0 V is presented to R, then ϕ is set to 5 V. If 5 V is presented to R and 0 V is presented to S, then ϕ is reset to 0 V. If 5 V is presented to both S and R, then the state of ϕ is recalled without changing ϕ. The overall output of the RS-latch, d_{out}, provides access to the voltage ϕ. If d_{out} is 0 V then we say that the latch is 'open', if d_{out} is 5 V, then we say that the latch is 'closed'. (The voltage $\overline{\phi}$ is the inverse of ϕ: $\overline{\phi}$ is high when ϕ is low, and low when ϕ is high). Even though an RS-latch uses only non-stateful components, NAND gates, it is itself a stateful process, i.e. its output depends on past as well as current input. Therefore, an RS-latch can be used as a storage device. If several RS-latches are connected together, then multiple bits of data can be

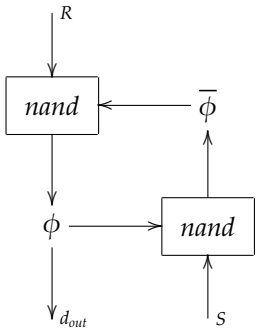

Figure 5.8: An RS-latch.

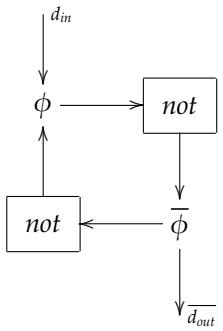

Figure 5.9: A static RAM cell.

stored.[11]

An RS-latch is an asynchronous process—it does not require a clock signal. There are a number of practical problems with using asynchronous components in a complex design. Therefore, in most electronic computers synchronous components are preferred. An example of a synchronous storage process is a JK flip-flop. A JK flip-flop has three inputs: J, K, and an input for a clock signal. The J and K inputs behave in similar way to the S and R inputs on an RS-latch. If 5 V is presented to the J input and 0 V is presented to the K input, then on the next clock tick the flip-flop is closed ($d_{out} = 5$ V). If 5 V is presented to the K input and 0 V is presented to the J input then on the next clock tick, the flip-flop is open ($d_{out} = 0$ V). If 0 V is presented to both the J and K inputs, then on the next clock tick the flip-flop's state is recalled without change. Multiple flip-flops can be connected together to store multiple bits of data. JK flip-flops are the most common way in which stateful parts of CPUs, such as their registers, are implemented.[12]

Another example of a stateful process is a RAM cell. Computer RAM is typically made up of an array of simple memory cells. There are many different designs of RAM cells, but the most common designs are based on the static RAM cell shown in Figure 5.9. The static RAM cell is an asynchronous component, but RAM cells are usually accessed in groups in a way that involves a clock signal.[13]

These three computational processes—the RS-latch, the JK flip-flop, and the static RAM cell—provide three examples of how stateful processes can be implemented using non-stateful components.

[11] For a full description of RS-latches, see Hayes (1993), 409–412.
[12] For a full description of JK flip-flops, see Hayes (1993), 432–439.
[13] For a full description of RAM cells, see Katz (1994), 357–373.

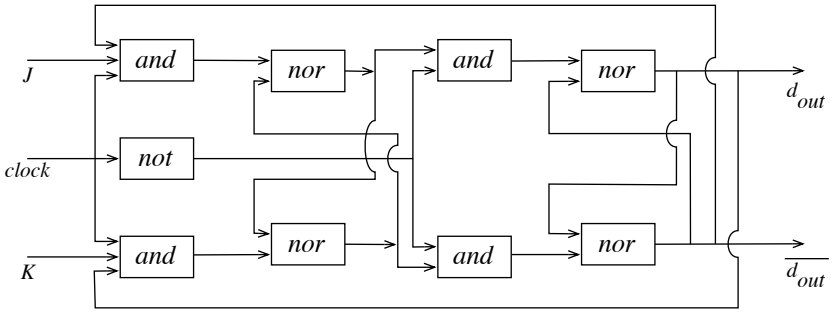

Figure 5.10: A JK flip-flop.

5.1.7 Recursion-based computation

Recursion is a distinctive way of performing a computation. A recursive computation involves a computational element that 'calls itself'. A well-known example of a recursive computation is the following algorithm for computing the factorial function:

```
fact(x){
  if x = 1 then return 1
  otherwise return x times fact(x-1)
}
```

Recursion requires two elements: (1) a control structure that allows functions to be called; and (2) self-reference that allows functions to call themselves. A recursive computational process that implements the factorial function is shown in Figure 5.11. The process involves two subprocesses: a control subprocess and a factorial subprocess. The control subprocess calls the factorial subprocess by sending it an input x that represents the number whose factorial is to be calculated. The factorial subprocess has the following I/O behaviour: if its input is a token that represents 1, then its output is a token that represents 1; otherwise, its output is a token that represents the string of characters $\ulcorner x.fact(x-1)\urcorner$, where x has been substituted for appropriately. If the control process receives a string of characters from the factorial process then it parses those characters and acts on what they say, i.e. it calls the factorial function and multiplies the result by

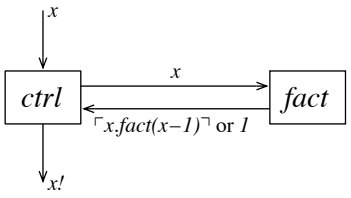

Figure 5.11: $f(x) = \text{fact}(x)$

the value of x. If the control process receives a token that represents 1 from the factorial process, then it performs the final multiplication and outputs the result.

This is just one example of a recursive computation, but other recursive computations are implemented in a similar way. It should be clear that such processes satisfy Definition 4.13. Hence, the PR-model provides a way in which to accommodate recursive computation.

5.1.8 Higher-level computational systems

The examples in the previous sections focused on low-level computation. In some cases, the most complex processes involved were logic gates. However, there is no reason why computation should be restricted to this level. As discussed in Section 4.2.3, an important feature of our computation talk is that potentially *any* process can be 'black-boxed' and treated as a computational simple. Simple processes need not be logic gates, they can be as complex as one likes so long as no question arises about how they work. This condition could be satisfied in a number of ways. It could be that we wish to maintain a studied neglect about how certain processes work; it could be that certain processes are already well-understood and we do not wish to explain them again; it could be that certain processes are not understood at all and wish to treat them as primitives; or it could be that certain processes are the responsibility of some other field of investigation. Whatever the reason, changes in our explanatory standards, *viz.* black-boxing, results in the creation of new computational simples. The PR-model does not privilege any level of computation. It allows for genuine computation to take place at any level of abstraction explanatory standards allow.

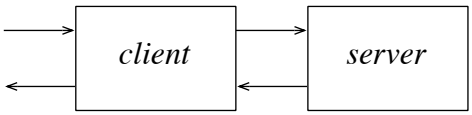

Figure 5.12: An example of a high-level computation.

The terms 'low-level' and 'high-level' are not intended to mark a qualitative distinction between different types of computation. The two names are just rough markings of different positions on a scale of explanatory standards. The term 'low-level' is roughly associated with explanations that treat logic gates as computational simples. The term 'high-level' is roughly associated with

explanations that treat more complex processes as computational simples. As stated above, neither level is inherently more privileged than any other from the point of view of the PR-model.

It is worth mentioning that black-boxing provides a way in which a single physical system can perform more than one computation. As we saw in Section 4.2.1, a process can be spanned by more than one sequence of representational subprocesses. Furthermore, we saw that if Q, R are spanning representational subprocesses, then $Q \circ R$ is also a spanning representational subprocess, and so consequently a candidate for a new computational simple. Hence, by changing one's explanatory standards (e.g. switching from 'low-level' to 'high-level'), one can potentially see the same system as performing more than one computation. In less contentious language: there may be many different computational descriptions—each of which attributes a different computational structure to a system—all of which are true, but under different explanatory standards.

5.1.9 Distributed computation

There is nothing in the PR-model that assumes that component subprocesses are physically close to each other. The subprocesses can be distant, they can even take place in different bodies, so long as they are linked by the spanning relations and counterfactual dependencies described. A single computation can be distributed over a number of spatial locations. There are many real-world examples of this: the computation performed by the SETI project, which is split over thousands of PCs worldwide; the computation performed during an international bank transaction, which is split across different countries; render farms, which render motion pictures over many different machines; and computations that take place on the internet, such as browsing the world wide web.[14]

Clark (1997), Clark and Chalmers (1998), and Hutchins (1995) argue that our cognitive processes can extend beyond the boundary of our skin/skull: cognitive processes can involve the environment, tools, and other people. The PR-model can accommodate these cases of extended cognition as cases of distributed computation. On this view, the computations involved in cognition need not take place entirely inside the body, they can be distributed between the body and the environment.

[14]For details of the SETI project, see `http://setiathome.berkeley.edu`. For distributed computation in general, see Tanenbaum and van Steen (2002).

5.2 The semantics of computation talk

How do we know that the PR-model is a good model of our computation talk? Like any semantic theory, the PR-model has to be judged on two grounds: whether it fits with our pre-existing semantic intuitions, and its possible pay-offs for other projects.

We saw in Section 5.1 that the PR-model can accommodate intuitions about a number of key examples of computation talk. The PR-model can also accommodate a number of more general intuitions about computation. For example, the PR-model can accommodate the intuition that computation involves representation, that computations are made up of parts, that computations are in a certain sense finite, that computations can be chained together, and that computation can explain how a process works. However, this intuition-based evidence is not decisive. First, it is conceivable that there are other models that can accommodate the same intuitions just as well. Second, our intuitions about what we mean by our computation talk are not themselves precise, and so it is difficult to judge how well a formal model like the PR-model fits them. Third, our intuitions about what we mean by our computation talk are, to some extent, malleable. They can be adjusted if we have good reasons for doing so, and so such evidence is not incontrovertible. Consequently, our pre-existing semantic intuitions cannot tell decisively in favour of the PR-model.

It is on the second criterion—the possible pay-offs of the PR-model—that the merits of the model should ultimately be judged. In the remainder of this chapter, and the next, I show that the PR-model has two desirable pay-offs. First, the PR-model gives a clear and intuitively satisfying account of the identity conditions of computations. Second, the PR-model shows how realism about computation is possible. This enables us to solve the two problems raised in Chapters 1 and 2: what the individuation conditions of computations are, and how we can be realists about cognitive science. It is pay-offs like these—answers to metaphysical problems and anti-realist challenges—that are the primary motivations for the PR-model. It is not unusual that once linguistic intuitions are broadly satisfied, the adoption of one semantic model over another is decided by its pay-offs for wider projects. Two famous examples are Russell's (1905) and Quine's (1980b) semantic treatment of non-referring terms ('the present King of France' and 'Pegasus'). For both Russell and Quine, their respective semantic models were motivated as a way of solving a metaphysical problem about apparent commitment to non-existing entities.

5.2.1 When a computation is performed

The PR-model gives an account of what we mean when we say that a system performs a computation. When we say that a system performs a computation, what we mean is that, inside that system, there is a computational process as defined by Definition 4.13. In other words, inside that system there is a spanning collection of explanatorily-simple representational subprocesses. For example, when we say that electronic calculator performs a computation, what we mean is that inside the calculator there is a sequence of simple representational processes, connected so as to make an overall process that calculates the value in question. Briefly put, a system performs a computation just in case it satisfies Definition 4.13.

This theory can be compared to the views of Chalmers (1996), Copeland (1996), and Mellor (1991a). According to Chalmers, a system performs a computation just in case there is mapping of distinct spatial regions of that system to the states of a CSA. According to Copeland, a system performs a computation just in case there is an honest model of a formal specification of that system. According to Mellor, a system performs a computation just in case that system contains a causal process whose inputs and outputs represent propositions.

It is worth noting that the PR-model does not attempt to provide an account of computation talk that is *synonymous* with our everyday talk of computation. The aim of the PR-model is to capture the *propositional content* expressed by our computation talk. The PR-model does not attempt, for example, to capture the force, tone, or ideas associated with that talk. The reason why propositional content is important to capture is that it is the aspect of meaning relevant to our primary interest: the truthmakers of computation talk. Propositional content is exactly that aspect of meaning that is at stake in the realist/anti-realist dispute about computation.

5.2.2 When two computations are the same

What do we mean when we say that two computations are the same or different? In Section 3.1.2, a key intuition was identified: two computations are the same just in case they have the same parts and those parts are connected in the same ways. I claimed that this intuition is the core of our notion of computational identity. However, this intuition, by itself, is not an account of computational identity. This is because the intuition does not explain what is meant by the 'parts' of a computation, what is meant by two parts being the 'same', or what is meant by two parts being connected in the 'same' ways.

The PR-model provides answers to these questions. The PR-model enables us to flesh out the intuition into a substantial account of computational identity.

This is done in the following way. First, the 'parts' of a computation are its spanning subprocesses. Second, two 'parts' are the 'same' just in case the two subprocesses are I/O equivalent. Finally, two 'parts' (i.e. subprocesses) are connected in the 'same' ways just in case their inputs and outputs are connected to the respective inputs and outputs of I/O equivalent subprocesses.

A number of aspects of this notion of computational identity have already been introduced. For example, it has already been argued in Section 4.2.2 that the identity-determining parts of a computation are its spanning subprocesses. It was also argued in Section 4.2.2 that those identity-determining parts must be representational processes. What remains to be done is to define the notions of I/O equivalence and 'being connected in the same way'. Once these notions are defined, then an account of computational identity along the lines described above can be achieved.

I/O equivalence

In Section 4.2.2, the core intuition underlying the notion of I/O equivalence was introduced. This intuition was that two processes are I/O equivalent just in case they map the same representational content to the same representational content. This intuition can be formalised in the following way.

Consider two representational processes P and Q. Suppose that process P, if presented with a token ϕ, would yield a token ψ as output. Suppose that process Q, if presented with a token ρ, would yield a token σ as output. The tokens ϕ, ρ and ψ, σ need not have any physical properties in common—processes P and Q may operate in different physical media. Nevertheless, it is possible for the two processes to be I/O equivalent. Suppose that both ϕ and ρ represent α, and both ψ and σ represent β. If this is true, then although the tokens of the two processes may not have any physical properties in common, the representational content of their input and output tokens is shared. Both process P and process Q map an input that represents α to an output that represents β. In other words, the two processes map the same representational content to the same representational content. Formally:

Definition 5.1. Two unary representational processes P (Ω_P, A, B), Q (Ω_Q, C, D) are I/O equivalent under interpretation function I iff:

1. $\forall(\phi, \psi, t) \in \Omega_P$, $\exists(\rho, \sigma, t') \in \Omega_Q$ such that $[\![\phi]\!]_I = [\![\rho]\!]_I$, $[\![\psi]\!]_I = [\![\sigma]\!]_I$.
2. $\forall(\rho, \sigma, t) \in \Omega_Q$, $\exists(\phi, \psi, t') \in \Omega_P$ such that $[\![\phi]\!]_I = [\![\rho]\!]_I$, $[\![\psi]\!]_I = [\![\sigma]\!]_I$.

Clause (1) of Definition 5.1 states that for every I/O pair ϕ, ψ of P, if ϕ represents α and ψ represents β, then there is an I/O pair of Q, say ρ, σ, for which ρ

represents α and σ represents β. Clause (2) is the converse of clause (1): for every I/O pair ρ, σ of Q, if ρ represents α and σ represents β, then there is an I/O pair of P, say ϕ, ψ, for which ψ represents α and ψ represents β. The relation of I/O equivalence is symmetrical: if P is I/O equivalent to Q, then Q is I/O equivalent to P. It is easy to show that the relation is also transitive and reflexive, and hence that it is an equivalence relation.

Note that the time delays of P and Q do not affect I/O equivalence. Two representational processes can have different time delays—one may be fast and the other slow—and both be I/O equivalent. This allows us to make sense of the idea that there are faster and slower ways of computing the same function.

It is straightforward to extend Definition 5.1 to cover general representational processes. Two general representational processes are I/O equivalent just in case, for each sequence of I/O pairs of one process, the representational content of each member of that sequence is the same as the representational content of the corresponding member of some sequence of I/O pairs for the other process.

Definition 5.2. Two representational processes P $(\Omega_P, \mathbf{A}, \mathbf{B})$, Q $(\Omega_Q, \mathbf{C}, \mathbf{D})$ are I/O equivalent under interpretation function I iff:

1. $\forall((\phi_1, \ldots, \phi_n), (\psi_1, \ldots, \psi_m), (t_1, \ldots, t_m)) \in \Omega_P$,
   $\exists((\rho_1, \ldots, \rho_n), (\sigma_1, \ldots, \sigma_m), (t'_1, \ldots, t'_m)) \in \Omega_Q$ such that, $1 \leq i \leq n, 1 \leq j \leq m$,
   $[\![\rho_i]\!]_I = [\![\phi_i]\!]_I$, $[\![\sigma_j]\!]_I = [\![\psi_j]\!]_I$.

2. $\forall((\rho_1, \ldots, \rho_n), (\sigma_1, \ldots, \sigma_m), (t'_1, \ldots, t'_m)) \in \Omega_Q$,
   $\exists((\phi_1, \ldots, \phi_n), (\psi_1, \ldots, \psi_m), (t_1, \ldots, t_m)) \in \Omega_P$ such that, $1 \leq i \leq n, 1 \leq j \leq m$,
   $[\![\rho_i]\!]_I = [\![\phi_i]\!]_I$, $[\![\sigma_j]\!]_I = [\![\psi_j]\!]_I$.

This completes the definition of I/O equivalence. Let us now turn to the other component of computational identity: the condition that the two relevant collections of subprocesses are connected in the same way.

Being connected in the same way

There is only one way in which a sequence of unary subprocesses can be connected, namely, in a chain. Two unary subprocesses are connected in the same way just in case they occupy the same positions in their respective chains. The order of the subprocesses in their chain can be inferred from their spanning predicate. If $P = p_1 \circ \ldots \circ p_n$, then computational process P is made up of subprocess p_1 followed by subprocess $p_2, \ldots,$ followed by subprocess p_n. Therefore, if $P = p_1 \circ \ldots \circ p_n$ and $Q = q_1 \circ \ldots \circ q_n$, then the respective subprocesses, p_i and q_i, of P and Q are *ipso facto* connected in the same ways.

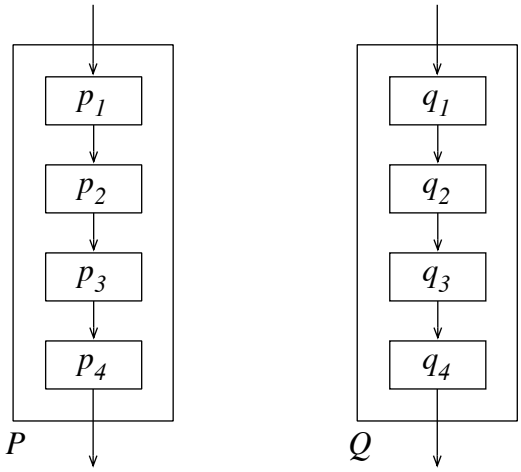

Figure 5.13: Two unary computational process whose subprocesses are connected in the same way.

Our core intuition about computational identity is that two computations are the same just in case they have the 'same' parts, and those parts are connected in the 'same' ways. The 'parts' of a computation are its spanning subprocesses. Two 'parts' (subprocesses) are the 'same' just in case they are I/O equivalent. Now we can say what it means for two unary 'parts' (subprocesses) to be connected in the 'same' ways in their respective computations. Two unary subprocesses p_i and q_i are connected in the same ways in their respective computations, P and Q, just in case $P = p_1 \circ \ldots \circ p_n$ and $Q = q_1 \circ \ldots \circ q_n$.

This allows us to define computational identity for unary computational processes:

Definition 5.3. Two unary computational processes P and Q are computationally identical iff $P = p_1 \circ \ldots \circ p_n$, $Q = q_1 \circ \ldots \circ q_n$, and $1 \leq i \leq n$, p_i and q_i are I/O equivalent.

An example of two unary computational processes that are computationally identical is shown in Figure 5.13. Processes P and Q could take place in different physical media. However, if their respective subprocesses, p_i and q_i, are I/O equivalent, then processes P and Q are computationally identical. This is how systems that do not share physical properties can perform the same computation.

Definition 5.3 covers unary computational processes, what about general computational processes? For general processes, it is not easy to say what

it means for two subprocesses to be connected in the same way. Our intuition about computational identity can therefore provisionally be formalised as follows:

Definition 5.4. Two computational processes P, Q are computationally identical iff $P = p_1 \odot \ldots \odot p_n$, $Q = q_1 \odot \ldots \odot q_n$, and $1 \leq i \leq n$:

1. p_i and q_i are I/O equivalent.

2. p_i and q_i are connected in the same ways in their respective computations.

What remains to be said is what it means for two subprocesses to be connected in the same way in their respective computations.

Let us go back to unary processes. The definition for unary processes—that two subprocesses are connected in the same way just in case they occupy the same positions in their respective chains—is a consequence of a more general principle. The more general principle is that two subprocesses are connected in the same ways just in case the inputs and outputs of each subprocess are connected to (other) inputs and outputs in the same way as the inputs and outputs of the respective subprocess in the other computation. Therefore, if p_i and q_i are connected in the same ways then, if the uth output of p_i is connected to the vth input of p_j, then the uth output of q_i is connected to the vth input of q_j. The situation is simple for unary subprocesses because there are so few ways in which unary processes can be connected. For general processes, this original principle would apply in all its complexity.

Unlike unary computational processes, the way in which general computational processes are connected *cannot be inferred from the spanning predicate*. If a *unary* computational process P is spanned by $p_1, \ldots, p_n$, then we can infer how $p_1, \ldots, p_n$ are connected: namely, in chain with p_1 followed by $p_2, \ldots,$ followed by p_n. However, if a *general* computational process P is spanned by $p_1, \ldots, p_n$, then not a great deal can be said about the way in which $p_1, \ldots, p_n$ are connected. As discussed in Section 4.3.2, there are a variety of ways in which $p_1, \ldots, p_n$ can satisfy $P = p_1 \odot \ldots \odot p_n$. How the subprocesses of a general process are connected—which input is connected to which output, and so on—cannot be inferred from the spanning predicate.

Since the spanning predicate does not specify the way in which the subprocesses are connected, we cannot require that the subprocesses of P and Q are connected in the same ways merely by specifying that $P = p_1 \odot \ldots \odot p_n$ and $Q = q_1 \odot \ldots \odot q_n$. A further condition must be added. This further condition should capture the above general principle. This principle was that two subprocesses are connected in the same ways just in case the inputs and outputs of each subprocess are connected to (other) inputs and outputs in the same way as

the inputs and outputs of the respective subprocess in the other computation. So if an ith input is connected to a jth output in one computation, then the ith input of the equivalent subprocess should be connected to the jth output of the equivalent subprocess in the other computation, and vice versa. This general principle can be formalised as follows:

Definition 5.5. p_i, q_i are connected in the same ways iff:

1. $\forall u \in \text{args}(p_i)$,
   - if $\text{in}_u(p_i) = \text{out}_v(p_j)$, then $\text{in}_u(q_i) = \text{out}_v(q_j)$
   - if $\text{in}_u(p_i) = \text{in}_v(P)$, then $\text{in}_u(q_i) = \text{in}_v(Q)$
   - if $\text{in}_u(p_i) = \text{in}_v(p_j)$, then $\text{in}_u(q_i) = \text{in}_v(q_j)$.
2. $\forall u \in \text{vals}(p_i)$,
   - if $\text{out}_u(p_i) = \text{in}_v(p_j)$, then $\text{out}_u(q_i) = \text{in}_v(q_j)$
   - if $\text{out}_u(p_i) = \text{out}_v(P)$, then $\text{out}_u(q_i) = \text{out}_v(Q)$
   - if $\text{out}_u(p_i) = \text{out}_v(p_j)$, then $\text{out}_u(q_i) = \text{out}_v(q_j)$.
3. $\forall u \in \text{args}(q_i)$,
   - if $\text{in}_u(q_i) = \text{out}_v(q_j)$, then $\text{in}_u(p_i) = \text{out}_v(p_j)$
   - if $\text{in}_u(q_i) = \text{in}_v(Q)$, then $\text{in}_u(p_i) = \text{in}_v(P)$
   - if $\text{in}_u(q_i) = \text{in}_v(q_j)$, then $\text{in}_u(p_i) = \text{in}_v(p_j)$.
4. $\forall u \in \text{vals}(q_i)$,
   - if $\text{out}_u(q_i) = \text{in}_v(q_j)$, then $\text{out}_u(p_i) = \text{in}_v(p_j)$
   - if $\text{out}_u(q_i) = \text{out}_v(Q)$, then $\text{out}_u(p_i) = \text{out}_v(P)$
   - if $\text{out}_u(q_i) = \text{out}_v(q_j)$, then $\text{out}_u(p_i) = \text{out}_v(p_j)$.

