CSci • B607 **Week #1 (b)** **Jan 11, 2001**

 Last edited Thursday, January 11, 2001

Construals

I. Review

A. Last lecture
   1. On Tuesday, we didn't make it all the way through the notes.
   2. But that's OK, because the plan, for today and next week, was to go into more detail on the issues that were introduced there, anyway.
   3. So rather than pick up where we left off, we'll proceed directly to the second lecture.
   4. By the end of next Thursday, we will have covered the material that we skipped.

B. Project (review)
   1. As we said last time, the overarching goal is to
      a. Develop a comprehensive theory of **computation in the wild**,
      b. With particular reference to computation's **second promise** (i.e., the *concepts* and *ideas* on which computation rests),
      c. Meeting 3 criteria:
         i. **Empirical:** It should do justice to computational practice (e.g., be capable of explaining Microsoft Word: the program, its construction, maintenance, and use);
         ii. **Conceptual:** It should discharge—or at least own up to—all intellectual debts (e.g., to semantics), so that we can understand what it says, where it comes from, and "what it costs"; and
         iii. **Cognitive:** It should provides a (conceptually) tenable foundation for the computational theory of mind—the thesis, sometimes known as "cognitivism," that underlies artificial intelligence and cognitive science.
   2. Three additional comments help make this project clear
   3. Definitions
      a. First, we pointed out (this is implicit in the empirical criterion) that we will not accept proposals that computation is *defined* in such-and-such a way.
      b. Personally, I don't believe in definitions, so for general reasons I wouldn't be sympathetic to such an approach.
      c. But even if someone were to countenance definitions, and were to proposes a definition of computing, that person would be responsible—for our purposes here—for showing *why that definition captured anything of interest to the notion of computing*.[1]
   4. "Specialness"
      a. Second, (as came up in discussion last time), we said that a theory—to make the

[1] Cf. different versions of set theory. In the philosophy of mathematics, any given definition of a set (predicative, enumerative, etc.) has to defend itself as having something to do with the naïve notion of "set" or "collection."

grade—should identify computation as something "**special**": i.e., as a distinct kind of thing, in the world (what philosopher's might call a "natural kind"[2]

b. One way to say this is to require that, in order for our project to succeed, computation must be (shown to be) a theoretical or intellectual **subject matter**—i.e., to be something described and explained by a powerful, intellectually satisfying theory.

5. Terminology
   a. It is common, in discussing this general topic, to make a distinction among several words: 'computer,' 'computation,' 'computable,' etc.
   b. For example, some people would characterise a "computation" as an abstract thing, and a "computer" as a concrete, physical device on which a computation runs.
   c. It is very important, however, that we *not* commit to any such terminological niceties, at this stage in the project. The reason is that, behind any such discrimination lies a (more or less articulated) theory of the very subject matter we are trying to understand.
   d. In particular, the distinction just formulated—distinguishing computations from computers—builds into it a kind of "abstract–concrete" dualism, of (I will ultimately argue) a very pernicious sort. The whole reason for engaging in a comprehensive foundational project of this sort is to ferret out and examine such presuppositions—to see whether they are warranted, figure out whether they are true.
   e. To commit to a terminological distinction of this sort in advance, therefore, would be to bias the investigation horribly.[3]
   f. Until further notice, therefore, I will skip around between and among such words as 'computer,' 'computing,' 'computation,' 'computational'—without assuming that they mean anything particularly distinct.
   g. Only *after* (if ever) we have accepted a theory of computing will it be legitimate to make a principled distinction between or among any of these.

C. Strategy
   1. Given that overarching goal, the plan is to apply these criteria to a variety of different **construals** of computing—ideas about the nature of computing.
   2. We identified three categories of construal.
      a. Primary: the first group are the ones that have received the lion's share of theoretical attention in the (various) literatures about computing:
         i. Formal symbol manipulation (FSM): the idea, derivative from a century's work in logic and metamathematics, of a machine manipulating symbolic or (at least potentially) meaningful expressions without regard to their interpretation or semantic content;

---

[2]Note to philosophers: I'm not sure (cf. the discussion of meaning and mechanism, next lecture) that there is any reason to suppose that computation is in fact a natural kind—or, more seriously, whether something needs to be a natural kind, in order for there to be a substantive intellectual theory of it. What matters here is that (in order for a theory to be acceptable) it should identify something—viz., computing—as the subject matter of deep, explanatory theory. If, in order to meet that criterion, the phenomenon must be a natural kind, then so be it; computing would have to be a natural kind. But I don't want to make any a priori commitment to that (additional) assumption.

[3]This would be an instance of what, in another context, I have called pre-**emptive registration**.

ii. **Effective computability (EC):** what can be done, and how hard it is to do it, mechanically, as it were, by an abstract analogue of a "mere machine";

iii. **Rule-following (RF)**, or **execution of an algorithm (ALG):** what is involved, and what behavior is thereby produced, in following a set of rules or instructions, such as when making dessert;

iv. **Digital state machines (DSM):** the idea of an automata with a finite disjoint set of internally homogeneous machine states—as parodied in the "clunk, clunk, clunk" gait of a 1950's cartoon robot;

v. **Information processing (IP):** what is involved in storing, manipulating, displaying, and otherwise trafficking in information, whatever that might be; and

vi. **Physical symbol systems (PSSH):** the idea, made famous by Newell and Simon, that, somehow or other, computers interact with (and perhaps also are made of) symbols in a way that depends on their mutual physical embodiment.

b. A second group (though whether these are specifically about *computing* remains to be seen) includes some new ideas, in terms of which people are starting to build and understand systems:

i. **Dynamics (DYN)**

ii. **Complex adaptive systems (CAD)**

iii. **Self-organising systems**

c. Finally, there is a "secondary" group of proposals that, for various reasons, don't get off the ground.

i. We mentioned three types:

α **Demeaning:** computing is *just* … mechanism, machine, artefact, etc.

β **Negative:** computing is *not* … conscious, original, alive, etc.

χ **Derivative:** computing is … abstract, universal, formal, … etc.

ii. Some of these are non-starters—such as derogatory dismissals that computers are "just machines," "only deal in 0s and 1s," or "have only derivative semantics (intentionality)." Others—such as the idea of computing having to do with "universality"—are what I call *post-theoretic*: they are only meaningful (can only be defined) with respect to a prior notion of computing.

3. Today, I want to spend some time going over the six primary ones, in order to give us some feel for what they say—what is at stake—and also for why they are different.

II. Character of the Construals

A. Go back to the six construals

1. Right away, it is evident that they *mean* different things
2. That is, they differ *intensionally*
3. May also different *extensionally*—i.e., apply to different things, be true of different systems

B. Example: *Semantics*

1. FSM: semantic or intentional
   a. Defined in terms of aboutness, because that is what it is to be a symbol
      i. Some computer scientists will object, saying that one can have "purely uninter-

preted" symbols; and that nothing in the theory of computation

ii. Some philosophers of mind, similarly, have tried to deny that FSM involves anything semantic or intentional, taking the "computational" level of description to be different from the "intentional" level of description. I will argue that that is a mistake. If you get rid of the intentionality/semantics (and, as we are doing, get rid of the digitality, which goes into construal #4), then all you are left with is:

— Leads to ⇒ **stuff manipulation**!

b. So in this class we will take the intentional aspect of the FSM claim very seriously

c. I.e., we will take FSM to be *defined in terms of a semantical notion*

d. FSMs needn't be digital, though (or anyway it isn't obvious that they need to be digital)—so the FSM construal might apply to a system that manipulated continuous graphs, or (to take a real example) an analog computers.

2. DSM, contrapositively, is *not* defined in terms of semantics. Would apply to anything whose constitutive states were discrete—such as a system made of Lincoln logs and a stepping motor. Wouldn't have to *mean* or *stand for* anything at all.
3. So FSM and DSM different in extension as well as different in intension.
4. What about #2 and #3: EC, RF, and ALG? Are they intentionally defined or not?
   a. Not immediately clear.
   b. That is something we will want to be very sharp-eyed about, when we go into them.
   c. Actually, in this course will primarily examine only #2 (EC), and then make some brief remarks about #3 (RF and ALG)—unless someone wants to take this issue off-line, do a project on it, and report back to the class.

C. Example: *Constitution vs. Behaviour*

1. Another example of an intensional (meaning) difference: whether the construal focuses on:
   a. *What is done*, without making a claim as to how the machine does it; or
   b. *How it works*, without making a claim as to what such a device can do.
2. The EC construal, for example, is usually understood as a case of the former (what is done)
   a. I.e., focus on *input/output*.
   b. That's what behind the idea that you can analyse programs (e.g.) in terms of *functions*.
   c. It is also implicit in the idea of *equivalence*. To say that two machines are *equivalent in power* (i.e., if you don't include complexity measures) is to say that they are behaviourally equivalent, in (as it happens) a very coarse-grained way.
3. This last is a very important point, which we talked about a bit last time, in answer to a question (◆):
   a. Notions of equivalence are *construal-relative*. So to say, for example, that people are Turing machines, or that we can do things that computers can't, and the like, is to buy into one particular construal (EC).
   b. This is of special interest to philosophy of mind. It has been a long time since anyone took purely behavioural characterisations of mind seriously. Very roughly: the demise of behaviourism in favour of functionalism and the representational theory of mind was a step from a behavioural (I/O) to a constitutive (how it works) stance.
   c. The notion of an algorithm—and equivalence measures based on that (when are two

machines *running the same algorithm*)—starts the move towards how things work, away from what they do. Though I take it the study of algorithms, and their identity conditions, is still an open (though vital) area of research. [4]

4. FSM and DSM construals, in contrast to EC, *do* make claims about how the machine works.

D. Summary
   1. All the six different construals, therefore, mean different things.
   2. Equivalence
      a. Even if (for example, via a mathematical proof) it were possible to show that they were in some sense equivalent, therefore—extensionally equivalent, say (i.e., true of or applicable to the same set of systems, in this or perhaps every possible world)—that would still be a *substantive result*, not a *tautology*.
      b. I.e., even if one were to accept an equivalence proof, however, that would do nothing to lessen the importance of distinguishing them, conceptually.
   3. As we have already seen, however, the six not only different intensionally (conceptually); they also differ extensionally, as well.
      a. That is: they are true of different systems in the world.
   4. It is thus not to strong to say this:

> ◆ The most important step, in getting at the conceptual foundations of computing, is to recognise that these six (and perhaps many others) construals of computing are both intensionally (conceptually) and extensionally distinct.

   5. That fact alone suggests that the foundations of computing need work.

E. Disciplines
   1. Another fact of considerable importance is that the different construals (as we mentioned briefly last time) have different theoretical allegiance in different disciplines
   2. In particular (of most importance to this class):
      a. The FSM construal is far and away the primary conception of computing in philosophy of mind and cognitive science (and to some extent in AI);
      b. The EC construal is, I believe, far and away the major conception of computing held by computer scientists (especially theoretical computer scientists)—though the IP one is gaining currency.
   3. It follows that section II of the course (on FSM) will be more familiar to people whose backgrounds are cognitive and/or philosophical, and section III (on EC), more familiar to those (a majority in this class, as it happens) who come from CS.
   4. Whatever your background, however, it is important to recognise that although you may have one conception (perhaps even an unquestioned assumption) as to what computing is, the person in the seat next to you may have a different one.
   5. Moreover, there are liable to be things that the other construals get at that are important.

---

[4] Cf. the analogous—wholly unsolved, as far as I know—issue in mathematics, about when two proofs are different, and when they are the same.

In fact you may end up learning more from our analysis of construals different from your own.

6. Either way, I believe that there is considerable truth in all of them—truth, though, that we are going to have to dig deep for, before we can get hold of it in a perspicuous and tenable fashion.

III. Information Processing

A. As well as differences between and among the six construals, there are differences *within* them. That is: different *readings* of each.

B. In AOS chapter 2, for example, I distinguish four readings of the information processing (IP) construal (#5):

1. **Lay:** whatever the word "information" means in ordinary English parlance. At least three salient characteristics:
   a. *Abstract*
      i. Independent of medium, language,
      ii. Same information could be conveyed by a note, a smile, a scarf left on a train seat by a spy.
      iii. More abstract than either the concrete details of a particular utterance or a particular state of affairs that the information was about.
      iv. Like "content", lives in a shadowy realm of interpretation
   b. *Veridical* or *authoritative*
      i. It's not information if it's speculation or you just made it up (even if that speculation turns out to be true)
   c. Autonomous, public commodity
      i. Available to anyone
      ii. Not part of a long story, or privately deduced.
      iii. Has to be something of a *commodity:* corpuscular, "mobile" (so that it can be exported into a wide variety of different contexts)
   d. Geoff Nunberg's image: what happens when you take a book by the spine and shake so hard that the words fall out.
2. **Popular**
   a. Whatever underlies such phrases as "information-processing," the "information network", the "Information Age," etc.
   b. No notion of authority or veridicality
      i. Can post anything on the Web, without it being true
      ii. Resultant discord about authority is palpable (cf. discussions of happy news)
   c. Is this the notion that underwrites computing? Doesn't seem right—not the right *kind*. Or at least if it is, computer science will turn into a social and historical theory, rather than scientific or purely intellectual.
3. **Syntactic** or **quantitative**
   a. The theory that started with Shannon & Weaver, ended up moved into computer science through Chaitin and Kolmogorov, is tied to notions of complexity, etc.

b. Undeniably useful
c. But not *semantic*. Started out more as a theory of information (or channel) *capacity*.
d. Still lots of interesting conceptual questions:
   i. How the "space of possibilities" wrt which a given signal carries information, is itself established, communicated, etc.
   ii. How "abstract" or physical the notion is: are the properties that can be information carrying themselves required to be effective or causally efficacious (must they be able to do useful work).

4. **Semantic**
   a. Cf. