2 — Requiem for the Computational Theory of Mind[†]

Preamble

Thanks to Terry Horgan, for that very gracious introduction. Thanks too to Güven Güzeldere and Stevan Harnad, for an excellent anniversary program. Thanks to the ever-gracious Betty Stanton, also, for taking the time to return to what we all know is her real home. And thanks to Ken Taylor, for the heroic job of local arrangements.

I also want to thank CSLI—the Center for the Study of Language and Information—for their financial support and help in hosting this meeting. I cut my teeth at CSLI; it is good to be home, and to give it public credit.

At first, in fact, I wondered whether I should not speak on a topic related to CSLI's theories of situated language. I had in mind the double-indexical theory of truth, which mysteriously seems to have escaped John Perry's attention: why we say "here, here"[1] to mark propositions with which we agree; "there, there," to mark those with which we do not. You might think *proximity* signals *approval*. But that cannot be right, since "now, now" indicates protestation, not approbation. The phenomenon must be related to that famous Morganbesser quip: that "yeah, yeah" shows how

[†]Presidential Address, 25th Anniversary Meeting, Society for Philosophy & Psychology, June 21, 1999. The talk is reproduced exactly as given, orally, complete with introductory remarks, in order to preserve the decidedly informal character of the (after-dinner) setting.

[1]It was fortunate thing that this was a talk.

two positives can make a negative. Beyond that, illumination awaits.

I store my notes about these deep issues in a file in the corner of my hard drive, under the heading "Questions of great philosophical significance in which I have never been able to interest any philosopher." Here is another. Suppose one refers to the sun. How long does it take reference to get there? At this very institution, I once asked the question of no less a luminary than Alonzo Church, who (without batting an eyelash) said that reference "travels at the speed of logic" (I take it that is supposed to be fast). Problem is, I thought physicists had dispatched the notion of simultaneity to the same dust heap as the luminiferous ether. So there is an issue to be resolved. Is Brentano's arrow of directedness genuinely superluminary? Do we need Bell's theorem? Maybe Penrose missed his calling. Haven't you sometimes wished he had studied semantics?

Fortunately, I will not talk about these issues, either, since I know nothing about relativity. Rather than speculate about things I do not understand, in fact, I will confine my remarks to something I do. That is: I will return to my home field of…*psychology*.

Now you may not have realised that I am a psychologist. Neither did I, till I was so generously offered this position. I have certainly never taken any psychology courses—as my colleagues at Indiana will readily attest. But it is a tradition for the presidency of SPP to alternate, with philosophy presidents on even years, psychology presidents on odd. Last year (1998) we had Bob McCauley; next (2000), Terry Horgan—both with intensionally-correct PhDs. The odd years are reserved for us scientific pretenders.

For that is what really matters: not what I am, but what I am *not*. I fall in the "other" category. And that seems just right. For I have always viewed SPP as something of a philosophers' "at home"—a party, hosted by philosophers, for scientists interested in the mind. The philosophers have a chance to find out what is actually true; we scientists undergo a little conceptual grooming. And the food is great.

From that perspective, coming from computer science (my real home) seems appropriate enough. For nothing, arguably, so uni-

fied cognitive science over its first twenty-five years, and the philosophy of mind along with it, as the computational theory of mind. Or nothing did unify it. Paradoxically, as you have undoubtedly noticed, computational presence in cognitive science has recently been on the wane. Where I teach, telling students to read about the computational theory of mind these days is rather like recommending they listen to Mantovani (or even Monteverdi). And I am not even fifty.

So that is what I really want to talk about tonight. What happened to the twenty-five-year-old conversation between computing and the philosophy of mind?

1 Talking to philosophers

I will get to the content of that conversation in a moment. First, though, a word about talking to philosophers.

As anyone from the provinces—oops, sciences—will tell you, talking to philosophers is a little strange. They talk a lot, first of all; that much is hard to miss. They use odd constructions—hypotheticals three layers deep, with patently untrue premises, somehow convinced that the conclusions still matter. And they talk fast. Except, as I discovered, it s a very special version of fast. Not just 217 words per minute (I tried that, once—no one understood a word). Rather: *fast at clause boundaries*, when the situation is vulnerable, lest someone take the floor. As long as you are manifestly *mid-clause*, however—grammatically safe from coming to the end—you can slow down to a reasonable speed, so that everyone can follow. Arvind Joshi should study the stuff.

More seriously, different norms apply. In computer science, papers *report* on research; in philosophy, they *are* research. When I first showed a paper to a philosopher, I felt as if I had given them a map of buried treasure—only to have them respond: "Great; such an exemplary map! Such good lines. All the labels are so well arranged. You can see these nice paths." Good of them and all…but, well, it was the *treasure* I was trying to interest them in. This difference must in part be because computer science is essentially engineering. Its methods are neither theoretical nor empirical, but *synthetic*. Never forget this fact: it is what we build, not what we say, that matters.

All these things can be learned, and partially accommodated-though as we will see, they have more impact than one might expect. And they are laced with another distracting issue: of vocabulary.

As everyone knows, philosophy, like physics, uses ordinary English for its technical terminology-a trap for the unwary. As do all other cognitive sciences. It might not be so bad, except-have you noticed?—they all use the same 200 words, but with different meanings. *Object*, *concept*, *variable*, *function*, *model*, *class*, *meaning*, *semantics*, *reference*, *identifier*, *interpretation*, *binding*, *type*—every one of these terms has a demonstrably different meaning across SPP's constituent fields. It makes me wish that interdisciplinary conferences would provide U.N.-style instantaneous translation into different disciplines. Go to a psychology talk, put on headphones, dial the switch, and presto!—you would hear it translated into linguistics. Feminist cultural theory, translated into analytic epistemology! On-the-fly translation between analytic and continental! Twenty years ago, not entirely facetiously, I suggested to Harry and Betty that Bradford Books publish a dictionary of these 200 words, with entries for how each is used in each cognitive discipline. When we were setting up CSLI, we even considered appointing a "technical term librarian": anyone planning to use a term in a technical sense would be required to sign it out-and promise to return it on a pre-assigned date.

The vocabulary problem is particularly acute between computing and philosophy, for an interesting historical reason. Computer science took many of its technical terms—*language*, *semantics*, *variable*, *name*, *identifier*, *syntax*, etc.—from philosophical logic, but then, like any good science, proceeded to change their meanings. This has had the unfortunate consequence of allowing some philosophers to think they understand what computer scientists are saying. Consider Searle. While I doubt that he has spent much time hacking C++ code or writing Java applets, he nevertheless thinks he knows what computation is—in spite of the fact that I do not know a single programmer who thinks he even remotely "gets it." Why does Searle think this? *Because we describe our machines using his words*. Everything we say; *it* all sounds so familiar, to him. *Searle would be right*, I tell my students, if computer scientists' own descriptions were interpreted

on the assumption that the words mean what they do in logic (which is where we got them from).

And his case is not unique. Unrecognised overlaps in technical vocabulary continue to sew conceptual misunderstandings about the foundations of our interdisciplinary project. A critical example is going to play a role in tonight's story, having to do with the word 'computation.'

2 The computational theory of mind

So let's turn to that. As I said, the real excitement in cognitive science, back in the 1970s, had to do with this thing called **the computational theory of mind**. You all know the story: a set of internal words or sentence-like tokens—items in a language of thought—that (i) carry content (mean something) about situations or states of affairs in the world, but (ii) are manipulated internally, not in virtue of that content, but instead in virtue of their abstract shape or "form." As in logic (too much as in logic, as we will see). This catechism was chanted in the halls of every self-respecting philosophy and cognitive science department.

To understand this hypothesis, and the excitement it generated, we have to back up a bit.

The basic situation everyone was wrestling with—psychologists, philosophers, computer scientists alike—had to do with intentionality. And intentionality, on widely-shared metaphysical assumptions, involves an interplay between **meaning** and **mechanism**. This dialectic had occupied people's imagination for centuries: how a "mere mechanism"—naught but a lump of clay—could sit up and think: reason, wonder, dream, even be conscious.

Crucially, minds and computers are both intentional, and so both instantiate the meaning/mechanism dialectic. Take the mechanism side. That computers are mechanisms is obvious; that minds are mechanisms did not use to be obvious, but most academics believe it now. Similarly on the meaning side. Once again, it is obvious that minds involve meaning (meaningfulness is a prime candidate for the "mark of the mental"). That computers traffic in meaning is equally evident, if you look at practice. Computational discourse is rife with intentional words: *information processing*, programming *languages*, *data* bases, *knowledge represen-*

tation systems, *simulations*, *models*, *correctness*, and so on.

Moreover, it is only because both minds and computers are intentional that there was any reason to suppose the computers might be intelligent—that computers were relevant to mind, that we might be computers. In most other ways, after all (physical make-up, price, eating habits, etc.) computers and people are almost completely different.[2]

But the idea that excited people, back in the 1970s, was not simply that computers, like us, were intentional—that we were building, not just growing, intentionality. What set the world on fire was something more pointed: a belief that computers were *special*—that Turing (or someone) had discovered a distinctive way of merging meaning and mechanism—a way that had no one had understood before, had now been discovered, was embodied in Turing machines and Eniacs and Pentiums, and might even—who knows?—be embodied in minds. It was this "something special" that was called **computing**.

