Harnad, S. (1994) Computation Is Just Interpretable Symbol Manipulation: Cognition Isn't. Special Issue on "What Is Computation" Minds and Machines 4:379-390 [Also appears in French translation in "Penser l'Esprit: Des Sciences de la Cognition a une Philosophie Cognitive," V. Rialle & D. Fisette, Eds. Presses Universite de Grenoble. 1996]

COMPUTATION IS JUST INTERPRETABLE SYMBOL MANIPULATION; COGNITION ISN'T

Stevan Harnad
Department of Psychology
University of Southampton
Highfield, Southampton
SO17 1BJ UNITED KINGDOM
email:harnad@ecs.soton.ac.uk
ftp://cogsci.ecs.soton.ac.uk/pub/harnad/
http://cogsci.ecs.soton.ac.uk/~harnad/
gopher://gopher.princeton.edu/11/.libraries/.pujournals
phone: +44 703 592582
fax: +44 703 594597

ABSTRACT:
Computation is interpretable symbol manipulation. Symbols are objects that are manipulated on the basis of rules operating only on the symbols' shapes , which are arbitrary in relation to what they can be interpreted as meaning. Even if one accepts the Church/Turing Thesis that computation is unique, universal and very near omnipotent, not everything is a computer, because not everything can be given a systematic interpretation; and certainly everything can't be given every systematic interpretation. But even after computers and computation have been successfully distinguished from other kinds of things, mental states will not just be the implementations of the right symbol systems, because of the symbol grounding problem: The interpretation of a symbol system is not intrinsic to the system; it is projected onto it by the interpreter. This is not true of our thoughts. We must accordingly be more than just computers. My guess is that the meanings of our symbols are grounded in the substrate of our robotic capacity to interact with that real world of objects, events and states of affairs that our symbols are systematically interpretable as being about.

KEYWORDS:
Church/Turing Thesis, cognition, computation, consciousness, discrete systems, dynamical systems, implementation-independence, robotics, semantic interpretability, sensorimotor transduction, symbol grounding problem, Turing Machine, Turing Test.

The fathers of modern computational theory (Church, Turing, Goedel, Post, von Neumann) were mathematicians and logicians. They did not mistake themselves for psychologists. It required several more decades for their successors to begin confusing computation with cognition (Fodor 1975; Newell 1980; Pylyshyn 1984; Dietrich 1990)

Before we can begin to sort out this confusion, we must first agree about what computation is (cognition will be a much harder one to agree on, but being all Cartesian cognizers ourselves, let us settle for an ostensive definition on that for now). Let me declare right away that I subscribe to what has come to be called the Church/Turing Thesis (CTT) (Church 1956), which is based on the converging evidence that all independent attempts to formalise what mathematicians mean by a "computation" or an "effective procedure," even when they have looked different on the surface, have turned out to be equivalent (Galton 1990).

Symbol Manipulation Systems and the Church/Turing Thesis

According to all these notational variants on what we might as well call Turing Machines, computation is just the manipulation of symbol tokens on the basis of their shapes (Turing 1990). The usual picture is that of a machine with a tape with symbols written on it; the reading head can move the tape forward or backward or it can halt;

and it can read, write or over-write symbols. The machine is built in such a way that what it does (read, write, move, halt) is determined by what state it is currently in, what other states it can go into, and what symbol it has just read (for example, the state may be something that acts according to the rule "Read, and if a 1 is read, move the tape left and return to this state; if a 0 is read, halt").

With operations as elementary as these, everything that mathematicians, logicians and computer scientists have done so far by means of logical inference, calculation and proof can be done by the machine. I say "so far," because it is still an open question whether people can "compute" things that are not computable in this formal sense: If they could, then CTT would be false. The Thesis is hence not a Theorem, amenable to proof, but an inductive conjecture supported by evidence; yet the evidence is about formal properties, rather than about physical, empirical ones.

There is a natural generalisation of CTT to physical systems (CTTP). According to the CTTP, everything that a discrete physical system can do (or everything that a continuous physical system can do, to as close an approximation as we like) can be done by computation. The CTTP comes in two dosages: A Weak and a Strong CTTP, depending on whether the thesis is that all physical systems are formally equivalent to computers or that they are just computers. Nothing rides on this until we come to the property of implementation-independence, below, and then the distinction will turn out to raise substantive problems for Computationalism (the thesis that cognition is just a form of computation, C=C), a thesis that will also come in a Weak and Strong form. For now, however, we need think of only one, undifferentiated CTTP.

There are people who doubt the truth of CTTP and there are people who doubt the truth of both CTT and CTTP. For such people, computation is either something else that we have not yet succeeded in formalising, or else it is something nonformalisable. Clearly, what is affirmed or denied in higher-order hypotheses linking computation and cognition will be very different in the minds of those who do and do not accept CTT. Those who reject it are much more likely, for example, to deny, Computationalism. It is accordingly important to point out that I do accept the Church-Turing Thesis, in both its formal and physical versions (CTT and CTTP), yet I too will be arguing against C=C.

Systematic Interpretability

There are still two important components of the definition of computation that I have left out. The first of these is controversial and is not usually regarded as part of the definition: Formal computation is clearly symbol manipulation, with the operations on the symbols (read, write, move, halt) being based, as already stated, on the shapes of the symbols. Such shape-based operations are usually called "syntactic" to contrast them with "semantic" operations, which would be based on the meanings of symbols, rather than just their shapes.

Meaning does not enter into the definition of formal computation. Recall that when you were first taught arithmetic (or algebra, or set theory) formally , the symbols (in arithmetic, "0," "1," "+," "=." etc.) were introduced as "primitive, undefined terms" -- although of course you already knew what they meant intuitively and in practice -- and then you were given rules for combining them into well-formed formulas and for deriving further well-formed formulas from those using the rules of logic and proof. All those rules were, in principle, syntactic. At no time was the meaning of the symbol used to justify what you were allowed to do with it. However, although it was left unmentioned, the whole point of the exercise of learning formal mathematics (or logic, or computer programming) is that all those symbol manipulations are meaningful in some way ("+" really does square with what we mean by adding things together, and "=" really does correspond to what we mean by equality). It was not merely a meaningless syntactic game.

So although it is usually left unstated, it is still a criterial, if not a definitional property of computation that the symbol manipulations must be semantically interpretable -- and not just locally, but globally: All the interpretations of the symbols and manipulations must square systematically with one another, as they do in arithmetic, at the level of the individual symbols, the formulas, and the strings of formulas. It must all make systematic sense, in whole and in part (Fodor & Pylyshyn 1988).

This criterion of semantic interpretability (which has been dubbed the "cryptographers constraint," because it requires that the symbol system should be decodable in a way that makes systematic sense) is not a trivial one to meet: It is easy to pick a bunch of arbitrary symbols and to formulate arbitrary yet systematic syntactic rules for manipulating them, but this does not guarantee that there will be any way to interpret it all so as to make sense (Harnad 1994b).

Computation: Trivial Vs. Nontrivial

The situation is somewhat analogous to the proverbial monkeys at a typewriter: We usually invoke this image to contemplate the likelihood of their typing a passage from Shakespeare by chance. But we could just as well imagine them typing systematically, following countless regularity-generating rules: What is the likelihood that any of those syntactic systems will be decipherable in a way that makes sense? (Note, I am not asking how we would decipher them, but whether any meaningful interpretation is even possible, irrespective of whether we could successfully decipher find it.)

In other words, the set of semantically interpretable formal symbol systems is surely much smaller than the set of formal symbol systems simpliciter, and if generating uninterpretable symbol systems is computation at all, surely it is better described as trivial computation, whereas the kind of computation we are concerned with (whether we are mathemematicians or psychologists), is nontrivial computation: The kind that can be made systematic sense of.

A symbol system with only two states, "0" and "1," respectively interpretable as "Life is like a bagel" and "Life is not like a bagel," is a trivial symbol system. Arithmetic and English are nontrivial symbol systems. Trivial symbol systems have countless arbitrary "duals": You can swap the interpretations of their symbols and still come up with a coherent semantics (e.g., swap bagel and not-bagel above). Nontrivial symbol systems do not in general have coherently interpretable duals, or if they do, they are a few specific formally provable special cases (like the swappability of conjunction/negation and disjunction/negation in the propositional calculus). You cannot arbitrarily swap interpretations in general, in Arithmetic, English or LISP, and still expect the system to be able to bear the weight of a coherent systematic interpretation (Harnad 1994 a).