An example of two computational processes whose subprocesses satisfy Definition 5.5 is shown in Figure 5.14.

Definition 5.5 is structurally similar to Definition 4.15, the definition of the general joining operator '$\odot$'. Definition 4.15 specified the various ways in which inputs and outputs of two processes can be connected so as to make those two processes join into one larger process. Definition 5.5 recycles these criteria to say that *if* the inputs and outputs of a subprocess p_i are connected in one of these ways, *then* the inputs and outputs of the other subprocess q_i are connected in the same way, and vice versa. Clauses 1–2 of Definition 5.5 specify that q_i should have the same pattern of connections as p_i. Clauses 3–4 specify that p_i should have the same pattern of connections as q_i.[15]

[15] Like Definition 4.15, it is possible to make Definition 5.5 more precise by removing any reference to inputs and outputs and writing it purely in terms of representation tokens and spatiotemporal regions. This can be done using the method shown in Appendix B.

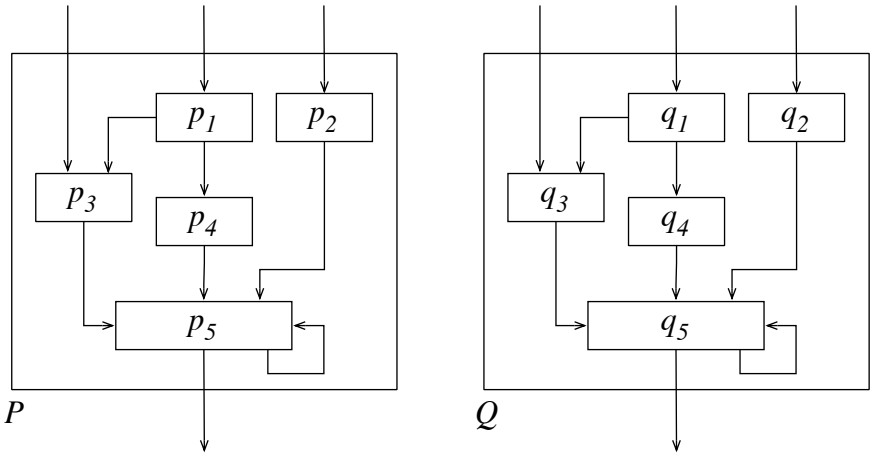

Figure 5.14: Two general computational process whose subprocesses are connected in the same way.

Definition 5.5 was last piece that we needed in order to formalise our original intuition about computational identity. Definition 5.4 now gives a complete account of computational identity. It is not hard to show that the resulting notion of computational identity is symmetrical, reflexive, and transitive. Hence, it is an equivalence relation. Note that the resulting notion of computational identity does not depend either on the time delays of the respective processes, or on the way in which their respective representations are structured.

What remains to be done is to show how this notion of computational identity relates to our talk about systems performing computations. Consider two physical systems. If there is *at least one* process in one system that is computationally identical to a process in the other system, then the two systems count as performing the same computation. This completes our account of what it means for two computational processes to be the same or different, and what it means for two systems to perform the same computation.

5.3 Pragmatics of computation talk

Like any semantic account there are counterexamples. Not every consequence of the PR-model matches up with our intuitions about real-world computation talk. The PR-model classifies some processes as computational that we normally do not, and it classifies some processes as performing different computations, when we normally think of them as performing the same computation.

What should we do about such cases? I believe that all such cases can be accounted for in terms of the pragmatics of computation talk. The PR-model is an account of the truth conditions of computation talk—it is a semantic account. The pragmatics of computation talk concerns the conditions under which statements are appropriate or inappropriate to assert. A statement may be true but inappropriate to assert in a given context, or false but appropriate to assert. A full pragmatics of computation talk is too large a task to be undertaken here, but some indication can be given of the kind of pragmatics that is required.

One consequence of the PR-model is that it classifies simple representational processes as computations. If a process is simple, then according to Definition 4.13, it is computational, since it is spanned by at least one simple representational process, namely itself. But in many cases, we do not regard simples as performing computations. How can we deal with this? One way of dealing with this case is to modify Definition 4.13 to explicitly exclude simple processes from qualifying as computations by, for example, requiring that a computational processes be spanned by more than one simple process. However, this would not respect the fact that in some rare cases we *do* wish to think of simples as performing computations. Therefore, such cases are perhaps better dealt with in the pragmatics. On this view, one could say that a simple process is a computation, but a computation of an uninteresting sort. Generally it is not appropriate to talk of simple processes as performing computations, because, among other things, they do not have an interesting computational structure that we wish to discuss.

Another kind of apparently problematic case arises from Definition 3.2, the definition of a process. According to Definition 3.2, a token, ϕ, which exists unchanged for a time t counts as a process. The process can be characterised by the triple $(\{(\phi, \phi, t)\}, A, B)$, where A and B are spatiotemporal regions that contain the token at the start and at the end of the time interval. According to Definition 3.2, a process takes place, even though nothing appears to happen. How can we deal with such a case? One way is to modify Definition 3.2 to explicitly exclude this kind of system from qualifying as a process by, for example, requiring that a process not leave all of its input tokens unchanged. However, arguably, in some cases we do wish to talk about processes that leave all of their input tokens unchanged. Therefore, it is again perhaps better to deal with such cases in the pragmatics. One could say that in such instances there is a process, but a process of an uninteresting and generally unremarkable sort. Statements about the existence of such processes are rarely asserted because such processes are rarely of interest.

Another potentially problematic case is that we often say that two systems 'perform the same computation' even though they do not satisfy Definition 5.4.

For example, we might say that two PCs perform the same computation even though one uses an addition subprocess that has 32-bit precision, and the other uses an addition subprocess that has 64-bit precision. The two addition subprocesses are not I/O equivalent since they diverge in their behaviour for outputs that represent numbers greater than 2^{32}. Therefore according to Definition 5.4, the two systems are not computationally identical. In such a case, it may make sense to distinguish strict computational identity from looser and more practical notions. For many purposes, it does not matter if certain processes are not, strictly speaking, computationally identical so long as the differences between them are so small that they can be ignored. In the case above, if we are not interested in sums larger than 2^{32}, then it might make sense to assert that the processes are computationally identical even if, under consideration, it would be admitted that the two processes are not, strictly speaking, computationally identical after all.

There are many other ways in which pragmatics of computation talk interacts with the truth-conditional semantics provided by the PR-model. However, our aim in constructing the PR-model was not to give an account of *every* aspect of meaning, pragmatic and otherwise, of our computation claims. Our aim in constructing the PR-model was to get a clear view of the truthmakers of computation talk. It was a metaphysical, not a semantic, problem that motivated the project. Now, let us turn to these truthmakers—the facts that make computation talk true or false.

Chapter 6

Metaphysics

We now have an account of what we mean when we say that a system performs a computation, and what we mean when we say that two computations are the same or different. A system S performs a particular computation just in case certain conditions obtain. What remains to be said are the kinds of metaphysical facts that make those conditions obtain. In particular, it remains to be said whether those facts can be mind-independent or whether they must be mind-dependent. If those facts can be mind-independent, then realism about computation talk is true. However, if the facts must be mind-dependent, then anti-realism about computation talk is true.

6.1 The terms of the PR-model

The PR-model analyses computation talk into conditions involving four basic notions: numerical identity, counterfactual dependence, representational content, and explanatory simplicity. In the following sections, I argue that the truth-makers of relevant claims involving these notions can be mind-independent. Let us consider each notion in turn.

6.1.1 Identity

The metaphysical facts about identity required by the PR-model have already been discussed in Sections 3.2.5 and 3.2.6. Briefly, the PR-model requires that, for any two representation tokens ϕ and ψ, it must be determinate whether ϕ is identical to ψ or not, and whether $[\![\phi]\!]$ is identical to $[\![\psi]\!]$ or not. In Section 3.2.5, I argued that if one accepts the Evans–Salmon argument, then the required facts are determinate.

Whether determinate or not, identity relations almost certainly appear to be mind-independent. Intuitively, if an entity a is numerically identical to an entity b then that is a matter that just concerns a and b. It does not require the added participation of, say, an interpreting agent. Pending good reasons otherwise, realism about identity relations seems plausible. Indeed, one could argue that at least *some* identity relations must be mind-independent in order for there to be interpreting agents at all. Consequently, it seems plausible to suppose that the truthmakers for identity claims can be mind-independent. (Note that if this condition is violated for certain entities, then those entities could still participate in computations; they just could not participate in mind-independent computations.)

6.1.2 Counterfactual dependence

One of most famous defenders of realism about counterfactual dependence is David Lewis. Lewis (1973) gives a semantics for counterfactual conditionals in terms of similarity relations between possible worlds. Lewis (1986b) then argues for a realist construal of the truthmakers for these conditionals in terms of a metaphysics of possible worlds. For Lewis, it is a brute mind-independent fact whether one possible world is closer, or further away, from another possible world. However, even if one were to reject Lewis's metaphysics, there are still reasons for a realist construal of counterfactual conditionals. These conditionals are used to analyse claims about causation, dispositions, and supervenience. If one wishes to be a realist about these claims, then one should also be a realist about counterfactual conditionals.

Note that realism about counterfactual conditionals is not the only option. Counterfactual conditionals could be given an expressivist analysis on which such conditionals express feelings, say of expectation, but perhaps do not state facts, and hence do not have truth values or truthmakers. Counterfactual conditionals could also be given a quasi-realist analysis, on which they have truth values, but their truthmakers are mind-dependent.[1] Yet another option is to analyse counterfactual conditionals as inference-tickets, which do not state facts but license and commit their users to certain inferences and explanations.[2] Realism about computation requires that these alternative approaches to counterfactual conditionals are false, or at least that they do not apply to all counterfactual conditionals.

[1] See Blackburn (1993).
[2] See Ryle (1949).

6.1.3 Explanation

The PR-model requires that the spanning subprocesses of a computation be computationally simple. Therefore, if there are to be mind-independent facts about computation, then there must be mind-independent facts about computational simplicity. Computational simplicity is defined by Definition 4.5 in terms of our explanatory standards: a representational process is computationally simple just in case its mechanism does not call for computational explanation. Therefore, if there are to be mind-independent facts about computation, then there must also be mind-independent facts about our explanatory standards.

At first sight, it looks like there is bad news in store for the realist about computation since our explanatory standards appear to be clearly mind-dependent. Our explanatory standards appear to depend on our beliefs, interests, and values: if we had different beliefs, interests, and values, then we would have different explanatory standards. Since according to the PR-model, claims about our explanatory standards are essential to claims about computation, it appears that claims about computations must also depend on our beliefs, interests, and values. Therefore, anti-realism about computation is true.

The preceding line of argument for anti-realism may be correct. It may be that our explanatory standards are an unavoidable source of anti-realism about our computation talk. It is worth noting that, even if this is the case, then by itself it is an interesting result. It was not obvious from the start that the most inextricable source of anti-realism about computation is the mind-dependence of our explanatory standards. (This anti-realist line of argument certainly does not appear in Searle (1992), Putnam (1988), or Kripke (1982)). However, I wish to suggest three ways out for a realist about computation.

Three ways out

First, a realist about computation might suggest that, contrary to appearances, our explanatory standards are mind-independent. This could be done by making a distinction between the explanatory standards we *in fact* adopt in a given situation, and the explanatory standards we *should* adopt. It may be true that the explanatory standards we in fact adopt depend on our beliefs, interests, and attitudes. However, arguably, the explanatory standards that we should adopt are an objective and mind-independent matter. Explanatory standards are not, after all, a matter of subjective taste. One cannot adopt any explanatory standards one pleases with impunity. There is a sense in which one can make a mistake with one's explanatory standards: one might choose a standard for explanation that is too low to be genuinely explanatory, no matter what one's beliefs and wishes are otherwise. There also seems to be a sense in which a

community of explainers' explanatory standards can be wrong. One possible explanation for this is that there is an objective fact of the matter about what our explanatory standards should be, and we can get this right or wrong in the standards that we in fact adopt. If this is correct, then the realist about computation has a way out. She can identify the explanatory standards relevant to computational simplicity with the explanatory standards that we should adopt, and hence with mind-independent facts. The disadvantage of this approach is that the premise on which it is based—that the norms of explanatory standards are mind-independent—is highly arguable.

A second way out for the realist about computation is to admit that our explanatory standards are mind-dependent, but claim that those explanatory standards are part of the semantic rather than metaphysical component of the analysis. Whether a system performs a particular computation depends on two factors: (1) what we mean by 'performs a particular computation'; and (2) the way the world is. The first component concerns the semantics of our computation talk. The second component concerns the metaphysical facts that make that semantic content true or false. The semantic component can be agreed to be mind-dependent: what we mean by our words clearly depends on our beliefs and attitudes. What the realist about computation claims is that *once this semantic content has been settled*, then it is a mind-independent matter whether that content is true or false. Therefore, it is possible for a realist about computation to admit that our explanatory standards are mind-dependent, provided she identifies those standards with the first rather than the second component of the analysis. On this view, the mind-dependence of explanatory standards is compatible with realism about computation.

There seems to be some justification for this move. Appeal to background knowledge and standards, including our explanatory standards, appears to be required before the semantic content of what we mean is fixed. For example, consider claims involving indexicals, such as '*This* process is the one that I wish to use'. Typically, there will be numerous processes in the region to which the speaker indicates. The way that a particular process is communicated to an audience is in virtue of shared standards, including shared explanatory standards. It is because, *inter alia*, we treat the same kinds of processes as simple, that we agree on which process is indicated in such a situation. Therefore, it does not seem unreasonable to include certain aspects of our explanatory standards in an account of what we mean. On this view, explanatory standards do not play a truthmaking role in making pre-existing semantic content true or false. Rather, explanatory standards partially determine *what that semantic content is*. The semantic content we express—that according to the realist is made true or false by mind-independent facts—has the distinctive shape it has in virtue of

our (mind-dependent) explanatory standards.[3]

The third way out for the realist is to simply drop the simplicity requirement from the PR-model. On this view, *any* collection of spanning representational subprocesses counts as a computation. Therefore, there is no need to worry about the mind-dependence of computational simplicity, since no claims about computational simplicity are made. However, even on this view, intuitions about simplicity still need to be accommodated. When we say that a process is a computation, we still mean that it is made up from processes that are in some sense simple. How can such intuitions about simplicity be accommodated? One solution is to deal with such intuitions in the pragmatics of computation talk. One could say that although it is true that any spanning collection of representational subprocesses is a computation, it is only appropriate to assert such a claim if those subprocesses are, in the relevant context, computationally simple. On this view, computational simplicity is a pragmatic feature of our computation talk, rather than semantic feature of our computation talk, or a metaphysical feature of the world.

This approach has three virtues. First, it allows the realist to drop the simplicity requirement from the PR-model, but still to acknowledge an important role that computational simplicity plays in our computation talk. Second, it allows the pragmatics of computation talk to be addressed by the pragmatics of explanation. Computational simplicity is defined in terms of our explanatory standards. The nature of our explanatory standards is part of the subject matter of the pragmatics of explanation. By classifying computational simplicity as a specifically pragmatic problem, we neatly accommodate a topic that already appears to be pragmatic under a pre-existing pragmatic heading.[4] Third, the current approach captures the intuition, described in the second approach above, that our explanatory standards are part of what we mean. Unlike the case above however, in this case our explanatory standards are part of the pragmatic, rather than the truth-conditional, component of what we mean. It is also worth mentioning that on this view the conditions for a system to perform a computation remain non-trivial. The conditions involving representation, spanning, and so on, are not easy to meet.

6.1.4 Representation

According to the PR-model, computation involves representation. Therefore, in order for there to be mind-independent facts about computation, there must be

[3] Perry (1997), for one, argues that truth-conditional semantics should include truth conditions that take into account wider elements, such as facts about speaker's intentions, knowledge and, presumably, explanatory standards.

[4] On the pragmatics of explanation, see Lewis (1986a) and van Fraassen (1980), Ch. 5.

mind-independent facts about representation. More precisely: if a computation is mind-independent, then it must be a mind-independent fact that each of its tokens, ϕ, represents a particular representational content $[\![\phi]\!]$. If this condition is not met, then the performance of that computation, and the identity of that computation, is not a mind-independent matter.

Clearly, not all representations are mind-independent. Many of the representations with which we are familiar require agents to interpret them. For example, the representation consisting of the ink-marks 'dog' represents *dogs*. However, 'dog' represents *dogs* only because we, as English-speakers, interpret it as doing so. 'Dog' does not represent *dogs* as a matter of mind-independent fact. A representation relation between 'dog' and *dogs* only obtains because of our beliefs and attitudes. In another linguistic community, the ink-marks 'dog' might represent *cats*, or might not represent anything at all. If there were no humans or similar interpreting agents, then there would be no fact about what 'dog' represents. Representations like 'dog' ineliminably require interpreters in order to have the content that they have. Let us call such mind-dependent representations 'conventional'.

Some philosophers argue that in addition to conventional representations, there are also 'natural' representations.[5] Natural representations are representations that satisfy the mind-independence criterion above. For these representations, it is a mind-independent matter that a representation token, ϕ, represents the content $[\![\phi]\!]$. It is generally expected that if natural representations are possible, then a reductive account of the natural representation relation can be given. In other words, such relations are not assumed to be *sui generis* features of the world.[6] Current proposals suggest that natural representation relations consist in either causal covaration relations, or in facts about our evolutionary history.[7]

Whatever one thinks about current proposals about the nature of natural representation relations, there seems to be good reasons for thinking that some natural representations exist. Natural representations appear to be required in order to prevent an infinite regress of conventional representations. A conventional representation represents because, *inter alia*, we have certain beliefs and attitudes. However, those beliefs and attitudes are themselves representations. As representations, they can either be conventional or natural. If conventional, then they get their content because, *inter alia*, we have certain other beliefs and attitudes. This raises the question of what determines the content of those

[5] For example, see Dretske (1981); Fodor (1990b); Millikan (1986).

[6] For example, 'It's hard to see . . . how one can be a Realist about intentionality without being, to some extent or other, a Reductionist . . . If intentionality is real, it must really be something else.' (Fodor, 1987, 97).

[7] See Dretske (1981); Fodor (1990b) for the first suggestion, and Millikan (1986) for the second.

beliefs and attitudes. If one says that those beliefs and attitudes are themselves conventional representations, then that raises the question of what determines *their* content, and so on. Ultimately, appeal to conventional representations cannot settle the issue of how a conventional representation relation holds. Each appeal to a conventional representation introduces yet more conventional representations, setting up an infinite regress. Natural representations would cut the regress and settle the representational content without introducing further representations. Unless one is willing to accept an infinite regress of conventional representation, natural representations must exist.[8]

Natural representations are the mind-independent truthmakers for the PR-model that we have been looking for: they satisfy the above criterion of mind-independence. Realism about computation is compatible with the existence of both conventional and natural representations. Just as it is possible for representation to come in two types, conventional and natural, so it is possible for computation to come in two types, mind-dependent and mind-independent. These two types are related in the following way. If some of the representations that participate in a computation are conventional, then that computation is mind-dependent. If all of the representations that participate in a computation are natural, then that computation is mind-independent.[9]

It is worth emphasising that realism about computation is not committed to the claim that *all* computations are mind-independent. Such a claim would be false given the existence of conventional representations and their manifest role in real-world computation. The computations performed by many electronic computers are conventional. It is because of mind-*dependent* conventions that 0 V represents *0*, and 5 V represents *1* in electronic computers. If we had different beliefs and attitudes, then these voltages would have different representational content, or no representational content at all. However, the existence of such cases of mind-dependent computation is, by itself, no threat to realism about computation. The realist about computation claims that mind-independent computation is *possible*, not that it is necessary. The anti-realist about computation claims that mind-independent computation is *impossible*.

An additional worry for a realist about cognitive science is that natural representations may not be available in cognitive science. It would be no help to cognitive science if natural representations exist but did not participate in cognitive processes. Fortunately, there are good reasons for thinking that natural representations, if they exist at all, do participate in cognitive processes.

[8] An alternative way out of the regress, one that is not considered here, is to be a dualist who posits mental states that are intrinsically representational.

[9] Similar points can be made concerning identity, counterfactual dependence, simplicity, and the other conditions. Realism about computation does not require that all counterfactual dependence relations be mind-independent, only that some are.

First, the regress argument given above suggests that beliefs, attitudes, and other mental representations are the best candidates for natural representations, since they are the most likely places at which to cut the regress. Second, the main positive theories of natural representation aim primarily at securing content for mental representations. Even if such theories fail to secure content for sophisticated mental representations like beliefs and desires, they still may be able to secure content for simple subpersonal mental representations, such as those used in shape recognition, syntax parsing, and other processes of interest to cognitive science.

6.1.5 Other notions

The PR-model uses a number of notions in addition to those mentioned above. For example, the PR-model uses the notion of a spatiotemporal region. I assume that it is possible to be a realist about spatiotemporal regions. The PR-model also uses the notion of a structured representation. This notion does not, by itself, pose any problem for realism about computation. We can be realists about structured representations just in case we can be realists about the relevant structures and composition operations. It is plausible that this condition is met for the cases of structured representation discussed in Chapters 3 and 4, *viz.* spatial concatenation, temporal concatenation, superposition of waveform, and coinstantiation of (mind-independent) properties.

6.2 Conclusion

The PR-model provides an account of what we mean when we say: (1) that a system performs a computation, and (2) that two computations are the same or different. If some of these computation claims are true, then they must have truthmakers. In this chapter, I have claimed that their truthmakers can be mind-independent. Provided one accepts realism about identity, counterfactual dependence, representation, and (possibly) explanation, then one can be a realist about computation. If one is willing to buy into the realist framework described above—which many philosophers do for independent reasons—then one can say that it is the world, not our interpretative attitudes, that make computation claims true or false.

Conclusion

Cognitive processes are complex and mysterious. It is not only unclear how they work, it is unclear how they are even possible. Cognitive science aims to answer these questions. The central claim of cognitive science is that cognitive processes are computations. According to cognitive science, we have certain cognitive processes because our brains perform distinctive computations. Processes that have been explained in this way include syntax parsing, shape recognition, and deductive inference.

The computational strategy has both benefits and risks attached. The benefit is that it provides a way of explaining how cognitive processes work and how they are possible. The risk is that it makes cognitive science hostage to fortunes of the notion of computation. If the notion of computation turns out to be problematic, or trivial, or interest-relative, then cognitive science is in trouble. Advocates of cognitive science should be worried, because the nature of our notion of computation in these respects is not obvious.

There are at least two major features of the notion of computation that are potentially problematic. First, as we saw in Chapter 1, the conditions under which two systems perform the same computation are unclear. Second, the way in which the physical makeup of a system relates to its computational identity is unclear. Concerns arising from these two areas have led Searle, Putnam, and Kripke to claim that we should be anti-realists about computation. According to Searle, Putnam, and Kripke, the computation that a system performs is not, and cannot be, a mind-independent feature of that system. Computation is invariably sensitive to, and requires, an interpretative agent.

If anti-realism about computation is correct, then there are at least three damaging consequences for cognitive science. First, an expected pay-off from cognitive science was an explanation of mental life in non-mental terms. If anti-realism about computation is correct, then this pay-off is undeliverable: instead of explaining mentality, cognitive science presupposes mentality from the start. Second, if anti-realism about computation is correct, then there are certain mental processes that, even in principle, cognitive science cannot ex-

plain. These mental processes are those that, according to the anti-realist, are partially constitutive of what it is to perform a computation. Third, provided we are not global anti-realists about science, anti-realism about computation entails that cognitive science is not on a par with the other sciences. While physics, chemistry, and physiology describe the world in mind-independent terms, cognitive science does not.

In this thesis, I have argued that anti-realism about computation can be resisted. If certain premises are accepted, then we can be realists about computation, and cognitive science can retain its claim to describe the world a mind-independent way. Two ways in which to argue for this conclusion should be distinguished. One way—purely negative—is to argue that there are flaws in the anti-realists' arguments. Unfortunately, attacking anti-realism does not, by itself, establish realism. Even if existing anti-realist arguments are flawed, anti-realism may still be correct for independent reasons, or realism and anti-realism may both be false. An alternative, positive, strategy is to put forward a realist account of computation as a competitor to the anti-realist accounts. If this account can explain the phenomena of computation as well its anti-realist colleagues, and we are willing to accept its premises, then there is no reason why we cannot be realists about computation. It is this second, positive, strategy for defending cognitive science that I have pursued in this thesis.

My positive account of computation comes in two parts. The first part concerns the semantics of our computation talk. This component of the account gives an account of what we mean when we say: (1) that a system performs a computation, and (2) that two computations are the same or different. The second part concerns the metaphysics. This component gives an account of the possible truthmakers of our claims about computation. Strictly speaking, the semantic component is neutral between realism and anti-realism. It is possible for an anti-realist about computation to accept the proposed semantic model. It is the second component—the nature of the truthmakers—that decides the issue of realism. If the truthmakers of computation talk can be mind-independent, then realism about computation is true. If those truthmakers must be mind-dependent, then realism about computation is false. (This is not to say that the semantic component is trivial or insignificant. On the contrary, much of the work of this thesis has gone into developing a semantic model that can *both* accommodate intuitions about computation and the possibility of mind-independent truthmakers.)

I have argued that realism about computation is true conditional on a number of assumptions. One can be a realist about computation provided one is a realist about identity, counterfactual dependence, representation, and (possibly) explanation. Realism about computation is conditional on realism in

these other areas. Fortunately, realism in these other areas has already received extensive defence. Such realism is generally considered a live option. Furthermore, realism in these other areas is generally motivated on grounds independent from any concerns about computation. Therefore, realism about computation should itself be considered a live option.

However, the argument cuts both ways. If one rejects these realist assumptions, then realism about computation is false. I believe that it is over this issue—the possibility of realism about identity, counterfactual dependence, representation, and (possibly) explanation—that the ultimate disagreement between the realist and anti-realist about computation lies. Searle, Putnam, and Kripke claim that we cannot be realists about computation. They also claim that we cannot be realists about at least one of the components above. Searle (1992) argues that mind-independent representation is impossible. Putnam (1981, 1988) argues that mind-independent representation is impossible. Kripke (1982), in his guise as a meaning sceptic, argues that mind-independent representation is impossible. None of these anti-realists accept the possibility of mind-independent representation, and none accept the possibility of mind-independent computation.

It is possible to disagree with Searle, Putnam, and Kripke over the details of their arguments. Arguably, they should have used a different model of computation talk, perhaps one more like the PR-model. However, I do not think that this is where the main problem in their argument lies. This is because *even if they had accepted the PR-model*, they would still have rejected the possibility of realism about computation. It is a disagreement about the nature of the notions mentioned above, especially representation, that is the main point of disagreement between realists and anti-realists about computation. In short, the arguments of Searle, Putnam, and Kripke are not flawed essentially because of an internal inconsistency, or lack of attention to the details of computation talk—although as they currently stand they may also be flawed in these respects. The essential problem in their arguments lies in their premises. In particular, in their premise that mind-independent representation is impossible. Searle, Putnam, and Kripke have independent reasons for claiming that this premise is true. However, this should not tempt philosophers unsympathetic to those reasons to be anti-realists about computation. Unless one wishes to buy into their wider anti-realist framework, there is no reason why one should also be an anti-realist about computation.

This response to anti-realism about computation should be distinguished from those of Chalmers (1996) and Copeland (1996). Chalmers and Copeland argue that the flaw in the arguments of Searle, Putnam, and Kripke is not that they have distinctively anti-realist premises, but that they do not have a

sufficiently sophisticated model of our computation talk. Chalmers proposes a model based on combinatorial state automata. Copeland proposes a model based on formal specification and honest models. I have argued that neither of these models, by themselves, can secure realism about computation. Even if Searle, Putnam, and Kripke were to accept these models, they would still not accept realism about computation. The semantic account of computation talk is important, but realism about computation can only be secured by accepting appropriate realist premises.

The PR-model, like the models of Chalmers and Copeland, cannot by itself defeat anti-realist arguments. What then does it do? First, the PR-model provides a rationale for why certain realist assumptions are important. It is not obvious why it is important to be a realist about X, Y, Z, unless one has established that *what we mean* when we say that a system performs a computation is that X, Y, Z obtain. Second, unlike the models of Chalmers and Copeland, the PR-model provides an account of what it means for two computations to be the same or different. The PR-model shows how claims about computational identity can have distinctive truthmakers, just as it shows how claims about the performance of computations can have distinctive truthmakers. Third, as discussed above, the PR-model plays a crucial role in the overall strategy of this thesis. That strategy is, instead of directly challenging anti-realist accounts, to provide a credible realist alternative. The PR-model specifies the details of that alternative.

We can get some idea of the final shape of the account by answering a few questions. Can two systems perform the same computation ? Yes—two systems can perform the same computation even if they do not share the same physical properties. All that needs to be shared is a pattern of counterfactual dependency relations between representations that have the same content. Can the same system perform more than one computation? Yes—this can happen in a number of ways. The tokens of the system may represent more than one thing, or, there may be more than one sequence of spanning representational subprocesses contained inside that system. Do all systems perform all computations? No—the requirements on performing a computation are non-trivial. For one thing, a system must contain representations in order to perform a computation. Brick walls do not perform computations because, among other things, they do not contain representations. Are the requirements on performing a computation too strict to be met by any system? No—as we saw in Chapter 5, the requirements are satisfied by many real-world systems, including many of those that we intuitively classify as paradigmatic cases of computation.

In Chapter 1, I argued that the man inside the Chinese room need not be able to run an algorithm putatively constitutive of mentality. At that point,

I did not have a positive account of computational identity. I argued for the conclusion using general intuitions about computation. Now that I have such an account, does the same conclusion still hold? First, can the man inside the room perform any computation? No—the conditions under which a computation is performed are too diverse for one system to be capable of performing all computations. Second, can the man inside the room perform *at least one* computation constitutive of human understanding, such as the computation that runs on Chinese speakers' brains? Perhaps—as discussed in Section 1.3.10, although the man and a Chinese brain are unlikely to be able to perform the same *low-level* computation, they may be capable of performing the same computation at a higher level of abstraction. However, this still leaves Searle in a weak position. First, he has to show that at the higher level of abstraction, the two systems *do* perform the same computation. Second, he has to show that at the higher level of abstraction the features of the algorithm that are meant to be constitutive of understanding are preserved. It is not clear how he could show either of these things. The burden of proof rests firmly on him.

Appendix A

Representational subprocesses

In this section, I argue that any plausible notion of I/O equivalence requires that the processes involved be representation-preserving. A process is presentation-preserving just in case it satisfies condition (2) of Definition 4.1, the definition of a representation process. This condition states that the mapping performed by a representational process preserves relations between representational content: $\forall \phi, \phi' \in \Phi$, if $\phi \vdash_P \psi$ and $\phi' \vdash_P \psi'$ and $[\![\phi']\!]_I = [\![\phi]\!]_I$, then $[\![\psi']\!]_I = [\![\psi]\!]_I$.

Result A.1. Any plausible notion of I/O equivalence requires that the processes involved be representation-preserving.

Suppose that Result A.1 is false, i.e. that I/O equivalence applies to processes that are not representation-preserving. Let us consider what such a notion of I/O equivalence could be. A process that is not representation-preserving cannot be I/O equivalent to any process that is representation-preserving. Therefore, let us consider only the conditions under which two non-representation-preserving processes would be I/O equivalent.

Suppose that processes R and S are not representation-preserving. Therefore, for process R, input tokens $\phi_1, \ldots, \phi_n$ represent α and map to output tokens $\psi_1, \ldots, \psi_n$ that represent β, while input tokens $\phi_{n+1}, \ldots, \phi_{n+m}$, which also represent α, map to output tokens $\psi_{n+1}, \ldots, \psi_{n+m}$ that represent something different, γ. For process S, input tokens $\rho_1, \ldots, \rho_u$ represent α and map to output tokens $\sigma_1, \ldots, \sigma_u$ that represent β, while input tokens $\rho_{u+1}, \ldots, \rho_{u+v}$, which also represent α, map to output tokens $\sigma_{u+1}, \ldots, \sigma_{u+v}$ that represent γ. Under what conditions are R and S I/O equivalent?

Three suggestions come to mind.

The first suggestion is that R and S are I/O equivalent just in case the respective numbers of β and γ-tokens are equal (i.e. $u = n$, $v = m$). This suggestion is clearly hopeless. The two processes R and S may operate in different physical media. It may be that process R takes place in silicon, while process S involves tin-cans and string. There is no reason why the respective numbers of tokens should be equal. Such a condition makes I/O equivalence too hard to achieve.

The second suggestion is that R and S are I/O equivalent just in case the proportion of β-producing to γ-producing α-input tokens is the same for both processes (i.e. $n/m = u/v$). This suggestion does not work either. Suppose that $n/m = u/v$ and the worlds in which $\phi_1, \ldots, \phi_n$ occur are close to actuality, while the worlds in which $\phi_{n+1}, \ldots, \phi_{n+m}$ occur are extremely distant (outlandish possibilities). Suppose that the worlds in which $\rho_{u+1}, \ldots, \rho_{u+v}$ occur are close to actuality, while the worlds in which $\rho_1, \ldots, \rho_u$ occur are extremely distant. Call an 'α-world' a world in which a representation of α occurs. For R, the closest α-world is always a β-world, while for S, the closest α-world is always a γ-world. Therefore, processes R and S violate the following intuition: they do not map the same representational content to the same representational content. This is our most basic intuition about I/O equivalence. Hence, even if their proportions match, R and S need not be I/O equivalent.

