Minds & Machines (2011) 21:179–202
DOI 10.1007/s11023-011-9232-4

Descartes Among the Robots

Computer Science and the Inner/Outer Distinction

Graham White

Received: 2 December 2009 / Accepted: 28 September 2010 / Published online: 9 February 2011

Abstract We consider the symbol grounding problem, and apply to it philosophical arguments against Cartesianism developed by Sellars and McDowell: the problematic issue is the dichotomy between inside and outside which the definition of a physical symbol system presupposes. Surprisingly, one can question this dichotomy and still do symbolic computation: a detailed examination of the hardware and software of serial ports shows this.

Keywords Physical symbol system · Grounding problem · Inner/outer · Cartesianism

Introduction

This article examines what is known as the Symbol Grounding Problem: this is a problem which arises out of the distinction between symbols—defined syntactically—and what symbols mean. The thought behind the problem seems to be this: computers can be understood as mere symbol-manipulators. We, however, are not pure symbol-manipulators: we use symbols in a *grounded* way, or, in other words, the symbols that we use give us access to their referents. This grounding must, one argues, be implemented in physical processes—Harnad (1990) talks of "meanings … in our brains"—and we would like to also implement those physical processes in computers, or some such machines. That would mean that computers, too, had come to have *grounded* symbols. The history of attempts to solve this problem has not, however, been encouraging. The "symbol grounding problem", then, can be taken to require either a description of how to give computers grounded symbols, or to

G. White (✉)
School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK
e-mail: graham@eecs.qmul.ac.uk

show either that it is, for some reason or other, impossible, or that it is not necessary. But first, we need to do some analysis. One of the main results of this analysis will be that, although the symbol grounding problem is defined in terms of symbols and meaning, in order for the problem to be non-trivial one has to specify the context in which symbols are defined and in which meaning has to be represented: and this specification of a context will require one to make a distinction between the inside and outside of a device (in practical terms, either inside and outside of a person's head, or inside and outside of a computer's box).

We will also have to deal with a related inside/outside distinction, this time a metaphorical one. This one (which I will talk about as intrinsic/extrinsic) will be between capacities which an thing has because of its nature (because, for example, the thing is an organism and its evolutionary history has endowed it with them), as opposed to capacities which it has been given by some other agent (because, for example, the thing is an artefact, such as a computer, and the capacities have been given to it by a designer or a computer programmer). We will be comparing the use of symbols by humans and computers: symbol use is, of course, intrinsic for humans but extrinsic for computers. Nevertheless, we will argue that one can learn a great deal about human symbol use by looking at symbol use by computers, and we will conclude with a rather speculative story about how computer-like devices might possibly end up with symbols which were not merely grounded, but *intrinsically* grounded.

There are two caveats to make about this whole argument. Firstly, I am not attempting to say definitively what the symbol grounding problem is: there is a good deal of disagreement in the cognitive science community about how precisely the problem should be defined (for example, the two referees of an earlier version of this paper disagreed subtly about this question). I am attempting to do the following: to make precise a formulation of the symbol grounding problem which has, at least, some textual support, and to analyse the problem, thus formulated, in such a way as to end up with a position which is—or so I hope—interesting for the present state of cognitive science.

Secondly, one of the reasons why the symbol grounding problem is as hard to define as it is is because there are underlying conceptual issues which are, themselves, difficult to resolve: one of these is Descartes' distinction between mind and body (and hence the motivation for the title of this piece). So some of the analysis will take some time to perform, for which I can only ask the reader's indulgence.

The Nature of Symbols: Syntax

There are two key technical concepts in setting up the Symbol Grounding Problem: the idea of characterising symbols syntactically, and the idea of the meaning of a symbol. These are problematic concepts, and neither of them is very well explained in Harnad's descriptions of the grounding problem: we will use later work, principally Boden (1988) and Millikan (1984) to give some analytical precision to them.

In this section we examine the syntactic characterisation of symbols. We start with the syntactic part of Harnad's characterisation of symbol systems: a physical symbol system is

1. "a set of arbitrary 'physical tokens' scratches on paper, holes on a tape, events in a digital computer, etc. that are
2. manipulated on the basis of 'explicit rules' that are
3. likewise physical tokens and strings of tokens. The rule-governed symbol-token manipulation is based
4. purely on the shape of the symbol tokens (not their their 'meaning'), i.e., it is purely syntactic, and consists of
5. 'rulefully combining' and recombining symbol tokens. There are
6. primitive atomic symbol tokens and
7. composite symbol-token strings …"(Harnad 1990)

We leave out the last half of the last item: it concerns semantics, and we will discuss it later. Item 3 is hard to understand, because it is vulnerable to an obvious regress argument of the sort given in Carroll (1895). However, it is clearly pointing towards some sort of cognitive penetrability (Boden 1988, p. 39); we can probably satisfy it by saying that there should be rules of two sorts, explicit and implicit rules, that the explicit rules should be given by strings of physical tokens, and that there should be, in some sense, "enough" explicit rules. It is not easy to say what it is to have enough explicit rules, but one of the things that we probably need is that the explicit rules should form the basis for a reasonable metatheory.

What this characterisation is aiming at is a specification of the nature of symbols: it is analogous to what Boden describes as "functional architecture", that is, "those properties of the hardware of a computational system which make possible, and constrain, the information-processing going on" (Boden 1988, p. 38), and it is also similar to the characterisation of physical symbol systems in Newell and Simon (1976). So this part of Harnad's project is unproblematic.

The Nature of Symbols: Semantics

Harnad then adds to the above conditions

> The entire system and all its parts—the atomic tokens, the composite tokens, the syntactic manipulations both actual and possible and the rules—are all

8. 'semantically interpretable' The syntax can be systematically assigned a meaning (e.g., as standing for objects, as describing states of affairs). (Harnad 1990)

Although this condition seems intuitively clear, it is quite hard to make formal sense of. Here are some of the difficulties.

Firstly, it cannot just mean that the symbols of the syntax can be assigned semantic values in a compositional way. This assignment of semantic values is, admittedly, what we think of as semantics when we are talking about formal logic: but, in formal logic, *any* consistent theory can be assigned semantic values in that

sense. Furthermore, for the purposes of symbolic logic, the semantic values so assigned could well be constructed out of the syntax itself: in fact, the standard proof of the completeness theorem for first order logic (which is what guarantees that suitable theories have semantics) simply constructs a model out of equivalence classes of terms in the language, using what is known as the Lindenbaum construction. Harnad, quite reasonably, wants more than that: he wants "objects" or "states of affairs" or something of the sort, rather than semantic values constructed mathematically from the syntax. Formal logic, however, is blind to the difference between semantic values which live "in the world" and those which are constructed out of mere syntax, so an analysis of the idea of groundedness cannot simply use the machinery of formal logic.[1]

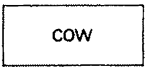
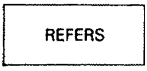
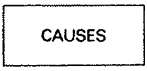
It looks, then, as if the point of Harnad's definition is that we are investigating a particular language, and that, for this language, we already have a model, *whose semantic values live in the world of spatio-temporal particulars*, and that the meaning demanded by the grounding problem should be the appropriate semantic value in that model. It is easy to imagine how this might work for singular terms which denote actually existing individuals, but other parts of our language might be more problematic: what, for example, corresponds to predicates? Propositions? Disjunctive propositions? Negative existential propositions? Now the possibility of providing answers to these questions is philosophically very contested: see, for example, the literature on truthmakers referred to by Glanzberg (2009). Even if one could successfully resolve these issues, it seems a bit of a detour: and, in any case, philosophical approaches which give a broad array of real-world semantic values generally do so by using exotic entities such as states of affairs, or universals, and these entities (even granted that they are part of the real world) would give very little concrete help to those in the cognitive science trade. Computers have no sensors for such objects.

What Would Grounding Be?

There are two more conceptual matters to be cleared up: the first has to do with the characterisation of possible solutions. It is so far quite unclear what a solution to the grounding problem would be in concrete terms. There is, one would think, some difference between ungrounded processes of syntactic manipulation and fully grounded cognitive processes. We would like to think that the difference might lie in extra knowledge, or information, or some other cognitive resource. So what would that extra item be?

It is tempting to think that the grounding resource might lie in something like the semantics of first-order logic, or, at least, some variant of it which assigned real-world referents to suitable syntactic items. But even this seems quite problematic when we attempt to cash it out. As Putnam writes,

[1] Cf. (Millikan 1984, p. 87): "The specialness that turns a mathematical mapping function into a representation-related relation in a given case must have to be some kind of special status that this function has in the real, the natural, or the *causal* order rather than the logical order."

> If concepts are particulars ('signs'), then any concept we might have of the relation between a concept and another object is *another sign*. But it is unintelligible … how the sort of relation the metaphysical realist envisages as holding between a sign and its object can be singled out either by holding up the sign itself, thus or by holding up yet another sign, thus
>
> 
>
>
> *or* by holding up yet another sign, thus
>
> 
>
>
> or perhaps
>
> 
>
>
> (Putnam, 1978, pp. 126f.); cf. (Millikan, 1984, p. 330)

That is: if we try to give an answer to the grounding problem in the usual way of logical semantics, then at least some of the symbols in the explanation (the ones which pick out the semantic values, and maybe also the metatheoretic symbols like 'refers') have to be grounded for the answer to say something: but, if they have to be grounded, then we are faced with a regress. So we need yet another ingredient in our solution of the grounding problem: an account of the *logical form* of the desired answer.

Grounding and Natural Kinds

The final issue has to do with how we can rule out trivial solutions of the problem. Here is a candidate trivial solution.

Example 1 Suppose we have an instance of the symbol grounding problem, so that we have symbols, and these symbols have semantics in the real world. Then define another instance of the problem: replace each symbol by the ordered pair of it and its real-world referent. The syntactic relations between symbols will now simply be given by the old relations on the first components of the ordered pairs: the grounding relation will be given by the projection onto the second component. What this shows us is that, firstly, there are symbol systems which have a trivial grounding problem, and, secondly, that each symbol system with a real-world semantics has a grounded system, isomorphic to it *qua* purely syntactic system, which is trivially grounded. But this, of course, will not do: the grounding problem arises because we are given symbol systems—for example, those which constitute computers and robots—for which we would very much like to have groundings, but for which we do *not* have groundings.

But this example raises another problem. As we shall argue, Harnad views syntactic systems as having natures which are constituted entirely by the syntactic relations between the symbols; he says, of his definition of a symbol system, that

> [n]one of these criteria is arbitrary, and, as far as I can tell, if you weaken them, you lose the grip on what looks like a natural category and you sever the links with the formal theory of computation, leaving a sense of "symbolic" that is merely unexplicated metaphor (and probably differs from speaker to speaker). Hence it is only this formal sense of "symbolic" and "symbol system" that will be considered in this discussion of the grounding of symbol systems. (Harnad 1990)

Note here that the semantics of a symbol system is not supposed to be part of its definition: all that is necessary is that the system should have *some* semantics or another, and whether it has such a semantics or other only depends on its syntax. So, if we implement the same system again, but in such a way that the syntactic interactions are preserved, then the two symbol systems will be type-identical.

But our example shows that groundedness is not a property of syntactic systems solely *qua* syntactic systems, because the two systems in the example were, *qua* syntactic systems, isomorphic. Thus, we seem to be concerned with implementations of syntactic systems in particular causal contexts.

A possible line might be this: maybe a solution to the grounding problem should consist in a description of distinct sorts of processes, which yield or implement grounded cognition, and which take place in brains but not in computers, or at least not so far? We have in mind something like Adams and Aizawa's position:

> We maintain that there is something distinctive about the brain. There are natural kinds of processes that happen to occur only within the brain. These processes differ from neurophysiological processes in so far as they consist of … causal operations on nonderived representations …. These processes also differ, we suppose, from typical processes that extend into the world from brains and from processes found in typical machines. In other words, we hypothesise that there are within the brain natural laws that are not identical to physical, chemical, biological or neurophysiological covering laws spanning the cranium. (Adams and Aizawa 2009, p. 80)

Now these authors are here talking of the difference between cognitive and non-cognitive processes *tout court*, whereas we are inquiring into the difference between grounded and non-grounded cognitive processes. However, the possibility still stands in our case: it is not inconceivable that a solution to the grounding problem could take the form that they sketch. And this at least provides a possible alternative to the first-order-model-theory type of answer: what we should aim at is a description of grounded cognitive processes.

Possible Answers

We have, then, seen that there are serious questions of logical form which we have to solve before solving the grounding problem. We can now survey a number of approaches which have, implicitly or implicitly, made their minds up about some of these questions.

Newell and Simon

Newell and Simon conjectured that "A physical symbol system has the necessary and sufficient means for general intelligent action" (Newell and Simon 1976, p. 116), and, in doing so, they relied on their own definition of a physical symbol system. Newell and Simon's general project is quite familiar, but the details of their definition of a physical symbol system are, on the other hand, less well known, and it is worth discussing them here in some detail: they are emphatically not the same as Harnad's definitions, and, because of this, they are an excellent antidote to the tendency to regard Harnad's definitions as obvious and, hence, unavoidable.

Firstly, physical symbol systems are *systematic*: symbols—or, rather, symbol *tokens*—have types, and these types are capable of being multiply instantiated. These symbol tokens can be components of larger entities, namely *expressions*. Newell and Simon do not specify the componenthood relation very precisely: they merely say that the symbol tokens making up an expression should be "related in some physical way (such as one token being next to another)" (Newell and Simon 1976, p. 116).

Secondly, physical symbol systems are *physical*. They are physical not merely in the sense that they are collections of physically embodied syntax, but physical in the sense that a physical symbol system incorporates, by its definition, a causal component: "the system also contains a collection of processes that operate on expressions to produce other expressions: processes of creation, modification, reproduction and destruction". (Newell and Simon 1976, p. 116)

This concludes their definition of a physical symbol system. On this basis, they make two further definitions (Newell and Simon 1976, p. 116):

> Designation An expression designates an object if, given the expression, the system can either affect the object itself or behave in ways dependent on the object
>
> Interpretation The system can interpret an expression if the expression designates a process and if, given the expression, the system can carry out the process

Finally, they list five requirements which physical symbol systems should satisfy: of these, it is the fourth which will concern us:

> Expressions are stable; once created they will continue to exist until explicitly modified or deleted. (Newell and Simon 1976, p. 116)

These definitions, although they are well motivated by Newell and Simon's work, are somewhat at odds with mainstream philosophy: for example, their notion of designation is entirely causal—one should contrast here (Floridi and Taddeo 2005), "[u]sually, the symbols constituting a symbolic system neither resemble nor are causally linked to their corresponding meanings"—and their notion of interpretation seems to be at home in a rather extreme pragmatism, according to which a subjects' grasp of the denotation of an expression can only be manifested by actions on the

part of that subject which affect the object designated. It is a sort of action-directed version of what are known as *causal theories of content* (Adams and Aizawa 2010).

Because of these definitions, it is not entirely obvious that the symbol grounding problem even *arises* for Newell and Simon. Designation and interpretation are defined in causal terms, so that, if a symbol has a designation, it is causally related to what it designates, and, if it has an intpretation, it designates a process which it can carry out: and these conditions look very like grounding. There is no room, in Newell and Simon's framework, for the sort of slack between semantic relations, such as are given by the model theory of first order logic, and the more full-bloodedly causal relations which are what a solution to the symbol grounding problem should provide.

Newell and Simon: Normativity

As well as Newell and Simon's causal definitions of semantic concepts, there are other, more technical, issues at work here, which are illuminatingly revealed by their requirement that expressions should be stable. It is doubtful whether this is compatible with physics: what it is saying is that an expression will—unless it is acted on by the processes of modification or deletion which are explicitly given with the physical symbol system in question—last for ever. So it is ruling out cosmic ray impacts, power failures, disk head crashes, spillages of coffee over equipment, and also quantum mechanical tunnelling. (It is the last of these which makes unwanted deletion eventually inevitable, rather than simply unfortunate.) All of these will cause particular symbol instances not to exist, and none of these (except for extremely eccentrically defined physical symbol systems) is a designated process of explicit modification or deletion.

Newell and Simon, it would seem, are tacitly defining physical systems, not as particular types of physical objects, but as particular types of *mechanisms*. Unlike physical objects, which are described merely by physical constituency and causal relations, mechanisms allow a distinction between intact and broken states, and also between normal and abnormal evolutions of those states.[2] Consequently, their definitions are to be read, not as applying to all causal processes applying to symbol tokens, but only to *normal* processes.

Similar restrictions apply to their definition of designation: a symbol, according to their definition, designates an object if it can affect that object. But, if executing a certain command on a laptop is so computationally intensive that it causes a laptop to overheat and thus burns the user's legs, and does so robustly and reliably, this should not mean that the command designates the user's legs: consequently, if we are to define a designation relation in this way, we must have a notion of non-deviant causal chains linking symbols to what they represent. Again, this notion is

[2] We use the term mechanism in a rather loose sense: it will also include biological organisms, for which these distinctions are also available, and for which we can find a developed theory in (Millikan 1984). On the other hand, Newell and Simon do not seem to have functionalism in mind, although they do speak, in a rather unspecified way, of the interpretations being determined by the "mutual relation" of symbol tokens. We shall, accordingly, not insist on functionalism at this stage.

not given to us merely by the physical concept of causality, but is part of our everyday concept of mechanism.

This can be illuminatingly connected with recent work in the philosophy of mind. Sellars writes, of the concept of *knowing*, that to describe a mental episode as an episode of knowing is "not [to] give an empirical description of that episode … [but to place] it in the logical space of reasons, of justifying or being able to justify what one says". (Sellars 1997, §36) In the same way, when we apply concepts such as Newell and Simon's 'designation' to episodes in the physical history of the matter contained in a particular spatial region, we are not describing this region and the matter contained in it merely in physical terms: we are regarding it as a mechanism, and thereby placing it in a particular logical space. We will, in the course of this article, make this concept of a mechanism more precise: what we should notice here is that this new explanatory framework brings with it a normative dimension. Causal processes of the mechanism can be normal, or they can be abnormal: states of the mechanism can be normal, or defective.

Millikan

Millikan (1984) has an evolutionarily grounded naturalistic account of linguistic communication that can, from our perspective, be regarded as filling in some of the gaps in Newell and Simon's account. There is a rather intricate series of definitions, which we can summarise as follows.

Firstly, her definitions are functional and evolutionary: the mechanisms involved in, for example, language are possessed by organisms because they perform certain functions (that is, the performance of these functions gave certain advantages to ancestors of an organism, and because of this the inheritance of mechanisms performing these functions has become evolutionarily established) (Millikan 1984, pp. 28ff) Such functions are called *proper functions* (the official definition is rather complex because, for example, functions are typically exercised by organs, whereas evolution proceeds on the level of organisms, so one has to allow for organs having proper functions because of the evolutionary advantage enjoyed by the organisms that possess organs with those functions).

Secondly, it is important for language that certain functions are not merely performed, but performed in a standard way (we must use roughly the same linguistic symbols as our audience, for example): this leads to a definition of *stabilising and standardising proper functions*, which are functions which are not merely inherited but inherited in such a way that the mechanisms of inheritance ensure that they are normally performed in a standard way (Millikan 1984, p. 31).

Thirdly, she allows for the fact that functions cause other functions, and that, more generally, functions stand in a complex web of causal and explanatory relations, and thus defines what she calls a *focussed proper function*. Notice first that some functions are *disjunctive*, that is, a mechanism may perform one function on one occasion and another function on another occasion. A focussed proper function is, roughly speaking, the most remote non-disjunctive function that a mechanism performs (Millikan 1984, p. 36): thus, for example, if we are looking for "the" proper function of the linguistic device of the imperative mood, we look for

> some characteristic contribution made by the imperative mood that can be understood as useful to both speaker and hearer. Or, if there are many such functions …, we look for a focussed function. (Millikan 1984, p. 56)

We should note, however, that Millikan does admit that "focussed proper function is a somewhat vague term" (Millikan 1984, p. 35).

On this basis, Millikan can describe the proper functions of a number of linguistic devices: for example, "the focussed stabilising function of the indicative mood is … the production of a *true* belief" (Millikan 1984, p. 59), and denotative and referential terms "function properly when they precipitate acts of identification of the variants in the world to which they correspond" (Millikan 1984, p. 71).

We should note one final thing. We can, using these concepts, give a relation between some mental items—such as concepts—and their referents; the process is complex, and is described in (Millikan 1984, Ch. 6). But these relations are generally not fully present to consciousness: as Millikan says, "intentionality is grounded in external natural relations". (Millikan 1984, p. 93) That is, the relation is constituted by things in the world (in Millikan's case, a very large quantity of things in the world, including the evolutionary history of the organisms in question): consequently, there is no reason to expect introspection to deliver the final truth about this relation. And Millikan gives examples (Millikan 1984, Ch. 8) which show that this theoretical possibility can easily obtain.

Where does this get us? There are, of course, modifications to be made to the account so far if we want to apply it to computers: these are things which have been designed and manufactured, rather than evolved. Nevertheless, we can talk, in analogous terms, of the proper functions of artefacts such as computers, and this introduces a normative dimension into our analysis: in Millikan's terms, we have a distinction between "what is Normal or proper" and "what is merely actual" (Millikan 1984, p. 86). So we can solve some of the puzzles we found with Newell and Simon: for example, the case of a computation heating the user's legs could be dealt with by arguing that the heating (although it may be very reliable, or indeed necessarily, produced by a computation of a certain sort, was nevertheless not a proper function of the computer, whose proper function was to execute computations, and, in context, was to execute *that* computation).

So Millikan's account seems to have certain advantages: it is naturalistic, it is grounded (and fairly plausibly grounded) in the theory of evolution, and it is considerably more subtle than Newell and Simon's. Millikan refers, in terms reminiscent of Sellars, to her position as an "attack upon 'the given'" (Millikan 1984, p. 92), and this seems very apposite.

However, Millikan's solution rules out at least some candidate solutions to the grounding problem: it is incompatible with solutions which rely on characteristics of mental contents, or of mental processes, to solve the problem. And it is still susceptible to our previous worry that, maybe, it solves the grounding problem too easily: if, as she argues (Millikan 1984, p. 93), Millikan's symbols essentially involve their entire evolutionary history, then they necessarily have enough of a casual context to be grounded anyway.

Clark

Clark (2008, 2001) has written extensively on the physical limits of the mind, on thought and embodiment, and on cognitive science. His work is richly textured and complex: we will, for clarity, confine ourselves to single example of his, from (Clark 2001, Ch. 7).

He considers a centrifugal governer for a steam engine: this is a mechanical device which, by using appropriate feedback, ensures that a steam engine runs at constant speed. He claims, following (van Gelder 1995), that

> constitutes a control system that is noncomputational, nonrepresentational, and that simply cries out for dynamic analysis and understanding. In particular, only a dynamic analysis can explain the complex, yet effective, relationship that is obtained between the arm angle and the engine speed. (Clark 2001, p. 127)

The moral of this example and others like it is, claims Clark, that

> body and world matter [to the mind] not simply because they provide an arena for useful action and a sensitive perceptual front-end, but because neural, bodily, and environmental elements are intimately intermingled courtesy of processes of reciprocal causation that cris-cross intuitive boundaries. … [T]he traditional tools of computational and representational analysis cannot do justice to such a complex interactive process, and … the mathematical and topological resources of dynamic systems are to be preferred. (Clark 2001, p. 128)

In other words, there is no symbol grounding problem for systems such as these because there are, properly speaking, no symbols: there is only a dynamical system, which can only be described *qua* dynamical system in a context which encompasses both the steam engine and the control mechanism. In particular, one cannot dissect it in such a way as to reveal any symbols with isolated syntactic natures. This is a position with some history, particularly in robotics: for example, Brooks (1990, 1991) argues that, for agents which directly experience, and interact with, the outside world (*embodied, situated* agents in his terminology), all that is necessary for grounding are appropriate sensorimotor couplings, rather than explicit representations grounded in some way. Brooks (1991) calls this the *physical grounding hypothesis*: it relies on causal conceptions which are very similar to those of Newell and Simon.

There are also similar positions in the philosophy of mind: in this area there has been a sustained critique of the dichotomy between entities inside, and outside, of the mind; McDowell (1998) has a critique, based on Sellars, of the Cartesian idea that the inhabitants of the inner realm (thoughts, sense data, and such) have natures which can be defined solely in terms of the relations which such entities have to each other. Such critiques have recently attracted a great deal of attention in the philosophical community (Taylor 2002; McCulloch 2002): those with longer memories will, of course, remember the phenomenological tradition (Merleau-Ponty 1945).

Assessment

The Nature of Symbols

Many of the fault lines in the discussion above are to do with the nature of mental or computational contents. Here we use the term 'nature' in the strong sense: that is, mental and computational contents are supposed to belong to some sort of natural kind, and their nature consists of their essential properties *qua* instances of that kind. We use the term 'mental or computational contents'—'contents' for short—to mean whatever mental or computational entities are supposed to play a role in computation or mental life: the term 'symbols' is question-begging, since Clark, for example, more or less denies their existence in a large number of interesting cases.

Thus, Harnad—and many others—think that mental contents are, by nature, members of syntactic physical systems, located inside computational devices: Millikan, by contrast, thinks that the nature of mental contents is to be naturally evolved self-propagating and self-standarding mechanisms located in organisms. Clark, on the other hand, thinks that mental contents are components of dynamical systems, but they are not necessarily located inside organisms.

There is, then, clearly, a spectrum of views, depending on what sort of properties make up the nature of mental contents. If one follows Harnad and other functionalists, then mental contents are specified by syntactic properties (suitably instantiated): if, on the other hand, one follows Millikan, then they are specified by their role in the evolutionary propagation of organisms. In Harnad's case, what is difficult to recover is the intentionality of mental contents: in Millikan's case, then intentionality is (relatively) easy, since (roughly speaking) the properties of mental contents which are propagated by evolution are their intentional properties.

Intrinsic and Extrinsic

There is another issue at work, however. What Harnad is trying to achieve is to show how a syntactic system can be a *grounded* physical symbol system: that is, a symbol system which has, as it were, its "own semantics". As Ziemke (1997) puts it, the function and internal mechanism of the artefactual agent should "be made *intrinsic* to the artefact itself" (Ziemke 1997, p. 87), or, in Harnad's words,

> How can the semantic interpretation of a formal symbol system be made intrinsic to the system, rather than just parasitic on the meanings in our heads? How can the meanings of the meaningless symbol tokens, manipulated solely on the basis of their (arbitrary) shapes, be grounded in anything but other meaningless symbols? (Harnad 1990)

Thus, what Harnad wants is an explanation of how symbol systems, *qua* symbol systems, can also be grounded. Similarly, one of the main tasks in Millikan's theory is to show how mental contents, *qua* evolutionary propagated mechanisms, can also have syntactic properties. So for both of these authors, the nature of their symbols gives a notion of what is intrinsic to that symbol system and what is extrinsic: and

their notions of grounding (Harnad's notion of the grounding that his symbols don't have but should, and Millikan's notion of the grounding that her symbols do have) ground the symbols in the system by circumstances intrinsic to that system. And, as Example 3 shows, we have to get the right notion of intrinsic: the symbols in this problematic example cheated by including their referents as well as their symbolic components, and thus making grounding too easy. The symbols in question were defined purely by their syntactic interaction, and their referents were, from this point of view, not intrinsic: this arbitrary inclusion of non-intrinsic items seems to be what makes this example suspect.

So there seem to be two key analytical issues at work here. One is the conception of the nature of symbols that our various authors work with; the other is whether there is a distinction between intrinsic and extrinsic, and, if so, how it is to be made. The answers to these questions determine the various positions of our authors on the symbol grounding problem.

Practical Computers

We now turn to the analysis of how these problems look practically, in computer science. Computers are systems which (in some respects) we know very well: we have designed and constructed them, and we have also designed the software which runs on them and the programming languages in which such software is written. Philosophical work on these themes in the philosophy of mind is necessarily written in ignorance of the underlying physical systems: we simply do not know enough about how brains work. Computer science gives us a substantial body of knowledge which we can, if we wish, bring to bear on problems like this, and this can result in a certain amount of illumination, not just for the philosophy of computer science, but also, if we are lucky, for the philosophy of mind.

It is worth remarking here on a characteristic of Newell and Simon's approach: they insist that it is an *empirical* conjecture. By this, they do not mean that the enquiry is to be conducted using the methods of the physical sciences: rather, that it should be conducted by observing the practice of computer programmers and other symbol system constructors and manipulators. This is an approach which has a great deal to commend it: as we argue in (White 2004), the structure of programming languages is a sort of mirror-image of the structure of human languages, and the same probably obtains between the philosophy of mind and corresponding areas in the philosophy of computer science.

This approach will illuminate both of the analytical issues which we found in our discussion of the symbol grounding problem. It will illuminate the distinction between intrinsic and extrinsic, because we know a good deal of the physical process that happen in computers and how those processes issue in computation. It will also illuminate the nature of symbols, because, according to our position in (White 2004), symbols in computer programming languages have natures which are given by the semantics of those languages: for many languages, such a semantics is actually written down in a formal language specification. So we will actually have a supply of symbol systems with interesting properties.

Since we are interested in the difference between intrinsic and extrinsic, our main example will be that of a *serial port* (Lawyer 2008). This is one of the ways in which a computer is connected to the outside world: it will allow us to examine the relation between the inside and outside of a computer. It is a comparatively old piece of hardware: many modern computers do not have them (their place has been taken by USB ports). However, serial ports exhibit the phenomena that we will be concerned with, while still being relatively simple (although nevertheless much more complex than one would expect). More modern systems would be more baroque and would require more—and significantly more impenetrable—explanation.

Serial Ports

A serial port links the exterior and the interior of a computer (at least in the naive geometrical sense of 'inside' and 'outside': it is a piece of hardware located inside the computer's box and protruding slightly outside of it, and by means of which the computer can be connected to external devices). Modems used to be connected to computers using them, as did printers and mice. Signals can thus be transmitted to, and received from, external devices. (We are here, in order to avoid begging any conceptual questions, using language which is as physical as possible).

There is, however, another view of serial ports: that of the programmer or of the user. If a program uses devices connected over a serial port, then a programmer must write code which interacts with that device using the serial port. The code will use primitives in some programming language or other, and the logical form of those primitives will give another, more phenomenological, view of what serial ports do. This language—or at least the documentation explaining the programming primitives—will use terms like 'inside' or 'outside', or terms (like, for example, 'transmit' and 'receive'), which presuppose some sort of inside/outside distinction. The physicalist, hardware-oriented inside/outside distinction will, in general, differ from the programmer's "phenomenological" inside/outside distinction: and this difference will drive the argument of this paper. To put this in the form of a question: is a serial port merely a physical entity, or does it play a more conceptually loaded role? To echo Sellars, what logical space do our programming concepts put it in?

The Hardware

A serial port appears on the outside of a computer as a socket, usually with nine pins. It is, as the name implies, serial—that is, data travels along the connection in individual bits, one after the other. (The fact that there are nine pins in the socket does not affect this: only one pair of pins, and the wires connected to them, carry data. The others are control pins.)

Behind the socket lies a chip (or, in modern hardware, part of another chip) called a UART.[3] The UART performs some functions that are necessary for any connection between the interior of a computer (within which data travels along a

[3] Universal Asynchronous Receiver-Transmitter: see Lawyer (2008, §1.6, §19)

bus, with up to 128 bits in parallel) and a serial connection: data arriving on the internal bus is serialised, that is, transmitted one bit after another, and it is transmitted using a waveform optimised to be proof against the vicissitudes of propagation along a wire open to the external world. The UART does this unattended: but it also communicates with sender and receiver (that is, the central processing unit of the computer and the external device, or vice versa) to ensure that these pieces of hardware are ready to transmit or receive.

But the UART also performs another function, which is not analytically necessary for the task described above. It performs *buffering*: that is, it accumulates data before notifying more central components that it has received it, and only performs such notification when the buffer is full (for communications in the other direction the same thing happens, *mutatis mutandis*: it notifies the CPU when it has transmitted data). It does this because notifying the CPU is an expensive operation—that is, takes a long time for the CPU to deal with it, by comparison with the its normal operations—and so it is important that it should not happen too often.

So, we could argue, buffering improves performance, and the more buffering one does—that is, the larger the capacity of the buffer—the greater the gain is. However, the UART only buffers at most sixteen bits at a time. Why should this be? Well, the UART has a dual responsibility—both to the CPU and to the external device—and the external device may not always be capable of handling the data sent to it (it may, for example, be a modem transmitting over an unreliable line, and external conditions may be bad). So the external device must be capable of turning off the flow when necessary: if it is to be capable of doing this, then the UART must not cache all too much data, because whatever it caches it transmits.

As well as communicating with the external device, the UART must communicate with the CPU of its host computer, either to tell the CPU that it is ready to send data, or that it has stopped sending data, or that it has received data: it does this by sending the CPU signals, called *interrupts*. Dealing with interrupts is costly: it takes a disproportionate amount of CPU time compared with the CPU's normal operations. So, computers generally set up a fairly large buffer (8 kilobytes or so) in main memory, and use this to communicate with the serial port. The CPU's dealings with the serial port will then happen comparatively infrequently, and will be, as it were, by proxy: all the CPU will do is to fill and empty the large buffer in main memory, and the UART—or other pieces of hardware—will see to it that the main memory buffer is appropriately emptied and filled, respectively. (Lawyer 2008, §3.3)

On the other end of the connection there is a similar setup. Most devices which are attached to a serial port will themselves perform some buffering, and so, when data is transmitted, there will be a fairly complex pattern of interaction involving the movement of data between these three buffers (the one in the remote device, the one in the UART, and the one in main memory), together with the interchange of control symbols between all three devices. (Lawyer 2008, §4.6)

Programming the Hardware

Before we talk about the programmers' interface to the serial port, we should clarify a few issues to do with programming in general. Firstly, programming is done in a

particular language: either low-level languages, such as assembler or C, or high-level languages such as Java. Secondly, a programming language is generally specified by a semantics (White 2004), in the sense of 'semantics' used by computer scientists: that is, we should have some more or less abstract definition of the internal state of a computer, and, for each of the primitives of the language we should describe what its effect is on the state of the computer that it is being executed on. Such a semantics, then, interprets the primitives of the computer language as (somewhat abstract) actions on the internal state (abstractly conceived) of the computer. And the difference between low-level and high-level computer languages lies, by and large, in the degree of abstraction that is applied: a high-level language like Java will allow only talk of data structures and operations on them, whereas C, a lower-level language, allows, in addition, talk of memory addresses (abstractly conceived) and regions of memory (again, abstractly conceived). But there are aspects of the hardware which even C allows no access to: for example, the CPU will copy data from memory into what are called *registers* in the CPU (Wikipedia 2009a), and will operate on the contents of those registers rather than data in memory: C allows no access to this, but instead treats all operations as if they operated directly on data in memory. And even assembly code—the supposedly lowest level of coding—is still somewhat abstract: it is what is called *relocatable* (Wikipedia 2009b)—that is, it should be capable of functioning without regard to the area in memory in which it and its data are located—and, consequently, it operates with abstract representations of memory addresses rather than memory addresses themselves. So, when we talk about the structure of computer languages, we are always talking about some abstract representation of the hardware of a computer.

We need to distinguish this computer scientists' semantics from the real-world semantics which Harnad wishes to be the target of the symbol grounding problem: so, from now on, we will describe the computer scientists' semantics as *formal* semantics, whereas real-world semantics will be referred to simply as semantics. Formal semantics, in the sense in which we will be using it, talks about the relation between symbols in a computer language and their implementations, and, to this extent, what it talks about is the *nature* of those symbols.

Serial ports, then, can be programmed in a variety of styles, ranging from very low-level to high-level. The low level techniques, although they can be quite elaborate, can also be very simple: one finds out where in main memory the large buffer is located, and writes directly to it (Various Authors 2009, §5). A low-level program will, generally, be considerably more elaborate than this (it will involve such things as interrupt handling), but what it will concretely do is write to the main memory buffer for the serial port. So, for such a program, the external context begins with the main memory buffer: once the program has placed its data there, it is no longer its responsibility.

The high-level interface, on the other hand, is much more abstract. The usual approach to input and output in general (not merely to or from serial ports) is based on six operations (open, read, write, close, seek, ioctl) (Comer 2005, p. 267). Read and write are fairly self-explanatory: open and close prepare a device for reading and shut it down. Seek (not available for all devices) moves to a new location on the

device, whereas ioctl carries out miscellaneous control functions (in the case of a modem connected to a serial port, for example, it may change the connection speed of the modem).

What is noticeable about the high-level interface is that (as well as being very abstract) is that these commands tend to apply distally, that is, to the device which is attached to a computer, rather than applying to acts of transmission through the proximal end of a link to that device. This is not entirely accidental: most transmission between computers and external devices involves some sort of flow control and buffering at both ends, so that, if one is opening or closing a link to a device, then coordinated action at both ends of the link is necessary (even though the coordinated action may be initiated from the computer). Consequently, even apparently simple acts of transmission may be surprisingly complex, and may involve checks and acknowledgements in both directions. So a high-level programming interface to these processes will have semantics that take in both the computer and the device that it is connected to.

This means that the boundary between the computer and the outside world is surprisingly fluid. The low-level programmer will view the outside world as beginning in the main memory buffer, because, once data is placed there, it will be handled by processes which are, strictly speaking, no longer under the control of the cpu. On the other hand, the high level programmer will have a viewpoint that encompasses both the computer and the external device: high level programs will initiate actions whose semantics, and whose success and failure conditions, will involve not merely the state of the computer but also the external device. This is reflected in the structure of the programming constructs used: for example, high level programming languages typically have methods for reporting error conditions, and these error conditions—usually known as exceptions[4]—will typically be distinguished between those which occur locally and those which occur remotely.[5]

The exception mechanism in languages like Java is interesting for several reasons. Firstly, there are explicit mechanisms for dealing with these error conditions, and they are explicitly given a semantics. That is, if an error occurs, there are recovery mechanisms, and these mechanisms undo all of the changes made by the action which caused the error to occur, except those—such as input and output—which have had an effect on the outside world which cannot be undone. Implicitly, then, these recovery mechanisms are committed to an internal/external distinction, but one defined in terms of the ability to undo changes rather than in terms of the interior and exterior of a device.

Secondly, exceptions introduce an element of normativity into the language: actions can either succeed or fail, that is, they can terminate without causing an

[4] Pedantically speaking, one should point out that Java actually divides what we have called exceptions into three subsets: Exceptions, Errors, and RuntimeExceptions. Errors are those which are so serious that there is no point in trying to recover from them: Exceptions and RuntimeExceptions are as we have described (but differ in whether the provision of code to handle them is mandatory or not).

[5] For example, the classes in java which report input-output errors (namely, the subclasses of java.io.IOException) include classes such as MalformedURLException, which typically arises locally, as well as RemoteException, which is, of course, remote, as well as ProtocolException, which is something arising out of the interaction between local and remote machines.

exception or they can cause an exception. Exceptions are typically caused by events over which the programmer has no control (and which hence are outside the intrinsic context, defined in the agent-based way). Such events are the usual misfortunes which can occur in the tussle between the external world and computers: network connections being physically severed or falling a victim to congestion, hard disc failures, unexpected user behaviour, and so on.

Assessment

Let us now consider this from the point of view of the nature of symbols. Low level languages has a formal semantics which refers to purely internal states of the machine: memory locations and their contents. The nature of these symbols, then, can be specified in terms of internal configurations of the machine and their relations to each other.

But, precisely because of this, the formal semantics of these languages can be profoundly unilluminating. For example, consider writing data to a serial port. In a low level programming language, as we have seen, we can do this by writing data to a particular location in memory. So we have a low level instruction (writing data to memory) whose real-world semantics is quite different from its overt form: and the correspondence between overt form and real-world semantics could only be recovered by using contextual data. This contextual data could not be recovered from the instruction itself, but only from the previous instructions of the program, details of the operating system, and so on; without this data, it would be impossible to distinguish the meaning of this instruction from the meaning of other instructions, such as those which changed the value of variables.

By contrast, high level languages have symbols which are more finely differentiated, which can express notions such as input and output, and which have notions of success or failure which are appropriate for such operations. The formal semantics of high level languages specifies this behaviour, so these symbols have a nature which depends, not just on the relations between the symbols and other symbols inside the host machine, but also on the relation between the host machine and remote devices. To this extent, then, the nature of the symbols in high level languages is not purely syntactic, where 'syntactic' is here construed in terms of relations between symbols inside the host machine.

The concept of what is purely syntactic depends, as we have seen, on a notion of what is intrinsic and what is extrinsic to a particular device. Just so here: low level languages can be construed as syntactic with respect to the device contained inside the case of the computer that they are implemented on (or, rather, with respect to a particular idealisation of the the device in question). High level languages can still be construed as syntactic, but with regard to an idealised device which may be larger in geographical extent than merely the inside of the computer's case: it may in general involve external devices and the network connecting them.

We can, of course, talk about the implementation of high-level languages from the point of view of what happens in the interior of the host computer. But we do so at a price: there are notions of success or failure which can be expressed using the symbols of high level languages, but whose formal properties cannot be predicted

using merely relations between symbols which inhabit that computer. We can make these symbols purely syntactic, but only if we talk about idealised machines with a larger geographical extent. The relation between the low-level and the high-level view involves an increase in the expressive power of the languages involved: the high level languages can talk about concepts of success or failure which, in general, involve remote devices as well as the host computer.

Cartesianism and the Nature of Symbols

We have now reached territory which looks very like the Cartesian conception of mind: we have symbols, we have internal or external domains with respect to which those symbols are defined, and we want to know how those symbols are related to the external domains. McDowell (1998) has an account of the genesis of this Cartesian conception of the mind: there is, as we shall see, there is a good deal of mutual illumination to be derived from comparing the two Cartesianisms—a Cartesianism of mind and a Cartesianism of machines—with each other.

McDowell writes, of what might have motivated Descartes, that

> [i]t seems scarcely more than common sense that a science of the way that organisms relate to their environment should look for states of the organisms whose intrinsic nature can be described independently of the environment; this would allow explanations of the presence of such states in terms of the environment's impact, and explanations of interventions in the world in terms of the causal influence of such states, to fit into a kind of explanation whose enormous power to make the world intelligible was becoming clear with the rise of modern science. (McDowell 1998, §6)

This has, as a consequence, that the intrinsic nature of states of the organism, and, in particular, of mental states is a matter of their relation to each other: we can describe this without any mention of their relation to the external world. The relation between mental states and the world can then be dealt with after the essential nature of those states has been elucidated. (McDowell 1998, §8)

Now the idea of decomposing the world and analysing the parts separately is, clearly, attractive, and attractive for good reasons, and so this sort of conception has been as important in computer science as it has been in the philosophy of mind: the vital role that modularity has played in such things as programming language design testifies to that. However, what this purely methodological principle does *not* licence is the idea that there should be one single, fixed boundary between inside and outside: computer science has a variety of sorts of theory, engineering has a variety of design principles, and all of these might draw their boundaries slightly differently from each other. The motives for drawing these sorts of boundaries are, by and large, pragmatic: one draws boundaries where they are most convenient for one's purposes.[6]

[6] Still less, of course, are we licensed to suppose that there should be some sort of metaphysical distinction between things on either side of the boundary: there is less temptation to do that now than in Descartes' time, but attempts are still made.

As McDowell puts it, "[w]e are compelled to picture the inner and outer realms as interpenetrating" (McDowell 1998, §5).

These considerations become especially pertinent if one is tempted to believe that the nature of symbols is exhausted by their relations with other objects on the inside, and that grounding these symbols will involve setting up relations between these symbols and external objects, while leaving the nature of the symbols unchanged. Now, as we have seen, the practice of serial port programming gives us two levels—the low level and the high level—with the latter being more grounded than the former. But the symbols are different in both cases, and there is no easy correspondence from one to the other.

And thinking of symbols as the inscriptions on the tape of a Turing machine, merely on its own, no help. Each level of abstraction will describe the physical workings of the computer in terms of some mathematical model of a computational device (which, maybe following some further coding, one could represent as a Turing machine, but merely the idea of a Turing machine is profoundly unilluminating about the nature of the symbols in question).

Furthermore, we have seen that there are high-level languages, such as Java, whose formal semantics is more closely aligned with their real-world semantics than is the case for low-level languages (this is, after all, what makes them high-level languages). So, in particular, the formal semantics of such a Java program could not be recovered, instruction by instruction, from a corresponding program in a low-level language.[7] Consequently, if we construct Turing machines to reflect the symbol manipulation which is going on in both languages, there will be no state by state, symbol by symbol correspondence between them.

Spontaneity and Receptivity

There is, however, a very large difference between computers and minds, and it is this. First some terminology. McDowell follows Sellars in rejecting the Cartesian architecture of epistemology: namely, the Cartesian idea that there was a category of data, which was internal and unproblematically given, and out of which our knowledge was to be reconstructed. If one rejects this, one should be able to say how the theory of knowledge might be otherwise structured.

McDowell's candidate for an alternative structure is based on the Kantian distinction between spontaneity and receptivity. We may picture a reasoning subject engaged in trying to make sense of the world: there will be episodes in which the world impinges on the subject (those are episodes of *receptivity*), and there will be episodes in which the subject can engage in free reasoning (those are episodes of *spontaneity*). (McDowell 1996); cf. Stroud (2002) Using this distinction, one can hope to build a theory of knowledge which does not use a category of internal, uproblematically given items of knowledge. This contrast, unlike the Cartesian one, seems more adapted to a conception in which inner and outer interpenetrate: the

[7] Contextually defined mappings in both directions are, of course, possible, and there is software to do it: compilers translate from high-level language to assembly code, and decompilers do the reverse (not always very successfuly).

distinction is not set up in purely geometrical terms, but is defined causally, so that it can still apply to a world in which the boundary between inner and outer fluctuates.

However, the scope of free reasoning, in McDowell's terms, is much larger than ours: it involves concept building, critical assessment of one's own reasoning, and the like. Computers as we understand them can hardly do any of this: they are not reflective in the Kantian sense which McDowell has in mind. And the ascent from low-level to high-level languages, significant though it is, is something which is performed by humans (computer science theorists, language designers, and the like). Computers, for their own part, still have something which is recognisably like a distinction between spontaneity and receptivity: this comes about because we can distinguish, among the causal processes in our computer, between those which proceed normally and those which proceed abnormally, and because, in addition, we have programming language constructs (namely exceptions) which are concerned with detecting abnormal termination and recovering from it. So we do genuinely have a related set of concepts (and one which is quite similar to Taylor's account of coping in (Taylor 2002), which is based on (Heidegger 1979)).

This world—the conceptual world of the computer programmer—is not, then, a world of bare symbols: and this, in turn, gives rise to problems for the symbol grounding problem. We shall turn to those problems in the next section.

Conclusions

Let us start with an obvious question: how do we know that it is possible to solve the symbol grounding problem? The history of even theoretical attacks on the Symbol Grounding Problem is not encouraging (Floridi and Taddeo 2005), and, as many commentators—for example, Ziemke (1997)—have pointed out, it is not obvious that this enterprise should succeed. A key difficulty is that we are supposed to have items which are capable of being defined purely in terms of their relations to each other (their *intrinsic*) relations, as Ziemke describes them), but which are also capable of bearing semantic relations to the external world. As McCulloch puts it, the "real problem" with supposing that we could have a symbol system defined solely by its syntax "comes from supposing that world-directed thinking can exist whether or not it has a world of the appropriate type to be directed at". (McCulloch 2002, p. 126) And, once we grant that, we are tempted to identify the realm of such states with an inner world,

> an autonomous realm, transparent to the introspective awareness of its subject; the access of subjectivity to the rest of the world becomes correspondingly problematic, in a way that has familiar manifestations in the mainstream of post-Cartesian epistemology. (McDowell 1998, §4)

One could, of course, draw pessimistic conclusions from this remarkable prefiguring of Harnad's enterprise: the history of philosophical thought since Descartes hardly gives one grounds for optimism. However, this pessimism only seems inevitable if one buys into the Cartesian enterprise: if one does not, then the position might not be so hopeless.

Let us start with the connectionist move which Harnad and others start with (Harnad 1990 2003; Floridi and Taddeo 2005): given an autonomous agent with sensors, one starts by applying, for example, a neural network to the sensor inputs and detecting invariant features of that input: one then defines symbols to denote these. These symbols should then denote features of the external world.

There are problems here: how one defines suitable criteria for invariance, how one defines the architecture for the neural net without question-begging semantic commitment, and many others (Floridi and Taddeo 2005). However, supposing that one can do so, we should notice that such symbols are grounded in a twofold sense: firstly, because they denote features of the outside world, and secondly, because they *depend on* the features of the outside world which was the training input for the neural network. Both directions of grounding are necessary, and it should be a property of a well-designed and trained neural net that, given suitable inputs, it will detect features which are appropriately similar to the features that it was trained on. But, if we have such a thing, it will have interesting semantics: it will stand for "things like these", where "these" is a demonstrative standing for the training data. So we have an internal symbol that depends, causally, on features of the external world: as McDowell argues, symbols like these are semantically possible, but definitely not Cartesian. (McDowell 1998, §3) Their nature cannot be defined solely by their relations to other inhabitants of the inner world, but must involve their link to the exterior. In our terms, these symbols are not merely syntactic (where 'syntax', here, is defined with respect to the interior of the computer).

Before considering how an autonomous agent can actually set up such neural networks, let us consider the place of action in this picture. We are assuming that the agent has actuators of some sort: supposing that it can perform actions, then it is natural to suppose that it may try to develop a way of judging the success or failure of actions. This could start from quite easy judgements (for example, the success of an eating action could be estimated by the effect on feeling hungry, and then the effect of hunting or gathering actions could be judged by the effect on the subsequent eating actions). Bringing action into the picture has a number of consequences: one is that it allows the agent to calibrate perception against action by evaluating the success or failure of perception-dependent action, and it allows it to develop motion-independent percepts by finding features of the environment which do not vary when it moves about (thus possibly allowing a solution to the problem of finding criteria for invariance). Another, though, is this: if the agent is to do this sort of thing explicitly, then it needs to represent its own actions by means of symbols, and, if it is to be able to represent success or failure, it needs to be able to have means of designating the success or failure of particular actions. Now representing actions means, in linguistic terms, having verbs in one's language, and representing success or failure of actions means having adverbs. So we already have a complexly structured language, and much of the structure comes from the idea of the success or failure of actions: this is an idea which does not seem to carry with it much of an anthropocentric bias. Note also that many of these judgements can be made about actions identified demonstratively, and that such judgements will, again, be internal symbols dependent on external events: they are, again, not definable solely in terms of their relations to other internal symbols.

Note that this sort of construction is not vulnerable to Putnam's objection above: we are not checking that a symbol correctly refers by first finding its referent and then checking that the reference relation actually applies to the pair of symbol and referent. Rather, we have notions of success or failure which have been developed by experience, and which applied to perception and to action. And, provided that these notions of success or failure were correct, we could argue that perceptions or actions, which were judged to be successful by internal, autonomously developed criteria, were grounded.

It would also be a notion of grounding which was actually in accordance with the phenomenology of our perceptions, or, at least, of mine: I am aware of external objects, I can make judgements of the veridicality of this awareness, based on internal criteria, but, on the other hand, I am not aware of symbols inside my brain. I cannot, for example, directly refer to my hippocampus, except by pointing at a brain scan. And grounding, or not, would not be a matter of the symbols inhabiting my head: it would be a matter of the process by which I deployed and developed those symbols and of the judgements of veridicality which I had thus learned to make.

None of this rather glibly formulated chain of hypotheses is intended to deny the severe difficulties, both experimental and mathematical, involved in constructing neural nets (or whatever one uses) that actually do fulfil the roles that are here sketched out. But what it does show is that the merely *conceptual* problems which are raised by Harnad's symbol grounding problem may well have a solution, and that this solution involves dealing with the Cartesianism involved in their formulation.

There is a final remark, which is this. Many critiques of Harnad—for example, Ziemke (1997)—diagnose the problem to lie in Harnad's cognitivist assumptions, and seek the remedy in some non-cognitivist paradigm (for example, in what Ziemke calls "enactivism"). Now this is as may be, and Ziemke's enactivism may well have a great deal to offer. However, our analysis seems to show that the problem with Harnad does not like with cognitivism per se, but, rather, with a particular form of cognitivism, namely with a cognitivism which tries to find an unproblematic, transparent inner realm of symbols: other cognitivist assumptions may well be more realistic and offer a greater chance of success in this sort of enterprise.

References

Adams, F., & Aizawa, K. (2009). Why the mind is still in the head. In P. Robbins & M. Aydede (Eds.), *The Cambridge handbook of situated cognition, Chap 5* (pp. 78–95). Cambridge: Cambridge University Press.

Adams, F., & Aizawa, K. (2010). Causal theories of mental content. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (Spring 2010 ed.). http://plato.stanford.edu/archives/spr2010/entries/content-causal/.

Boden, M. A. (1988). *Computer models of mind*. Cambridge: Cambridge University Press.

Brooks, R. (1990). Elephants don't play chess. *Robotics and Autonomous Systems, 6*, 3–15.

Brooks, R. (1991). Intelligence without representation. *Artificial Intelligence Journal, 47*, 139–159.

Carroll, L. (1895). What the tortoise said to Achilles. *Mind 4*, 278–80.

Clark, A. (2001). *Mindware: An introduction to the philosophy of cognitive science*. New York: Oxford University Press.
Clark, A. (2008). *Supersizing the mind: Embodiment, action, and cognitive extension*. Oxford: Oxford University Press.
Comer, D. E. (2005). *Essentials of computer architecture*. Upper Saddle River, NJ: Pearson Prentice Hall.
Floridi, L., & Taddeo, M. (2005). Solving the symbol grounding problem: A critical review of fifteen years of research. *Journal of Experimental and Theoretical Artificial Intelligence, 17*(4), 419–445.
Glanzberg, M. (2009). Truth. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (Spring 2009 ed.). http://plato.stanford.edu/archives/spr2009/entries/truth/.
Harnad, S. (1990). The symbol grounding problem. *Physica, D42*, 335–346. http://cogprints.org/3106/.
Harnad, S. (2003). Symbol-grounding problem. *Encyclopedia of Cognitive Science, LXVII*(4), http://cogprints.org/3018/.
Heidegger, M. (1979). *Sein und Zeit* (15th ed.). Tuebingen: Max Niemeyer.
Lawyer, D. S. (2008). The serial howto. http://tldp.org/HOWTO/Serial-HOWTO.html, available online at The Linux Documentation Project.
McCulloch, G. (2002). Phenomenological externalism. In Smith (pp. 123–139).
McDowell, J. (1996). *Mind and world*. Cambridge, MA: Harvard University Press.
McDowell, J. (1998). Singular thought and the extent of inner space. In *Meaning, knowledge and reality* (pp. 228–259). Cambridge, MA: Harvard University Press. Originally published in (1986) P. Pettit & J. McDowell (Eds.), *Subject, thought and context*. Oxford: Clarendon.
Merleau-Ponty, M. (1945). *Phénoménologie de la Perception*. Paris: Gallimard.
Millikan, R. G. (1984). *Language, thought and other biological categories*. Cambridge, MA: MIT Press.
Newell, A., & Simon, H. A. (1976). Computer science as empirical enquiry: Symbols and search. *Communications of the ACM, 19*(3), 1975 ACM Turing Award Lecture.
Putnam, H. (1978). *Meaning and the moral sciences*. London: Routledge.
Sellars W. (1956) Empiricism and the philosophy of mind. In H. Feigl & M. Scriven (Eds.), *Minnesota studies in the philosophy of science* (Vol. 1, pp. 253–329). Minneapolis: University of Minnesota Press
Sellars, W. (1997). *Empiricism and the philosophy of mind*. Cambridge, MA: Harvard University Press. Reprint of Sellars (1956).
Smith, N. H. (Ed.). (2002). *Reading McDowell: On mind and world*. London: Routledge.
Stroud, B. (2002). Sense-experience and the grounding of thought. In Smith (pp. 79–91).
Taylor, C. (2002). Foundationalism and the inner-outer distinction. In Smith (pp. 106–119).
van Gelder, T. (1995). What might computation be, if not cognition? *Journal of Philosophy, XCII*(7), 343–381.
Various Authors. (2009). Serial programming. Wikibooks, http://en.wikibooks.org/wiki/Programming:Serial_Data_Communications
White, G. G. (2004). The philosophy of programming languages. In L. Floridi (Ed.), *The Blackwell guide to the philosophy of computing and information* (pp. 237–247). Oxford: Blackwell.
Wikipedia. (2009a). Processor register—Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Processor_register&oldid=329987020, [Online; accessed 28-December-2009].
Wikipedia. (2009b). Relocation (computer science)—Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Relocation_(computer_science&oldid=328774935), [Online; accessed 28-December-2009].
Ziemke, T. (1997). Rethinking grounding. In A. Riegler & M. Peschl (Eds.), *Does representation need reality?* (pp. 87–94). Vienna: Austrian Society for Cognitive Science. Proceedings of New Trends in Cognitive Science (NTCS 97): ASoCs Technical Report 91–01.