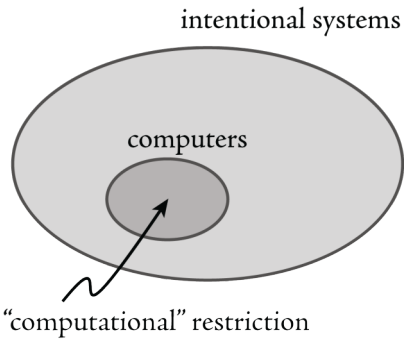

Figure 1 — Computing as special

It is critical to understand the importance of the *"specialness"*—this idea that there was a distinct, identifiable, scientific property "being a computer" or "being computational," which was: (i) uniquely true of all and only computers; (ii) would play a role in scientific laws; and (iii) might be relevant to understanding the mind. Fodor is perfectly clear about this in "Methodological Solipsism."[3] That people were general intentional systems (or even, somewhat more narrowly, general representational systems) he took to be obvious-and boring. The computational theory of mind, on the other hand, he took to be non-obvious, and exciting. It was non-obvious and exciting because there was something distinctive

[2]I once gave a talk arguing that the most important difference between computers and people was that computers have back-up tapes. It seemed innocent enough—until, when I arrived at the occasion, I discovered that it was being hosted at a senior citizen home.

[3]Fodor (19■■).

about being computational. *Computer science had had an idea*. That was Turing's brilliance.

Perhaps a picture will help (figure 1). The larger circle is the space of all (perhaps all possible) intentional systems; the smaller one is the space of all computers or computational systems. The "computing" idea we are talking about is the (characteristic) property of the subset.

So that was context in cognitive science mid 1970s—which, as it happens, was when I went to graduate school at MIT.

Now you have to understand what it was like to come to philosophy from computing, only to discover it was all excited about your field. It was a little like reading about your area of expertise in the New York Times: thrilling headlines—but a bit queasy-making, as soon as you started to read. In particular, it was not clear what philosophers actually knew about computing. In "Tom Swift and his Procedural Grandmother,"[4] Fodor said that "*providing a compiler for a language is tantamount to specifying a [procedural] semantic theory for that language*." *Compiler*? Did he maybe mean *interpreter*? (Compiling is just translating; it does not make things actually happen.)

So I went to talk to Fodor. "How do you know that computers work in the way you describe?", I asked. Now Fodor, modulo a certain bluster, is a pretty honest guy. "I made it up!", he said. "Look," he went on, "you are the computer scientist; it works in whatever way you think it does."

"If your understanding rests on my understanding," I thought, "then you are in trouble."

Seriously, here was the situation. Computer science—or computational practice, or something like that—was supposed to be supplying cognitive science with an *idea*: an account of the property "computational," on which the computational theory of mind could rest. The catechism—the claim that computation was formal symbol manipulation—was, I took it, philosophy's best attempt to describe that property.

I was supposed to represent computer science. Sure enough, I

[4]Fodor (19■■).

knew how to design programming languages, write compilers, build operating systems. But an idea? An idea good enough for philosophy? That is serious. I wasn't at all sure about that.

3 Formal Symbol Manipulation

So while most people in cognitive science busied themselves with their main project—to find out whether formal symbol manipulation was true of mind—I set out on a parallel, side path: to find out *whether it was true of computers*—specifically, what I call computation in the wild:[5] concrete, real-world systems. That is, to put it as bluntly as possible: I set out to determine the truth of the following radical thesis: *the computational theory of computing*.

The natural way to do this, you might think, would be to ask computer scientists, or look at computer science—at the discipline that builds and studies and theorises computing—and see whether it takes computing to be formal symbol manipulation. Not that their answer would necessarily be definitive; computer science could be wrong. But it is not often that a whole field is wrong. Anyway, it seemed like a good place to start.

So I went to computer science, and "asked it," as it were, whether it thought computers were formal symbol manipulators.

Computer scientists had never heard of formal symbol manipulation. They did not know what I was talking about.

Now this is a bit tricky, so bear with me. Computer science had heard the words 'formal symbol manipulation.' In fact they used the words—or rather, correlative words from logic—syntax, semantics, model, interpretation, completeness, etc. But remember what I said about vocabulary. They used the same words, *but they meant different things!* On the surface, it *sounded* as if we were talking about the same things. But if you pushed, in order to make sure that we were really on the same wavelength, it turned out that they were talking about something else.