The third suggestion is that R and S are I/O equivalent just in case *some* pairs of tokens have the same representational content (i.e. $n, m, u, v \geq 1$). This suggestion fails too. Pick any two representational processes, P and Q, that are not I/O equivalent. Suppose that P accepts tokens $\phi_1, \ldots, \phi_n$ representing α as input, and yields tokens $\psi_1, \ldots, \psi_n$ representing β as output. Suppose that Q accepts tokens $\rho_2, \ldots, \rho_{1+v}$ representing α as input, and yields tokens $\sigma_2, \ldots, \sigma_{1+v}$ representing γ as output. P and Q are clearly not I/O equivalent. However, they can be made I/O equivalent by the simple expedient of adding impurities: a single γ-producing α-token to P, and a single β-producing α-token to Q. The resulting processes are not representation-preserving, but they satisfy the condition above for I/O equivalence. This is absurd. I/O equivalent cannot be this easy to achieve. If it were this easy to make systems I/O equivalent, then there would be no need for the computer industry to work so hard to create I/O equivalent systems.

These suggestions could be elaborated, but the prospects for such notions are doubtful. There seems little reason to think that conditions involving numbers or proportions of tokens have a bearing on I/O equivalence. We should conclude that the notion of I/O equivalence does not apply to processes that are not representation-preserving. Our intuitions about I/O equivalence do not apply to non-representation-preserving processes. Any plausible notion of I/O

equivalence requires that the processes involved be representation-preserving.

Appendix B

The joining operator

In this section, I show how to formalise Definition 4.15, the definition of the joining operator for general processes. In Section 4.3.2, the joining operator was defined informally. I shall now show that the informal definition can be rewritten in terms of expressions that only involve the basic terms of the PR-model: tokens, spatiotemporal regions, and counterfactual dependence. We shall see that, just as for unary processes, the basic terms of the PR-model are sufficient to define spanning for general processes.

The informal nature of Definition 4.15 arise from three sources. First, 'in' and 'out' functions are used to talk about inputs and the outputs, and talk of inputs and outputs is not basic in the PR-model. Second, the predicate 'unattached' is applied to inputs and outputs without an explanation of its meaning. Third, the time delays of the overall process are said to equal those of two subprocesses combined, but no account is given of what this condition means. In this section, these informal elements are replaced by conditions that only involve the basic terms of the PR-model.

First, to recap, the definition of the general joining operator was:

Definition 4.15. $P = Q \odot R$ iff P, Q, R are processes and:

1. $\forall i \in \text{vals}(Q)$ either:
   - $\exists j \in \text{args}(R)$ such that $\text{out}_i(Q) = \text{in}_j(R)$, or
   - $\exists j \in \text{vals}(P)$ such that $\text{out}_i(Q) = \text{out}_j(P)$, or
   - $\exists j \in \text{args}(Q)$ such that $\text{out}_i(Q) = \text{in}_j(Q)$.
2. $\forall i \in \text{vals}(R)$ either:
   - $\exists j \in \text{args}(Q)$ such that $\text{out}_i(R) = \text{in}_j(Q)$, or
   - $\exists j \in \text{vals}(P)$ such that $\text{out}_i(R) = \text{out}_j(P)$, or

- $\exists j \in \text{args}(R)$ such that $\text{out}_i(R) = \text{in}_j(R)$.

3. Every unattached input of Q, R is an input of P and every input of P is an input of either Q or R. Every unattached output of Q, R is an output of P, and every output of P is an output of either Q or R.

4. The time delays of P equal those of Q and R combined.

Now I shall show how to formalise this definition.

B.1 In and out functions

In and out functions provide a convenient shorthand for talking about the relationships between inputs and outputs, but there are a number of important aspects of that talk that need more careful treatment.

Relations between input and output

First, in Section 4.3.2 connections between inputs and outputs were specified by placing an identity symbol '=' between in and out functions. For example, the statement $\text{out}_i(Q) = \text{in}_j(R)$ was said to mean that the ith output of Q is the jth input of R; the statement $\text{in}_i(Q) = \text{in}_j(R)$ was said to mean that the ith input of Q is the jth input of R; and the statement $\text{out}_i(Q) = \text{out}_j(R)$ was said to mean that the ith output of Q is the jth output of R. However, characterising the connections between inputs and outputs in this way is slightly misleading. The actual relationship between connected inputs and outputs is not that of identity. Instead, it is a relationship that I shall call 'containment'.

Generally, is more accurate to say that the output of a process Q *feeds into* the input of a process R, or that it is *contained by* the input of R, rather than saying that the output of Q *is* the input of R. Typically, a connection relationship is not symmetrical. Two outputs can be connected to a single input, but it need not be that the input is identical to the two outputs. An output can feed into an input, but if that input region is spatiotemporally large, it may be that the input can receive tokens from other sources separate from the output: although the output can feed into the input, it need not be that the input feeds into the output. Of course, in certain circumstances the relationship between input and output can be symmetrical. However, such symmetry is not a necessary condition. Generally speaking, if an output is contained within an input, then that input need not be contained within the output. In what follows, I shall mark this relationship between inputs and outputs with the symbol '$\subset$'.

The first two conditions of Definition 4.15 can be rewritten with this containment relation in mind. This allows for a more accurate characterisation of the

possible relationships between inputs and outputs. Condition (1) of Definition 4.15 should be rewritten as:

- $\exists j \in \text{args}(R)$ such that $\text{out}_i(Q) \subset \text{in}_j(R)$, or
- $\exists j \in \text{vals}(P)$ such that $\text{out}_i(Q) \subset \text{out}_j(P)$, or
- $\exists j \in \text{args}(Q)$ such that $\text{out}_i(Q) \subset \text{in}_j(Q)$.

Condition (2) of Definition 4.15 should be rewritten as:

- $\exists j \in \text{args}(Q)$ such that $\text{out}_i(R) \subset \text{in}_j(Q)$, or
- $\exists j \in \text{vals}(P)$ such that $\text{out}_i(R) \subset \text{out}_j(P)$, or
- $\exists j \in \text{args}(R)$ such that $\text{out}_i(R) \subset \text{in}_j(R)$.

Reduction to basic terms

Talk of inputs and outputs is not basic in the PR-model. Therefore, one should show how the statements above, e.g. $\text{out}_i(Q) \subset \text{in}_j(R)$, can be reduced to statements involving only tokens, spatiotemporal regions, and counterfactual dependence. This can be done along lines similar to the treatment of unary processes.

First, a few preliminary definitions:

Definition B.1. If P $(\Omega, \mathbf{A}, \mathbf{B})$ is a process:

- $\text{domain}_i(P) = \{\phi_i : ((\phi_1, \ldots, \phi_i, \ldots, \phi_n), (\psi_1, \ldots, \psi_m), (t_1, \ldots, t_m)) \in \Omega\}$
- $\text{range}_i(P) = \{\psi_i : ((\phi_1, \ldots, \phi_n), (\psi_1, \ldots, \psi_i, \ldots, \psi_m), (t_1, \ldots, t_m)) \in \Omega\}$
- $\text{A}_i(P) = A_i \in (A_1, \ldots, A_n)$
- $\text{B}_i(P) = B_i \in (B_1, \ldots, B_m)$

Containment relations between inputs and outputs can be defined as follows:

Definition B.2. If Q, R are processes:

- $\text{out}_i(Q) \subset \text{in}_j(R)$ iff $\text{range}_i(Q) \subset \text{domain}_j(R)$ and $\text{B}_i(Q) \subset \text{A}_j(R)$
- $\text{out}_i(Q) \subset \text{out}_j(R)$ iff $\text{range}_i(Q) \subset \text{range}_j(R)$ and $\text{B}_i(Q) \subset \text{B}_j(R)$
- $\text{out}_i(Q) \subset \text{in}_j(Q)$ iff $\text{range}_i(Q) \subset \text{domain}_j(Q)$ and $\text{B}_i(Q) \subset \text{A}_j(Q)$
- $\text{in}_i(Q) \subset \text{in}_j(R)$ iff $\text{domain}_i(Q) \subset \text{domain}_j(R)$ and $\text{A}_i(Q) \subset \text{A}_j(R)$

This completes the treatment of talk of inputs and outputs.

B.2 Unattached inputs and outputs

Condition (3) of Definition 4.15 states that every unattached input of Q, R is an input of P, and that every unattached output of Q, R is an output of P. No account was given of what it means for an input or output to be 'unattached'. This condition can be defined in the following way:

Definition B.3. The ith input of Q is unattached iff

1. $\nexists j \in \text{vals}(R)$ such that $\text{out}_j(R) \subset \text{in}_i(Q)$, and
2. $\nexists j \in \text{vals}(Q)$ such that $\text{out}_j(Q) \subset \text{in}_i(Q)$.

Definition B.4. The ith input of R is unattached iff

1. $\nexists j \in \text{vals}(Q)$ such that $\text{out}_j(Q) \subset \text{in}_i(R)$, and
2. $\nexists j \in \text{vals}(R)$ such that $\text{out}_j(Q) \subset \text{in}_i(R)$.

Definition B.5. Every unattached input of Q, R is an input of P iff:

1. $\forall i \in \text{args}(Q)$, if the ith input of Q is unattached then $\exists j \in \text{args}(P)$ such that $\text{in}_i(Q) \subset \text{in}_j(P)$.
2. $\forall i \in \text{args}(R)$, if the ith input of R is unattached then $\exists j \in \text{args}(P)$ such that $\text{in}_i(R) \subset \text{in}_j(P)$.

It is straightforward to construct the condition for unattached outputs.

B.3 Time delays

Condition (4) of Definition 4.15 states that the time delays of the overall process, P, equal those of its two subprocesses, Q and R, combined. This condition can be formalised in the following way. First, three conventions need to be introduced. (If different conventions are introduced, then a slightly different notion of computation is obtained). The conventions are as follows. First, the time delay associated with the ith output of a process is the interval from the arrival of the *last* of that process's inputs to the yielding of its ith output. Second, if the ith input to a process receives two different tokens ϕ_1, ϕ_2 (maybe because two different outputs feed into its input region), then the token that arrives *first* is the token that counts as the input to that process. Third, if the ith output region of a process yields two different output tokens ψ_1, ψ_2 (maybe because it is a region shared between two different outputs), then the token that occurs *first* is the output of that process. An example in which all three conventions come into play is shown in Figure B.1.

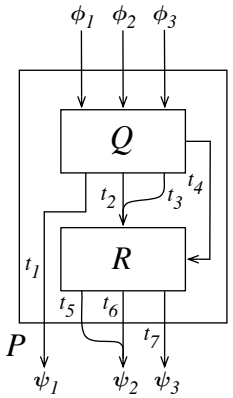

Figure B.1: An example of joining for general processes. The time delays of the component processes, Q, R, are marked.

Suppose that inputs $\phi_1, \ldots, \phi_n$ are presented to P. By Definition 4.11, P will yield $\psi_1, \ldots, \psi_m$ as output after respective times $t_1, \ldots, t_m$. Let us define $\text{delay}_i(P, \overline{\phi}_P)$ as the time delay associated with the ith output of P when $\phi_1, \ldots, \phi_n$ are presented as input to P. The expression $\overline{\phi}_P$ is an abbreviation of the condition that the tokens $\phi_1, \ldots, \phi_n$ are presented as input to P. In this case, $\text{delay}_i(P, \overline{\phi}_P) = t_i$. If $\phi_1, \ldots, \phi_n$ are presented as input to P, then some non-empty subset of those tokens will *ipso facto* be presented as input to subprocess Q because of the necessary connections between P and Q stipulated by (1) of Definition 4.15. Suppose that Q receives tokens $\_, \_, \phi_j, \_, \_$ as input. Process Q will therefore yield a sequence $\_, \_, \theta_i, \_, \_$ of tokens as output after respective time delays, $\_, \_, t'_i, \_, \_$. Define the time delay of the ith output of Q when $\phi_1, \ldots, \phi_n$ are presented to P as $\text{delay}_i(Q, \overline{\phi}_P)$. In this case, $\text{delay}_i(Q, \overline{\phi}_P) = t'_i$. If $\phi_1, \ldots, \phi_n$ are presented as input to P, then some set of tokens (perhaps intermediate tokens, perhaps the original input tokens to P) will *ipso facto* be presented to as input to subprocess R. Suppose that R receives the tokens $\_, \_, \theta_j, \_, \_$ as input. Process R will therefore yield a sequence $\_, \_, \psi_i, \_, \_$ of tokens as output after respective time delays, $\_, \_, t''_i, \_, \_$. Define the time delay of the ith output of R when $\phi_1, \ldots, \phi_n$ are presented to P as $\text{delay}_i(R, \overline{\phi}_P)$. In this case, $\text{delay}_i(R, \overline{\phi}_P) = t''_i$.