For example, in English try swapping the interpretations of true vs. false or even red vs. green, not to mention functors like "if" vs. "not": the corpus of English utterances is no longer likely to be coherently interpretable under this arbitrary nonstandard interpretation; to make it so, every symbol's interpretation would have to change in order to adjust systematically for the swap. It is this rigidity and uniqueness of the system with respect to the standard, "intended" interpretation that will, I think, distinguish nontrivial symbol systems from trivial ones. And I suspect that the difference will be an all-or-none one, rather than a matter of degree.

A computer, then, will be the physical implementation of a symbol system -- a dynamical system whose states and state-sequences are the interpretable objects (whereas in a static formal symbol system the objects are, say, just scratches on paper). A Universal Turing Machine is an abstract idealization of the class of implementations of symbol systems; a digital computer is a concrete physical realization. I think a wall, for example, is only the implementation of a trivial computation, and hence if the nontrivial/trivial distinction can be formally worked out, a wall can be excluded from the class of computers (or included only as a trivial computer).

Implementation Independence

So we are interested only in nontrivial computation. That means symbols, manipulated on the basis of their shapes only, but nevertheless amenable to a systematic interpretation. Symbol systems that are meaningful, in other words. We needed this criterion of meaningfulness not only to focus our interest on nontrivial computation, but to point out the second important property of computation that I noted was still to come: The shapes of the symbol tokens must be arbitrary. Arbitrary in relation to what? In relation to what the symbols can be interpreted to mean. For example, in formal Peano arithmetic, the equality symbol "=" is manipulated purely on the basis of its shape, and its shape bears no physical relation to the property of "equality" -- it neither resembles it nor is it causally connected to it. The same is true of "3," which neither resembles "threeness" nor is causally connected to it in the world.

This is a natural place to point out that the symbols of natural language likewise have this property of arbitrariness in relation to what they mean (what Saussure called "l'arbitraire du signe). Whatever else a natural language is, it is surely also a formal symbol system, because it does indeed have a syntax that generates all and only its well-formed utterances, and those utterances are indeed systematically interpretable as meaning what they mean. Another question one might ask about the monkeys at typewriters concerns the likelihood that they would stumble upon syntactic rules that generated all and only symbol strings that were systematically interpretable as utterances in some natural language. Since all formal languages (like arithmetic, predicate calculus and LISP) are subsets of natural language, does the probability of generating a symbol system that is systematically interpretable as a natural language (or the language of thought, Fodor 1975) also lurk behind the definition of formal computation?

We are in danger here of falling into the conflation of computation and cognition that we set out to remedy, so let it be noted that the criterion of interpretability-to-a-cognizer need be no more intrusive in computational theory than the criterion of observability-to-a-cognizer is in quantum theory (and certainly without any of the latter's legacy of paradoxes): We need only note that, among symbolic systems, those that are doing nontrivial computation are rare indeed. We may need a successful human interpretation to prove that a given system is indeed doing nontrivial computation, but that is just an epistemic matter. If, in the eye of God, a potential systematic interpretation exists, then the system is computing, whether or not any Man ever finds that interpretation.

But to return to the arbitrariness of shape that we needed semantics in order to flesh out: It would be trivial to say that every object, event and state of affairs is computational because it can be systematically interpreted as being its own symbolic description: A cat on a mat can be interpreted as meaning a cat on the mat, with the cat being the symbol for cat, the mat for mat, and the spatial juxtaposition of them the symbol for being on. Why is this not computation? Because the shapes of the symbols are not arbitrary in relation to what they are interpretable as meaning, indeed they are precisely what they are interpretable as meaning.

Again, this may seem a trivial case to single out, but it grades into more problematic cases: those in which the "symbols" physically resemble or are causally connected to what they mean. The simple operations of a Turing Machine are not based on any direct connection between the symbols and their meanings; if they were, then that would entail countless special cases and the formal invariants that characterise the interesting and powerful phenomenon of computation would be lost. Another way of characterising the arbitrariness of the shapes of the symbols in a formal symbol system is as "implementation independent": Completely different symbol-shapes could be substituted for the ones used, yet if the system was indeed performing a computation, it would continue to be performing the same computation if the new shapes were manipulated on the basis of the same syntactic rules.

It is this shape-independence of computation that rules out special cases in which the symbols' specific physical shapes or their physical connections to what they mean play a causal role in the "computation." Imagine a machine that could compute 2 + 2 only while it was successively connected to two pairs of things (it could hardly be said to be computing 2 + 2 rather than merely instantiating it); or a machine that could only generate "0" when it had no inputs, "1" when it had one input and so on. The power of computation comes from the fact that neither the notational system for the symbols nor the particulars of the physical composition of the machine are relevant to the computation being performed. A completely different piece of hardware, using a completely different piece of software, might be performing exactly the same formal computation. What matter are the formal properties, not the physical ones. This abstraction from the physical particulars is part of what gives the Universal Turing Machine the power to perform any computation at all.

Turing Indistinguishability

So let us write out long-hand what computation has turned out to be: implementation-independent, systematically interpretable, symbol manipulation. By this means alone, it has turned out to be possible to perform any calculation any mathematician has dreamt of, and also to make machines do many of the other intelligent things that only people (and animals) could do previously. Perhaps it was quite natural to conclude under the circumstances that since (1) we don't know how cognizers cognize, and since (2) computation can do so many of the things only cognizers can do, cognition is just some form of computation (C=C). After all, according to the CTT, computation is unique and apparently all-powerful; and according to the CTTP, whatever physical systems can do, computers can do.

This is the kind of thinking that first made computationalists think they might be psychologists (and made psychologists become computationalists), and, empirically speaking, it made some sense at the time. It even got an extra boost from one of the closet problems of cognition: consciousness (Harnad 1982,1991; Nagel 1974,1986). All physicalist attempts to solve the "mind/body problem," which is a problem we all have in seeing how mental states could be physical states, run aground on one point: Any causal or functional explanation of a physical system is always equally compatible with a mental and a nonmental interpretation [Harnad 1994]; the mental interpretation always seems somehow independent of the physical one, even though they are clearly correlated, because the causality/functionality always looks perfectly capable of managing equally well, in fact indistinguishably (causally/functionally speaking), with or without the mentality.

Consider the difference between a causal/functional system that adaptively avoids tissue damage and another that does the same thing, but feels/avoids pain in so doing: one is tempted to speak of the functional/causal role of the pain , but whatever functional/causal role one assigns it, one could just as well assign to its physical substrate, and then one could just as well subtract the pain and refer only to the functional causal role of its physical substrate. And what is true -- or untrue -- of pain, is true of belief and desire too, indeed of all mental states. They all have this peekaboo relation to their physical substrate.

Well, the implementation-independence of computational states fits well with this peekaboo feature of mental states: If cognition is just a form of computation, it's no wonder we have trouble equating the mental with the physical: We'd have trouble equating the computational with the physical too, because computation is independent of its physical realization (Pylyshyn 1984).

And Computationalism (C=C) also drew some of its compellingness from its semantic interpretability property: Once you find that a system is semantically interpretable, it is hard to see it in a de-interpreted way (as with the "undefined" "+" in formal arithmetic). It's like trying to hear again a foreign language you have already mastered the way it sounded when it was still meaningless to you (not easily done!). Now if the interpretation is mentalistic and not merely semantic, it becomes even more irresistible, forever confirming itself (by definition, in virtue of its systematic semantic interpretability); I have called this the "hermeneutic hall of mirrors" (Harnad 1990 b, c, Hayes et al.1992).

Alan Turing (whom some readers might have been tempted to cite as a counterexample to my earlier suggestion that none of the fathers of computational theory mistook himself for a psychologist) could be seen as inviting us to step into the hermeneutic circle in his advocacy of his celebrated Turing Test (Turing 1964). According to Turing, if there was a person whom we could not tell apart from other people in any way (for years and years, say), then we would have no nonarbitrary basis for concluding that he had no mind if we were informed that he was a machine. This could be construed as encouragment to succumb to the hermeneutic power of semantic interpretability -- particularly because, to rule out biases based on appearance, the Turing Test was formulated as involving a pen-pal, who communicated with symbols only. But one can give Turing credit for more sense than that, and interpret him as arguing that only a fool would presume to try to distinguish between functional indistinguishables (Harnad 1992 b, 1994 b).

The Symbol Grounding Problem

So I see Turing as championing machines in general that have functional capacities indistinguishable from our own, rather than computers and computation in particular. Yet there are those who do construe Turing's Test as support for C=C. They argue: Cognition is computation. Implement the right symbol system -- the one that can pass the penpal test (for a lifetime) -- and you will have implemented a mind. Unfortunately, the proponents of this position must contend with Searle's (1980) celebrated Chinese Room Argument, in which he pointed out that any person could take the place of the penpal computer, implementing exactly the same symbol system, without understanding a word of the penpal correspondence. Since computation is implementation-independent, this is evidence against any understanding on the part of the computer when it is implementing that same symbol system.

But, as I suggested, Searle's Argument does not really impugn Turing Testing (Harnad 1989); it merely impugns the purely symbolic, pen-pal version of the Turing Test, which I have called T2. It leaves the robotic version (T3) -- which requires Turing-indistinguishable symbolic and sensorimotor capacity -- untouched (just as it fails to touch T4: symbolic, sensorimotor and neuromolecular indistinguishability).

So only a pure symbol system is vulnerable to Searle's Argument, and it is not hard to see why. I have dubbed the reason the "Symbol Grounding Problem" (Harnad 1990 a,1993 c): No one knows what cognition is, but we know that cognizers do it. Symbol systems have the remarkable property of being able to compute whatever is computable (that's CTT). In that respect, what they can do and what people can do seems to run along the same channels. But there is one critical respect in which they diverge: A string of symbols such as "2 + 2 = 4" or "the cat is on the mat," generated by a symbol system, is an instance of nontrivial computation if it is systematically interpretable as meaning what "2 + 2 = 4" and "the cat is on the mat" mean. But that meaning, as stated earlier, is not contained in the symbol system. The system is merely syntactic, manipulating meaningless symbols on the basis of shape-based rules, and the shapes of the symbols are arbitrary in relation to what they are interpretable as meaning. Looking for meaning in such a system is analogous to looking for meaning in a Chinese/Chinese dictionary when one does not know any Chinese: All the words are there, fully defined; it is all systematic and coherent. Yet if one looks up an entry, all one finds is a string of meaningless symbols by way of definition, and if one looks up each of the definienda in turn, one just finds more of the same. The search is ungrounded. It is all systematically interpretable to someone who already knows some Chinese , but in and of itself it is meaningless and leads only to an infinite regress.

Now here is the critical divergence point between computation and cognition: I have no idea what my thoughts are, but there is one thing I can say for sure about them: They are thoughts about something, they are meaningful, and they are not about what they are about merely because they are systematically interpretable by you as being about what they are about. They are about them autonomously and directly, without any mediation. The symbol grounding problem is accordingly that of connecting symbols to what they are about without the mediation of an external interpretation (Harnad 1992 d, 1993 a).

One solution that suggests itself is that T2 needs to be grounded in T3: Symbolic capacities have to be grounded in robotic capacities. Many sceptical things could be said about a robot who is T3-indistinguishable from a person (including that it may lack a mind), but it cannot be said that its internal symbols are about the objects, events, and states of affairs that they are about only because they are so interpretable by me, because the robot itself can and does interact, autonomously and directly, with those very objects, events and states of affairs in a way that coheres with the interpretation. It tokens "cat" in the presence of a cat, just as we do, and "mat" in the presence of a mat, etc. And all this at a scale that is completely indistinguishable from the way we do it, not just with cats and mats, but with everything, present and absent, concrete and abstract. That is guaranteed by T3, just as T2 guarantees that your symbolic correspondence with your T2 pen-pal will be systematically coherent.

But there is a price to be paid for grounding a symbol system: It is no longer just computational! At the very least, sensorimotor transduction is essential for robotic grounding, and transduction is not computation.

Cognition: Virtual Vs. Real

Or is it? Let us recall the two versions of Church's Thesis introduced earlier: There was the purely formal one, CTT, and the physical one, CTTP, and the latter came in a Weak and Strong form. There is no problem with the Weak CTTP, because it merely claims that every physical system is formally equivalent to a Turing Machine, and that would be true of a transducer as well, just as it would be of an airplane, a furnace, or a solar system: All of these can be simulated, state-for-state, by a computer, to as close an approximation as one likes. But no one would ever mistake the simulation for the real thing (except perhaps a person in a Virtual Reality simulation, which, I hope the reader realizes, is completely irrelevant to the issue at hand, which is precisely not about a trompe l'oeil for an observer/interpreter, but about the real, interpreter-independent world): A computer simulation of an optical transducer does not transduce real light (that would require a real optical transducer), it transduces virtual light, i.e., computer-simulated light, and the transduction itself is virtual. The same is true of a virtual plane, which does not really fly, but merely simulates flight in simulated air. By the same token, virtual furnaces produce no real heat, and virtual solar systems have no real planetary motion or gravity.

All these cases are completely unproblematic on the Weak CTTP. No one is tempted to propose a "C=F" Computationalist Thesis according to which flying is just a form of computation. Computation is just implementation-independent, systematically interpretable symbol manipulation, whether the symbols are interpretable as a plane, a furnace, a transducer, a pen-pal or even a robot. A virtual plane does not really fly, a virtual furnace does not really heat, a virtual transducer does not really transduce; they are just symbol systems that are systematically interpretable as, respectively, flying, heating and transducing. All of this should be quite obvious. A bit less obvious is the equally valid fact that a virtual pen-pal does not think (or understand, or have a mind) -- because he is just a symbol system systematically interpretable as if it were thinking (understanding, mentating).

Still less obvious (particularly in view of what was just said about T3 robotic grounding of T2 symbol systems), but still valid, and for exactly the same reasons: a virtual robot does not think (any more than it moves, or sees, or does sensorimotor transduction): A virtual robot in a virtual world is merely a symbol system systematically interpretable as if it were thinking (moving, seeing, transducing). The truth of the Weak CTTP would guarantee that such simulations are possible, and it would strongly imply that computational modeling was an excellent way of arriving at an understanding of physical systems (including furnaces, planes, solar systems, transducers and robots -- Searle (1980) called this "Weak AI"), but it would not support C=C ("Strong AI") Why? Because even if the simulated T3 robot captured (i.e., simulated) every relevant property of cognition, it would still be just an ungrounded symbol system. To ground it, one would have to build a real T3 robot -- and I hope it is obvious that that would not amount to merely attaching sensorimotor transducers to the computer doing the simulation (any more than building a real plane or furnace would amount to merely attaching sensorimotor transducers to their respective simulations). But even if it did , it would only be the grounded system as a whole -- plane, furnace, T3 robot -- that would be flying, heating and thinking. The purely symbolic module would not. Hence C=C would still be false.

Differential Equations and Implementation Dependence

What about the Strong CTTP, according to which a plane is a computer? Well, either this is wrong, or it is uninteresting. The only virtue of informing cognitive scientists that cognition was computation rather than some other physical process it might have been was that computation was a natural kind of some kind. If every physical process is in fact computation, then the cognitive scientist might as well have ignored the computationist's message and just kept on hunting for what the right physical process(es) might turn out to be.

But I actually think the Strong CTTP is wrong, rather than just vacuous, because it fails to take into account the all-important implementation-independence that does distinguish computation as a natural kind: For flying and heating, unlike computation, are clearly not implementation-independent. The pertinent invariant shared by all things that fly is that they obey the same sets of differential equations, not that they implement the same symbol systems (Harnad 1993 a). The test, if you think otherwise, is to try to heat your house or get to Seattle with the one that implements the right symbol system but obeys the wrong set of differential equations.

So much for strong CTTP. What about C=C? A weak version, according to which cognition can be simulated -- to as close an approximation as one likes, by computation -- is really just a variant of the Weak CTTP: Why would a nondualist expect that cognition would differ in this respect from any other physical process (flight, heat, gravity), all likewise simulable by computation? But Strong Computationalism, according to which every implementation of the right symbol system would cognize, is either wrong in exactly the same way the Strong CTTP is wrong (or vacuous), or, if it posits the Strong C=C while rejecting the Strong CTTP then it is merely equivocating on the unobservability of cognition (a property that sets cognition apart from flight, heating, movement, transduction, and even gravity).

For cognition, defined by ostension (for lack of a cognitive scientific theory), is observable only to the mind of the cognizer. This property -- the flip-side of the mind/body problem, and otherwise known as the other-minds problem -- has, I think, drawn the Strong Computationalist unwittingly into the hermeneutic circle. Let as hope that reflection on Searle's Argument and the Symbol Grounding Problem, and especially the potential empirical routes to the latter's solution (Andrews et al in prep; Harnad 1987, Harnad et al 1991,1994), may help the Strong Computationalist break out again. A first step might be to try to try to deinterpret the symbol system into the arbitrary squiggles and squoggles it really is (but, like unlearning a language one has learnt, this is not easy to do!).

Andrews, J., Livingston, K., Harnad, S. & Fischer, U. (in prep.) Categorical Perception Induced by Learning

Church, Introduction to mathematical logic. Princeton, Princeton University Press 1956

Dietrich, E. (1990) Computationalism. Social Epistemology 4: 135 - 154.

Fodor, J. A. (1975) The language of thought New York: Thomas Y. Crowell

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

Galton, A. The Church-Turing thesis: its nature and status. AISB Quarterly, Autumn 1990 (no.74):9-19.

Harnad, S. (1982) Consciousness: An afterthought. Cognition and Brain Theory 5: 29 - 47.

Harnad, S. (ed.) (1987) Categorical Perception: The Groundwork of Cognition. New York: Cambridge University Press.

Harnad, S. (1989) Minds, Machines and Searle. Journal of Theoretical and Experimental Artificial Intelligence 1:

5-25.

Harnad, S. (1990 a) The Symbol Grounding Problem. Physica D 42: 335-346.

Harnad, S. (1990 b) Against Computational Hermeneutics. (Invited commentary on Eric Dietrich's Computationalism) Social Epistemology 4: 167-172.

Harnad, S. (1990 c) Lost in the hermeneutic hall of mirrors. Invited Commentary on: Michael Dyer: Minds, Machines, Searle and Harnad. Journal of Experimental and Theoretical Artificial Intelligence 2: 321 - 327.

Harnad, S. (1991) Other bodies, Other minds: A machine incarnation of an old philosophical problem. Minds and Machines 1: 43-54.

Harnad, S. (1992 d) Connecting Object to Symbol in Modeling Cognition. In: A. Clarke and R. Lutz (Eds) Connectionism in Context Springer Verlag.

Harnad, S. (1992 b) The Turing Test Is Not A Trick: Turing Indistinguishability Is A Scientific Criterion. SIGART Bulletin 3(4) (October) 9 - 10.

Harnad, S. (1993 a) Grounding Symbols in the Analog World with Neural Nets. Think 2(1) 12 - 78 (Special issue on "Connectionism versus Symbolism," D.M.W. Powers & P.A. Flach, eds.).

Harnad, S. (1993 b) Artificial Life: Synthetic Versus Virtual. Artificial Life III. Proceedings, Santa Fe Institute Studies in the Sciences of Complexity. Volume XVI.

Harnad, S. (1993 c) Problems, Problems: The Frame Problem as a Symptom of the Symbol Grounding Problem. PSYCOLOQUY 4(34) frame-problem.11.

Harnad, S. (1993 d) Grounding Symbolic Capacity in Robotic Capacity. In: Steels, L. and R. Brooks (eds.) The "artificial life" route to "artificial intelligence." Building Situated Embodied Agents. New Haven: Lawrence Erlbaum

Harnad, S. (1994 d) The Origin of Words: A Psychophysical Hypothesis In Durham, W & Velichkovsky B (Eds.) "Naturally Human: Origins and Destiny of Language." Muenster: Nodus Pub.

Harnad, S. (1994 c) Levels of Functional Equivalence in Reverse Bioengineering: The Darwinian Turing Test for Artificial Life. Artificial Life 1(3): 293-301.

Harnad, S., Hanson, S.J. & Lubin, J. (1991) Categorical Perception and the Evolution of Supervised Learning in Neural Nets. In: Working Papers of the AAAI Spring Symposium on Machine Learning of Natural Language and Ontology (DW Powers & L Reeker, Eds.) pp. 65-74. Presented at Symposium on Symbol Grounding: Problems and Practice, Stanford University, March 1991; also reprinted as Document D91-09, Deutsches Forschungszentrum fur Kuenstliche Intelligenz GmbH Kaiserslautern FRG.

Harnad, S. Hanson, S.J. & Lubin, J. (1994) Learned Categorical Perception in Neural Nets: Implications for Symbol Grounding. In: V. Honavar & L. Uhr (eds) Symbol Processors and Connectionist Network Models in Artificial Intelligence and Cognitive Modelling: Steps Toward Principled Integration. pp. 191-206. Academic Press.

Hayes, P., Harnad, S., Perlis, D. & Block, N. (1992) Virtual Symposium on Virtual Mind. Minds and Machines 2: 217-238.

Nagel, T. (1974) What is it like to be a bat? Philosophical Review 83: 435 - 451.

Nagel, T. (1986) The view from nowhere. New York: Oxford University Press.

Newell, A. (1980) Physical Symbol Systems. Cognitive Science 4: 135 - 83

Pylyshyn, Z. W. (1984) Computation and cognition. Cambridge MA: MIT/Bradford

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

Turing, A. M. (1964) Computing machinery and intelligence. In: Minds and machines. A. Anderson (ed.), Engelwood Cliffs NJ: Prentice Hall.

Turing, A. M. (1990) Mechanical intelligence (D. C. Ince, ed.) North Holland