What computer scientists had heard about was Turing. And Fodor was certainly right about one thing: Turing was one smart guy (I would definitely recommend him for tenure). But as for

[5]With an apology to Ed Hutchins.

understanding formal symbol manipulation, I sometimes think Turing was the villain. Because if you look hard at the original Turing papers, he almost drops the ball with respect to the critical issue for formal symbol manipulation: namely, the *symbol* part, the part having to do with *representation*. And make no mistake: formal symbol manipulation is an idea about the processing of semantically interpreted structures (as I have said elsewhere, if you are not interested in semantics, then you should call your thesis "stuff manipulation"[6]). Turing's 1937 paper[7] opens with laudable representational clarity: "the computation of functions whose representation as a decimal can be calculated by finite means."[8] But by the time you get to the middle—to all that stuff about universal machines and modelling and quadruples and what functions can be computed and so on and so forth—the difference between numbers and numerals has been stirred into oblivion. And as everyone knows, if you cannot tell a number from a numeral—if you do not mind your "uses" and your "mentions"—you definitely are not going to be invited to the next philosophers' party.

No, the theory of Turing machines, I have come to believe—and of effective computability, and complexity theory, and just about all the theoretical edifice of theoretical computer science—is *not about formal symbol manipulation at all*! It is about something else. What else? I will get back to you in a minute on that.

But back to the main plot. Here I was, trying to figure out whether computers were formal symbol manipulators. Theoretical computer science was no help. And so I had to do my own empirical study.

It sure took me long enough (going on thirty years). But I can now tell you the answer.

The answer is no.

There are many reasons. Tonight I will just quickly mention two.

The way to figure out whether computers are formal symbol

[6]Smith (19■■).
[7]Turing (19■■).
[8] «check, and refer»

manipulators is to see how formal symbol manipulation is *special*—and then to see whether real-world computers are special in that way. That is, one needs to see what essential restriction formal symbol manipulation places on general intentional systems—what exactly it is that formal symbol manipulation claims distinguishes the smaller inner circle from the broader outer one—and then determine whether the resulting restricted class coincides with the class of computers in the wild.

Now formal symbol manipulation, interestingly enough, is not itself a formal thesis, and so it is not exactly clear what it means. But after long study I have determined that, at least at a minimum, it places the following two restrictions on intentionality:

1. That there be symbols; and
2. That they be manipulated formally.

I will look briefly at each.

3a Language-like tokens

Start with the symbols. In this context, symbols are taken to be elements in an explicit, compositional array of language-like (representational) tokens. And on this characterisation, it sure seems as if computing involves symbols. Just look at an ordinary program-in, say, an Emacs buffer:[9]

```
if empty (paper-tray)
   then display-user dialogue-box(“out of paper”)
   else start-copying ... whatever
```

This looks paradigmatically symbolic. But appearance is distracting. There is no more reason to suppose that computing involves symbols, from looking at a program, than to think that car engines combust symbols, because there are symbols in the blueprints used to control the automatic milling machines.[10]

Sure enough, program ingredients are explicit and language-like. But *programs are completely irrelevant for psychology*. Programs are a convenience we use to constrain universal, general-purpose computing engines in order to flexibly implement fixed, task-specific architectures. Sure enough, you can mix and match

[9] «ref»
[10] See Smith (19■■).

identifiers and so forth, when writing a program-as compositionality and productivity require. But once the writing is done, the resulting program is held constant, during its execution[11]—invisible to the process it describes. It could as well be eliminated (and often is eliminated, by the compiler).[12]

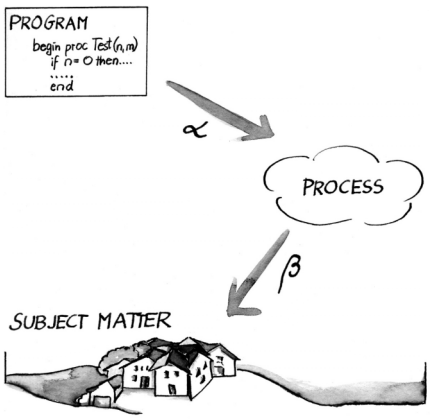

Figure 2 — Program, Process, and Task Domain

What matters to psychology is not *programs*, but *the processes and architectures those programs specify*: how they operate, interact with the world, modify their internal state (see figure 2). And the state-bearing ingredients inside such processes are called *data structures*. So the question we need to ask, with respect to the computational theory of computing, and thus *before* we take up any subsequent question about the computational theory of mind, is the following one: *are data structures explicit, language-like tokens?* The answer, in general, is *no*. (Do not be distracted by the fact that they are given names in programs. That is irrelevant; those names are not themselves the data structures; they (like all names) are *names*: they *denote* data structures—just as names of engine parts denote pieces of steel.) In the vast majority of cases, data structures are highly-constrained, purpose-specific, and non-generic. They have none of the properties that Fodor, Pylyshyn, Evans, van Gelder, and others think symbols have: of modularly designating arbitrary predicates or relations, that, modulo certain appropriateness conditions, can be algebraically recombined in systematic and productive ways.[13]

(In passing, I might note that the *program-process* relation, labelled a in the diagram, is what computer science calls "semantics." What we in computer science are interested in is the rela-

[11]Except for self-modifying programs, of which there are virtually none.
[12]Taken from Smith (19■■), page ■■.
[13]See for example Fodor & Pylyshyn (19■■), Evans (19■■), and van Gelder (19■■).

tion labelled b in the diagram, between the thereby-specified processes and the worlds or task domains that the processes are about. To distinguish, we might call these program semantics and process semantics, respectively. This is one example of the sort of terminological confusion I mentioned at the outset.)

Admittedly, some computer systems employ symbols: theorem provers, expert systems—plus of course interpreters and compilers. But those, I venture to say, comprise no more than one percent of the programs that are written. Compared to them, there are hundreds of billions (if not trillions) of lines of commercial code that are *not symbolic*. These real-world programs—or rather, real-world processes—are tacit, implicit, non-conceptual. (Just try asking Windows NT why it put up a blue screen of death. Or Unix whether it likes thrashing so much. Or Deep Blue whether any of the arrangements of pieces of any of the configurations it examined reminded it of the face of its designer.)

And so restricting intentionality to symbol-using systems is vastly too narrow to capture computation in the wild.

3b Independent of semantics

OK, so that is the first reason that real-world computing is not formal symbol manipulation: there are no symbols (in the required sense). The second reason is that they are not manipulated formally. That is: data structures are not—at least in general—manipulated independent of their semantics.

Now that phrase—"independent of semantics"—is as recalcitrant as any in the philosophy of mind. Let me just say this about it. The systems where it is motivated—where there is some reason to think it is true—are those systems that are *entirely disconnected or detached from their subject matters*: theorem provers proving theorems about inaccessible cardinals, NASA simulation systems figuring out whether a rocket will venture outside the solar system—things like that. Where the independent-of-semantics mandate is *not* reasonable is where systems are thickly engaged, causally, with their subject matters. This is well-recognised: Devitt, Levine, Anthony, and others have pointed out that the formality condition is difficult even to *understand*, and almost cer-

tainly not true, in cases of transducers.[14] And there is Fodor's own telling comment: "Please don't ask me about transducers; I am particularly busy just now."[15]

So the formality condition is (arguably) true when systems are disconnected. Is that a necessary condition on computation in the wild? What about those trillion lines of commercial code?

The answer is interesting. By far the majority of those programs that are *good* programs—situations where computing systems have proved resilient and successful—are cases where computers traffic, directly, in their subject matters: network routers, compilers, window systems, document processing systems, email programs, web browsers, and so on. *Especially in those situations in which it is most* successful, computation in the wild, far from being detached, is highly involved in its subject matter.

In sum: once we set overlapping vocabulary aside, and look the subject matter squarely in the face, we are forced to conclude that real-world computing is not formal symbol manipulation.

It is all a bit ironic. It turned out that what I was discovering, on my side path, was the same lesson the main body of cognitive science was discovering, about people. Just as they were abandoning so-called "computational" (i.e., abstract, disconnected, purely logical) models of mind, in favour of embodied, engaged, interactive alternatives, so too I (and a lot of other computer scientists, I might add[16]) were, in our own way, *rejecting abstract, disconnected models of computers,* in favour of—you guessed it!—embodied, engaged, interactive alternatives.

Now this raises an interesting possibility. You might think, given all these results, that my brief would be to resuscitate the computational theory of mind. Maybe we can have a new computational theory of mind, one framed not in terms of the worn out formal symbol manipulation idea, but in terms of a new idea—of dynamic, embodied, real-world interactive computing. Tacit programming. Ready-to-hand software! Whatever. Then

[14]«Refs»
[15]«Ref; check with Murat»
[16]E.g., see Stein (19■■)

gramming. Ready-to-hand software! Whatever. Then the computational theory of mind could be true once again.

As is evident in my title, however, I have come to bury the computational theory of mind, not to praise it. So I still have some explaining to do.

4 Computing

I have said that formal symbol manipulation does not work as a theory of computing. But I also said that it was not computer scientists' idea of computing, anyway. So what is their idea. What does computer science take computers to be?

It turns out there are several answers—several standard candidates. They are all familiar: *information processors*, *digital automata*, *rule-governed systems*, *rule-following systems*, *physical symbol systems*, etc. Ideas are not the easiest things to count, but by my lights there are anywhere from half a dozen to a dozen different such stories. Some (even many) people think that these stories are all the same—because of various equivalence proofs. But that is an elementary, if common, mistake. Those equivalence proofs are extraordinarily coarse-grained, and *gloss over everything that matters for a theory of mind*. At the level we care about them, the ideas all differ, both intensionally and extensionally. A Lincoln Log contraption (so long as its parts were not information-bearing), would be a digital state machine, but not an information-processor. If continuous representations are possible, which seems not only possible but likely, then a formal symbol manipulation system could fail to be a digital state machine. And so on.

My real project, therefore, has been to assess not just the formal symbol manipulation idea, but this whole suite of other alternatives—ideas I call '**construals**' of computing. The plan, for each, has been to understand where it came from, what it says, and—crucially—whether it is true of real-world computers. That is, in terms of our graph, I have wanted to see whether any of these other alternatives could do better than formal symbol manipulation idea did in restricting the class of intentional systems to all and only computers.

I will not bother you with the details. Let me simply cut to the bottom line. There are three results, of increasing strength.

1. First, *none of these other standard construals works, either*. No one alone, nor any group in combination, is strong enough to delimit the proper computational subset of the full space of general intentional systems.
2. Second, I was not able to find or make up any non-standard construal that worked, either. So in the end my search for a conceptually sound and empirically adequate theory of computation-in-the-wild failed. After 30 years of looking, I have come up empty-handed.
3. Third, I did learn something, though. Not only did I fail; I had to fail. I had to fail because there is no such theory. There never will be such a theory. There is no theory, because *there is nothing there to have a theory of*.

There are not any computers! It is all a hoax, perpetrated by Bill Gates.

Seriously, of course there are computers. What I am saying is that the property computational or being a computer does not pick out a natural or scientific kind. It is not a property that will figure in scientific laws, or underwrite any deep or interesting scientific generalisations. Nothing of scientific interest holds of a computer in virtue of its being a computer, or of anything at all in virtue of its being computational.

In the end, computers turn out to be rather like *cars*: objects of inestimable historical and economical and social importance, the existence of which has and will continue to transform our lives. Lots of theories apply to cars: physics, thermodynamics, ergonomics, ecology, and so on. But no one writes equations with CAR(x) in them,[17] and very few universities have departments of automotive science. (I am embarrassed to say that MIT, where I came from, did have such a department—but hey, it is an engineering school, and anyway, my understanding is that they have since thought better of it, and shut the place down.)

For "computational" to be a scientific property—for there to be a theory of computation—as I have said since the beginning, there

[17]Except of course for John McCarthy and Doug Lenat.

must be something *special* about computers. It is that "specialness" that a theory should capture. And the result of my analysis—the reason why 'computational' is not going to survive as a scientific property—is that *there is not anything sufficiently special*. In spite of the press, real-world computers turn out not to be necessarily formal, or necessarily digital, or necessarily abstract, or necessarily context-independent…or necessarily any other property that has been suggested (or that I have been able to find). Rather, what computers are are dynamic, intentional systems-socially constructed[18] intentional artifacts, the best, at any moment in history, that we know how to build.

Period. No more and no less. That is all there is to say.

Now, for a computer scientist, you might think that this is a dismal result. On the contrary, however, I believe almost exactly the opposite: *the lack of a theory of computing is the most positive, optimistic, exciting possibility that anyone*—even the most unregenerate computational triumphalist—could possibly hope for. I do admit, though, that seeing things this way requires a change of perspective. That change of perspective is something I now want to explain.

On the old view, computing was taken to be an autonomous, distinctive subject matter—warranting its own theory, its own academic department, its own vocabulary. A subject matter whose name could be chiseled into the facades of twenty-first century university buildings, alongside physics and mathematics and literature and maybe even economics. What I am claiming is that it is no such thing. Rather, what computing is is an *historical occasion*—a laboratory of middling complexity, between the frictionless pucks and inclined planes of physics, and the full-blooded complexity of the human condition, in which to see issues of meaning and mechanism play out.

To put it in a slogan, computation is a **site**, not a **subject matter**.

[18]Socially constructed not as a meta-philosophical standpoint, but in the literal sense of being constructed by groups of people.

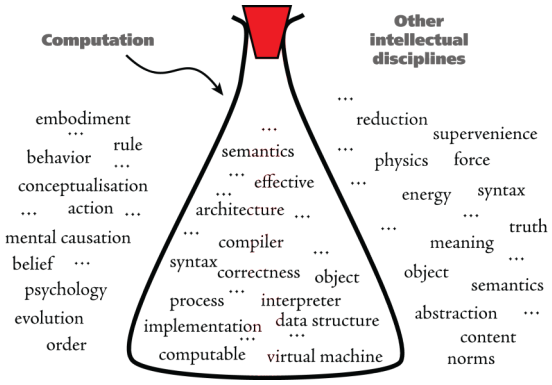

Figure 3 — Computation as closed

A pair of figures may indicate why this is a good result, not a bad one.

Figure 3 indicates the traditional view we have been working with, in which computing is taken to be an autonomous discipline—a subject matter with its own theories, vocabulary, insights, phenomena. Instead of drawing it as a simple circle, I have indicated it this time as a stuffed Erlenmeyer flask. Inside are the properties and relations classically taken to be computationally specific: *implementation*, *abstraction*, *effectiveness*, *complexity*, and so on. Outside, this time, I have put the rest of the intellectual map: philosophy, psychology, economics, whatever.

The picture I am recommending is given in figure 4. What I am indicting, as the result of my 30 years of study, is not the contents of the flask, but the flask itself—the bottled-up corker of an idea that there is an interesting property of "being a computer" that separates what is inside from what is outside. So what I have done is to peel back the flask—undo the conceit that computational things are theoretically distinct. What this allows, as the picture shows, is that phenomena that have been studied as computation-internal are *now allowed to join up with their appropriate partners that have been thought to be computation-*

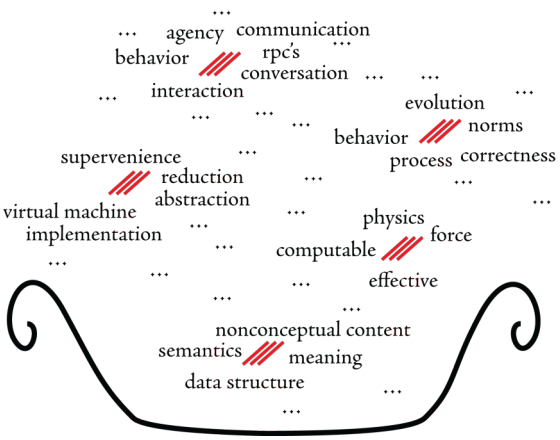

Figure 4 — Computation as open

external.

Take just one example: the question of how a system, described at one level of description, relates to that same system, described at another (say, lower) level of description. In computing we call this *implementation*; there is perhaps no more critical and powerful a notion. All sorts of issues arise: of *virtual machines*, of *abstract data types*, of *implementation boundaries*. There is very interesting work going on at the moment breaking down the idea that such abstraction boundaries are opaque, instead seeing how properties of the underlying implementation inevitably "shine through" at upper levels.[19] Various labels are used for this- "grey box" or "glass box" abstraction, for example, in place of the prevailing idea of "black box." But of course the very same issues are studied outside computing—for example in philosophy, under notions of type- and token-reduction, local and global supervenience, non-reductive physicalism, etc. And there is interesting work going on there, too—for example in the literature on emergence. What the idea that computation is a distinct phenomenon has done is to keep these two discussions apart. Somewhere in another corner of my hard drive I have another list: titles and abstracts for "PhD theses needing written."[20] One of them is on bringing these two disciplines together—which after all are talking about exactly the same thing.

In sum, my original misgivings, in response to Fodor, were right. *Computation does not give us an idea.* What it gives us is something else, entirely—something hugely valuable, I believe, even a sine qua non without which we will never come to understand the mind. But it is a thing of a completely different *kind*. What computing gives us is *experience*: insights and intuitions and practice and knowledge about the very same intentional phenomena that are being studied everywhere else. Phenomena, I might add, about which I do not think we yet have any very good theories of (but more on that later).

Now I have presented this picture a few times to computer scientists, and I have been stunned by their response. They are surpris-

[19]«References (e.g. to 'grey box abstraction,' IRL work, Kiczales et al., etc.)»
[20]With apologies to the Pennsylvania Dutch.

ingly agreeable! "It makes sense!" they say. And I agree: there is something quite deliciously relaxing about it. For one thing, it caters to their computer scientists' egos: it means that computation is not just taking over the world: it *is* the world (they like that). But there is more serious agreement. It makes sense, for one thing, of the daunting and seemingly limitless complexity of computational practice—and the fact that, as the field matures, more and more kinds of training, more and more kinds of practicioners, are being drawn into it—from theatre designers to anthropologists to novelists to quantum physicists.

And yet, needless to say, the reconception I am recommending is no small change. Just so that we know what is at stake, let me list just five of its most important consequences.

1. It renders vacuous all statements of the form "computers can (or cannot) do a" (for any a). Will computers be intelligent? Sure—as soon as we figure out what intelligence is, and how to construct it. Will computers be our friends? Yes; if we end up figuring out how to build friends. And not, if not.

   And so on.

   Sorry, Bert.

2. It evacuates the computational theory of mind of all intellectual substance. To say that the mind is computational is nothing more than to say that it is a materially-embodied intentional system. Which we have known for thousands of years.

   Sorry, Jerry.

3. More strongly, it removes the term 'computational' from all interesting theoretical discourse (except, perhaps, from historical studies of engineering).

At this point a personal computer was lifted up from underneath the lectern, placed onto a table, laid it on its side, and draped with a black cloth. At which point the lecture continued…

4. It implies that the mathematical theory known as the "theory of computation"—the theory of Turing machines, effective computability, complexity, etc.—must either be (i)

discarded, or (ii) recognised as in fact being a theory of something else.

Sorry, Dana.

5. It challenges the integrity of computer science departments.

It is good I have tenure.

These are strong conclusions, but I think they are conclusion we must embrace.

5 Conclusion

There are many more things to say. Consider those construals of computing, for example—all those ideas about what characteristic property identified the computational subset, about what it was in virtue of which computation was *special*. I have claimed there is no such subset to be identified. So they failed in their original purpose. Does that mean we should throw them away?

No, they can be resurrected—this is a requiem, after all. But as befits the occasion, the they need to be transformed, in the process.

We do not have much time too look at them now. It turns out, if you do look at them (I cannot resist a few comments!) that each construal rests on a basic, seminal insight—an idea or intuition into the nature of (all or some) intentional systems. But in each case, the construal formulates its insight in particularly stringent terms. In particular, it formalizes or "absolutizes" its insight: forces it into black-and-white, all-or-nothing form. In each case, the absolutist formalisation turns out to be too strong. But if the black and white nature of the formalisation is relinquished—and the goal of identifying a computational subset of intentionality dropped—then the aboriginal insight can be recovered and used in the only remaining project worth doing: developing a general theory of intentionality.

The insight underlying formal symbol manipulation has to do with the tension between the semantic and the effective-in particular, with how systems have to use what is local and effective in order to behave appropriately with respect to distal situations that they are not causally engaged with. In that form, this is an

unbelievably general and important insight, that we should never lose sight of. In formal guise, though, it ends up claiming that the semantic and the effective are independent—which as we have seen, is far too strong.

Even more interesting is the insight underwriting the "theory of computation" so favored in theoretical computer science. What it turns out to be, on this reconstruction, is a theory of pure mechanism. It is not a theory of computing—not just because there is no such thing as computing for it to be a theory of, but also because it does not deal with the "meaning" half of computing's fundamental dialectic. Rather, it is a theory of the *flow of effect*—of how states and state changes, arranged in architectures and processes, can be affected by and themselves produce behaviour. What we call a theory of computation or computability, that is, is neither more nor less than a general mathematical theory of causality. It does not look like a theory of causality, because the quest for formality has led it to be formulated in a way that is totally abstract. But once we let go of that conceit, we can see that what it is really doing is dealing with is the architecture of cause and effect, at a slightly more abstract level than in terms of the physics of concrete devices.

But as I say, these stories—and the work they involve—must be left for another time. For make no mistake: adjusting our theories to accommodate these changes will occupy us for at least another twenty-five years. Students should rest assured. As far as I can tell, most of the intellectual work remains to be done.

What I want to close with is a theme that has been with us since the beginning: there is more in the blood and bones of working programmers than has yet been formulated in language that other cognitive sciences can understand. In a way, I think of twentieth century computing as semiotic alchemy: a rag-tag bunch of practices, thick with inarticulate pre-theoretic knowledge, rich and disorganised-practices that, in spite of their distraction of turning web pages into gold, nevertheless contains within them the seeds for revolutionary theory-practices that, like fourteenth and fifteenth century alchemy, will, once those theories are developed, be largely forgotten, perhaps even shunned, until sometime, around the twenty-third century, someone in science studies

writes a doctoral dissertation arguing that in fact we knew more back here in the twentieth century than the twenty-first and twenty-second centuries thought.

Buried in these practices are powerful intuitions—intuitions about architecture and implementation, about ontology and abstraction, about programs and processes and behaviour and state, about mechanism and effectiveness and how to exploit the tiniest otherwise irrelevant portions of atomic structure to set up long-distance correlations with distal states of affairs. Synthetic practices—not of studying or theorising, but of building. Practices sobered by the humility that comes not from merely thinking you understand something, but from actually trying to construct it-and thereby encountering the fact that you understand virtually nothing at all.

These practices, still under explosive development, lie in wait for philosophers and psychologists to mine. Just do not think they are a subject matter; just do not think we are doing something special. *Forget the original c-word!* For ironically, the idea that computing was really supplying an idea—that computation was a legitimate, autonomous, subject matter, that the term 'computational' denotes a scientifically interesting property—has been the major impediment blocking our appreciation of the vast intellectual importance of these synthetic developments.

Or maybe I can put it positively.

Only if we understand that there is no such thing as computing will we be in a position to appreciate computing's monumental impact on our intellectual life.