It is convenient to be able to quantify over the connections between the processes. This can be made easier by introducing three matrices, **Q**, **R**, and **I**. These matrices represent the connections between processes P, Q, and R by the placement of 0's and 1's in their appropriate rows and columns. Matrix **Q** characterises the connections between the outputs of process Q and the outputs of process P. Matrix **R** characterises the connections between the outputs of

process R and the outputs of process P. Matrix $\mathbf{I}$ characterises the connections between the outputs of Q and the inputs of R.

Definition B.6. For processes, P, Q, R, let us define $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{I}$, such that:

$$Q_{ij} = \begin{cases} 1 & \text{if } \text{out}_j(Q) \subset \text{out}_i(P) \\ 0 & \text{otherwise.} \end{cases}$$

$$R_{ij} = \begin{cases} 1 & \text{if } \text{out}_j(R) \subset \text{out}_i(P) \\ 0 & \text{otherwise.} \end{cases}$$

$$I_{ij} = \begin{cases} 1 & \text{if } \text{out}_j(Q) \subset \text{in}_i(R) \\ 0 & \text{otherwise.} \end{cases}$$

As an example, the processes in Figure B.1 would have the following matrices:

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \mathbf{Q} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{I} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We can now formalise Condition (4) of Definition 4.15. Condition (4) states that the time delays of the overall process, P, equal those of its two subprocesses, Q and R, combined. This can be formalised as follows:

Definition B.7. The time delays of P (Ω, $\mathbf{A}$, $\mathbf{B}$) equal those of Q and R combined just in case $\forall \phi_1, \ldots, \phi_n \in \Omega$, and $\forall i \in \text{vals}(P)$:

$$\text{delay}_i(P) = \min(\min_j(Q_{ij}\, \text{delay}_j(Q, \overline{\phi}_P)), \min_j(R_{ij}\, \text{combined}_j(R, \overline{\phi}_P))).$$

where $\min(x, y)$ is the smallest non-zero value of x, y, otherwise 0.

In order to see why the condition has this form, consider the various ways in which the ith output of process P could be produced. The ith output of P could *either* be an output of Q *or* an output of R. Suppose that it is an output of Q. It is possible that more than one output of Q contributes to this overall output. If more than one output of Q contributes, then the output of Q that occurs first will be the one that counts as the output of P. Therefore, the first term of Definition B.7 takes the *minimum* of the time delays of the outputs of Q that are connected to the ith output of P. Suppose instead that the ith output of P is an output of R. It is possible that more than one output of R contributes to this output of P. Therefore, a minimum of the time delays is also taken. However, unlike the previous case, some of the inputs to R may be processed by Q *before* being presented to R. Therefore, even if the ith output of P is an output of R,

the time delay associated with that output of P cannot simply be equated with the time delay of R. The time delay for the ith output of P is a function of the time delays of Q *and* the time delays of R. This function, $\text{combined}_j(R, \overline{\phi}_P)$, is defined below. Finally, it is possible that the ith output of P is an output of *both* Q and R. The output that occurs first is, as before, the output that counts as the overall output of P. Therefore, an additional minimisation operation is required, namely, the outer minimisation operation.

The function $\text{combined}_j(R, \overline{\phi}_P)$ measures the time delay from the presentation of $\phi_1, \ldots, \phi_n$ to P to the yielding of the jth output of R. This function is defined as follows:

Definition B.8. $\text{combined}_j(R, \overline{\phi}_P) = \max_k(\min_l(I_{kl}\, \text{delay}_l(Q, \overline{\phi}_P))) + \text{delay}_j(R, \overline{\phi}_P)$.

In order to see why the condition has this form, consider that process R cannot yield its jth output until all of its inputs are present. The inputs to R can come from either P or Q. The inputs from P have no associated time delay, since we are measuring time delays from the presentation of input to P. However, there are time delays associated with inputs from Q. If the kth input to R is an output of process Q, then the time delay for the arrival of that input is the *minimum* of the time delays of all the outputs of Q connected to that input of R. The first term of Definition B.8 takes the *maximum* of these k-values because *all* of the inputs to R need to be present before its own time delays can start to be measured. The additional time delay that process R then adds is given by $\text{delay}_j(R, \overline{\phi}_P)$. Therefore, the *total* time delay of the jth output of R—the delay from the presentation of $\phi_1, \ldots, \phi_n$ to P to the yielding of the jth output of R—is the time to required to produce the input of R *plus* the time required to produce the jth output of R. This is the value given by the function above. Note that if Q and R are in parallel, then the first term of $\text{combined}_j(R, \overline{\phi}_P)$ is zero, and the time delay of the jth output of R equals that associated with the jth output of R considered in isolation.

This completes the definition of the joining operator for general processes.

Bibliography

Achinstein, P. (1983). *The Nature of Explanation*. Oxford University Press, Oxford.

Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences*, 10:25–61.

Armstrong, D. M. (1997). *A World of States of Affairs*. Cambridge University Press, Cambridge.

Armstrong, D. M. (2004). *Truth and Truthmakers*. Cambridge University Press, Cambridge.

Backus, J. (1978). Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM*, 21:613–641.

Barrett, H. C. (2005). Enzymatic computation and cognitive modularity. *Mind and Language*, 20:259–287.

Bechtel, W. and Abrahamsen, A. (1991). *Connectionism and the Mind: An Introduction to Parallel Processing in Networks*. Oxford University Press, Oxford.

Belnap, N. D. (1996). Agents in branching time. In Copeland, B. J., editor, *Logic and Reality: Essays on the Legacy of Arthur Prior*, pages 239–271. Oxford University Press, Oxford.

Blackburn, S. (1993). Hume and thick connexions. In *Essays in Quasi-Realism*, pages 94–107. Oxford University Press, Oxford.

Block, N. (1980). What intuitions about homunculi don't show. *Behavioral and Brain Sciences*, 3:425–426.

Block, N. (1981). Psychologism and behaviorism. *Philosophical Review*, 90:5–43.

Block, N. (1986). Advertisement for a semantics for psychology. *Midwest Studies in Philosophy*, 10:615–678.

Block, N. (1995). The mind as the software of the brain. In Smith, E. E. and Osherson, D. N., editors, *An Invitation to Cognitive Science, Vol. 3, Thinking*, pages 377–425. MIT Press, Cambridge, MA.

Boden, M. A. (1988). *Computer Models of the Mind*. Cambridge University Press, Cambridge.

Boden, M. A. (1989). Escaping from the Chinese room. In *Artificial Intelligence in Psychology*, pages 82–100. MIT Press, Cambridge, MA.

Boolos, G., Burgess, J. P., and Jeffrey, R. C. (2002). *Computability and Logic*. Cambridge University Press, Cambridge, 4th edition.

Brody, B. A. (1980). *Identity and Essence*. Princeton University Press, Princeton, NJ.

Burge, T. (1986). Individualism and psychology. *Philosophical Review*, 95:3–45.

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

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague.

Chomsky, N. (1959). Review of *Verbal Behavior* by B. F. Skinner. *Language*, 35:26–58.

Chomsky, N. (1980). *Rules and Representations*. Cambridge University Press, Cambridge.

Chomsky, N. (1995). *The Minimalist Program*. MIT Press, Cambridge, MA.

Churchland, P. M. (1981). Eliminative materialism and the propositional attitudes. *Journal of Philosophy*, 78:67–90.

Churchland, P. S. (1986). *Neurophilosophy*. MIT Press, Cambridge, MA.

Clark, A. (1997). *Being There*. MIT Press, Cambridge.

Clark, A. and Chalmers, D. J. (1998). The extended mind. *Analysis*, 58:7–19.

Copeland, B. J. (1993). The curious case of the Chinese gym. *Synthese*, 95:173–186.

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

Copeland, B. J. (1998). Turing's O-machines, Searle, Penrose and the brain. *Analysis*, 58:128–138.

Copeland, B. J. (2002). The Chinese room from a logical point of view. In Preston, J. and Bishop, M., editors, *Views Into The Chinese Room*, pages 109–122. Oxford University Press, Oxford.

Cotogno, P. (2003). Hypercomputation and the physical Church–Turing thesis. *British Journal for the Philosophy of Science*, 54:181–223.

Cummins, R. (1983). *The Nature of Psychological Explanation*. MIT Press, Cambridge, MA.

Cummins, R. (1989). *Meaning and Mental Representation*. MIT Press, Cambridge, MA.

Cummins, R. and Cummins, D. D. (2000). *Minds, Brains, and Computers*. Blackwell, Oxford.

Davidson, D. (1984). *Inquiries into Truth and Interpretation*. Oxford University Press, Oxford.

Dennett, D. C. (1971). Intentional systems. *Journal of Philosophy*, 68:87–106.

Dennett, D. C. (1978a). *Brainstorms*. MIT Press, Cambridge, MA.

Dennett, D. C. (1978b). A cure for the common code. In *Brainstorms*, pages 90–108. Bradford Books, Montgomery, VT.

Dennett, D. C. (1980). The milk of human intentionality. *Behavioral and Brain Sciences*, 3:428–430.

Diller, A. (1994). *Z: An Introduction to Formal Methods*. Wiley, New York, 2nd edition.

Dretske, F. I. (1981). *Knowledge and the Flow of Information*. MIT Press, Cambridge, MA.

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

Evans, M. G. J. (1978). Can there be vague objects? *Analysis*, 38:208.

Fodor, J. A. (1975). *The Language of Thought*. The Harvester Press, Sussex.

Fodor, J. A. (1980a). Methodological solipsism considered as a research strategy in cognitive psychology. *Behavioral and Brain Sciences*, 3:63–109.

Fodor, J. A. (1980b). Searle on what only brains can do. *Behavioral and Brain Sciences*, 3:431–432.

Fodor, J. A. (1983). *The Modularity of Mind*. MIT Press.

Fodor, J. A. (1987). *Psychosemantics*. MIT Press, Cambridge, MA.

Fodor, J. A. (1990a). *A Theory of Content and Other Essays*. MIT Press, Cambridge, MA.

Fodor, J. A. (1990b). A theory of content, II. In *A Theory of Content and Other Essays*. MIT Press, Cambridge, MA.

Fodor, J. A. (1994). *The Elm and the Expert*. MIT Press, Cambridge, MA.

Fodor, J. A. (2000). *The Mind Doesn't Work That Way*. MIT Press, Cambridge, MA.

Fodor, J. A., Bever, T. G., and Garrett, M. F. (1974). *The Psychology of Language*. McGraw-Hill, New York.

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

Fox, J. F. (1987). Truthmaker. *Australasian Journal of Philosophy*, 65:188–207.

Geach, P. T. (1980). *Reference and Generality*. Cornell University Press, Ithaca, NY, 3rd edition.

Gibson, J. J. (1979). *An Ecological Approach to Visual Perception*. Houghton Mifflin, Boston, MA.

Harman, G. (1973). *Thought*. Princeton University Press, Princeton.

Haugeland, J. (1978). The nature and plausibility of cognitivism. *Behavioral and Brain Sciences*, 2:215–260.

Haugeland, J. (1981). Semantic engines: An introduction to mind design. In Haugeland, J., editor, *Mind Design*, pages 1–34. MIT Press, Cambridge, MA.

Haugeland, J. (1990). The intentionality all-stars. *Philosophical Perspectives*, 4:383–427.

Hayes, J. P. (1993). *Introduction to Digital Logic Design*. Addison-Wesley, Reading, MA.

Hempel, C. G. (1965). *Aspects of Scientific Explanation*. Free Press, New York.

Hennessy, J. L. and Patterson, D. A. (1998). *Computer Organization and Design*. Morgan Kauffman, San Francisco, CA, 2nd edition.

Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E., McClelland, J., and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 282–317. MIT Press, Cambridge, MA.

Hofstadter, D. R. and Dennett, D. C. (1981). Reflections on Searle's *Minds, Brains, and Programs*. In *The Mind's I*, pages 373–382. Harvester Press, Brighton.

Hogarth, M. L. (1994). Non-Turing computers and non-Turing computability. In Hull, D., Forbes, M., and Burian, R. M., editors, *PSA 1994*, volume 1, pages 126–138. Philosophy of Science Association, East Lansing, MI.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of National Academy of Sciences*, 74:2554–2558.

Horowitz, P. and Hill, W. (1989). *The Art of Electronics*. Cambridge University Press, Cambridge, 2nd edition.

Houk, J. C., Buckingham, J. T., and Barto, A. G. (1996). Models of the cerebellum and motor learning. *Behavioral and Brain Sciences*, 19:368–383.

Hutchins, E. (1995). *Cognition in the Wild*. MIT Press, Cambridge, MA.

Ito, M. (1984). *The Cerebellum and Neural Control*. Raven Press, New York.

Jackendoff, R. S. (1987). *Consciousness and the Computational Mind*. MIT Press, Boston, MA.

Johnson-Laird, P. N. (1983). *Mental Models*. Cambridge University Press, Cambridge.

Johnson-Laird, P. N. (1988). *The Computer and the Mind*. Harvard University Press, Boston, MA.

Katz, R. H. (1994). *Contemporary Logic Design*. Benjamin/Cummings, Redwood City, CA.

Kripke, S. A. (1982). *Wittgenstein on Rules and Private Language*. MIT Press, Cambridge, MA.

Lachman, R., Lachman, J. L., and Butterfield, E. C. (1979). *Cognitive Psychology and Information Processing*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Lawlor, D. W. (2001). *Photosynthesis: Molecular, Physiological and Environmental Processes*. BIOS Scientific Publishers, Oxford.

Lewis, D. K. (1970a). Causation. *Journal of Philosophy*, 70:556–567.

Lewis, D. K. (1970b). General semantics. *Synthese*, 22:18–67.

Lewis, D. K. (1973). *Counterfactuals*. Blackwell, Oxford.

Lewis, D. K. (1979). Counterfactual dependence and time's arrow. *Noûs*, 13:455–476.

Lewis, D. K. (1983). New work for a theory of universals. *Australasian Journal of Philosophy*, 61:343–377.

Lewis, D. K. (1986a). Causal explanation. In *Philosophical Papers*, volume 2, pages 214–240. Oxford University Press, Oxford.

Lewis, D. K. (1986b). *On the Plurality of Worlds*. Oxford University Press, Oxford.

Lewis, D. K. (1988). Vague identity: Evans misunderstood. *Analysis*, 48:128–130.

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

Lycan, W. G. (1980). The functionalist reply (Ohio State). *Behavioral and Brain Sciences*, 3:434–435.

Lycan, W. G. (1987). *Consciousness*. MIT Press, Cambridge, MA.

Marr, D. (1969). A theory of cerebellar cortex. *Journal of Physiology (London)*, 202:437–470.

Marr, D. (1982). *Vision*. W. H. Freeman, San Francisco, CA.

Mellor, D. H. (1991a). How much of the mind is a computer? In *Matters of Metaphysics*, pages 61–81. Cambridge University Press, Cambridge. First published 1988.

Mellor, D. H. (1991b). Properties and predicates. In *Matters of Metaphysics*, pages 170–182. Cambridge University Press, Cambridge.

Mellor, D. H. (2000). The semantics and ontology of dispositions. *Mind*, 109:757–780.

Miller, A. and Wright, C., editors (2002). *Rule-Following and Meaning*. Acumen, Chesham.

Millikan, R. G. (1986). Thoughts without laws: Cognitive science with content. *Philosophical Review*, 95:47–80.

Molesworth, W., editor (1837–1845). *Elements of Philosophy, The English Works of Thomas Hobbes of Malmesbury*, volume 4. J. Bohn, London.

Montague, R. (1974). *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, Yale.

Mulligan, K., Simons, P., and Smith, B. (1984). Truth-makers. *Philosophy and Phenomenological Research*, 44:287–321.

Neisser, U. (1967). *Cognitive Psychology*. Prentice-Hall, Englewood Cliffs, NJ.

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

Noonan, H. (2003). *Personal Identity*. Routledge, London.

Noonan, H. W. (1980). *Objects and Identity*. Martinus Nijhoff, The Hague.

Noonan, H. W., editor (1993). *Identity*. Dartmouth, Aldershot.

Nowick, S. M., Josephs, M. B., and van Berkel, C. H. (1999). Scanning the special issue on asynchronous circuits and systems. *Proceedings of the IEEE*, 87:219–222.

Ord, T. (2002). Hypercomputation: Computing more than the Turing machine. Master's thesis, University of Melbourne, Melbourne.

Pellionisz, A., Llinás, and Perkel, D. H. (1977). A computer model of the cerebellar cortex of the frog. *Neuroscience*, 2:19–36.

Penrose, R. (1989). *The Emperor's New Mind*. Oxford University Press, Oxford.

Perry, J. (1997). Indexicals and demonstratives. In Hale, B. and Wright, C., editors, *A Companion to the Philosophy of Language*, pages 586–612. Blackwell, Oxford.

Plantinga, A. (1974). *The Nature of Necessity*. Oxford University Press, Oxford.

Putnam, H. (1975a). The meaning of "meaning". In *Mind, Language and Reality, Philosophical Papers, vol. 2*, pages 215–271. Cambridge University Press, Cambridge.

Putnam, H. (1975b). *Mind, Language and Reality, Philosophical Papers, volume 2*. Cambridge University Press, Cambridge.

Putnam, H. (1981). *Reason, Truth and History*. Cambridge University Press, Cambridge.

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

Pylyshyn, Z. W. (1980). The "causal power" of machines. *Behavioral and Brain Sciences*, 3:442–444.

Pylyshyn, Z. W. (1984). *Computation and Cognition*. MIT Press, Cambridge, MA.

Pylyshyn, Z. W. (1999). What's in your mind? In LePore, E. and Pylyshyn, Z. W., editors, *What is Cognitive Science?*, pages 1–25. Blackwell, Oxford.

Quine, W. V. O. (1980a). Identity, ostension, and hypostasis. In *From a Logical Point of View*, pages 65–79. Harvard University Press, Cambridge, MA.

Quine, W. V. O. (1980b). On what there is. In *From a Logical Point of View*, pages 1–19. Harvard University Press, Cambridge, MA.

Restall, G. (1996). Truthmakers, entailment and necessity. *Australasian Journal of Philosophy*, 74:331–340.

Rey, G. (1986). What's really going on in Searle's "Chinese room". *Philosophical Studies*, 50:169–185.

Rey, G. (2003). Chomsky, intentionality, and a CRTT. In Antony, L. M. and Hornstein, N., editors, *Chomsky and his Critics*, pages 105–139. Blackwell, Oxford.

Roberts, L. (1990). Searle's extension of the Chinese Room to connectionist machines. *Journal of Experimental and Theoretical Artificial Intelligence*, 2:185–187.

Rosen, G. (1990). Modal fictionalism. *Mind*, 99:327–354.

Rumelhart, D. E., McClelland, J., and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, Cambridge, MA.

Russell, B. A. W. (1905). On denoting. *Mind*, 14:479–493.

Ryle, G. (1949). *The Concept of Mind*. Hutchinson, London.

Salmon, N. U. (1981). *Reference and Essence*. Blackwell, Oxford.

Salmon, N. U. (1986). Modal paradox: parts and counterparts, points and counterpoints. In French, P. A., Uehling, T. E., and Wettstein, H. K., editors, *Midwest Studies in Philosophy XI: Studies in Essentialism*, pages 75–120. University of Minnesota Press, Minneapolis.

Searle, J. R. (1980a). Author's response. *Behavioral and Brain Sciences*, 3:450–456.

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

Searle, J. R. (1990). Is the brain's mind a computer program? *Scientific American*, 262:20–25.

Searle, J. R. (1992). *The Rediscovery of the Mind*. MIT Press, Cambridge, MA.

Searle, J. R. (1994). Searle, John R. entry. In Guttenplan, S., editor, *A Companion to the Philosophy of Mind*, pages 544–550. Blackwell, Oxford.

Searle, J. R. (1997). *The Mystery of Consciousness*. Granta Books, London.

Segerberg, K. (1996). To do and not to do. In Copeland, B. J., editor, *Logic and Reality: Essays on the Legacy of Arthur Prior*, pages 301–313. Oxford University Press, Oxford.

Simon, H. A. (1969). *The Sciences of the Artificial*. MIT Press, Cambridge, MA.

Sloman, A. (1986). What sorts of machines can understand the symbols they use? *Aristotelian Society Supplement*, 60:61–80.

Smolensky, P. (1988). The proper treatment of connectionism. *Behavioral and Brain Sciences*, 11:1–74.

Sprevak, M. D. (2005). The Chinese carnival, review of J. M. Preston & M. Bishop (Eds.) *Views into the Chinese Room*. *Studies in History and Philosophy of Science, Part A*, 36:203–209.

Stalnaker, R. (1976). Possible worlds. *Noûs*, 10:65–75.

Stalnaker, R. (1988). Vague identity. In Austin, D. F., editor, *Philosophical Analysis: A Defense by Example*. Kluwer, Dordrecht.

Stich, S. P. (1983). *From Folk Psychology to Cognitive Science*. MIT Press, Cambridge, MA.

Stich, S. P. and Warfield, T. A., editors (1994). *Mental Representation: A Reader*. Blackwell, Oxford.

Sudkamp, T. A. (1998). *Languages and Machines*. Addison-Wesley, Reading, MA, 2nd edition.

Tanenbaum, A. S. and van Steen, M. (2002). *Distributed Systems: Principles and Paradigms*. Prentice Hall, Englewood Cliffs, NJ.

Turing, A. M. (1936–1937). On computable numbers, with an application to the *Entscheidungsproblem*. *Proceeding of the London Mathematical Society, series 2*, 42:230–265.

Turing, A. M. (1939). Systems of logic based on ordinals. *Proceedings of the London Mathematical Society, series 2*, 45:161–228.

van Fraassen, B. C. (1980). *The Scientific Image*. Oxford University Press, Oxford.

van Gelder, T. (1995). What might cognition be, if not computation? *Journal of Philosophy*, 91:345–381.

van Inwagen, P. (1990). *Material Beings*. Cornell University Press, Ithaca, NY.

Wandell, B. A. (1995). *Foundations of Vision: Behaviour, Neuroscience and Computation*. Sinauer, Sunderland, MA.

Welch, P. D. (2004). On the possibility, or otherwise, of hypercomputation. *British Journal for the Philosophy of Science*, 55:739–746.

Wertheimer, M. (1985). A Gestalt perspective on computer simulations of cognitive processes. *Computers in Human Behavior*, 1:19–33.

Wiggins, D. R. P. (1986). On singling out an object determinately. In Pettit, P. and McDowell, J. H., editors, *Subject, Thought and Context*, pages 169–180. Oxford University Press, Oxford.

Wiggins, D. R. P. (2001). *Sameness and Substance Renewed*. Cambridge University Press, Cambridge.

Williamson, T. (1990). *Identity and Discrimination*. Blackwell, Oxford.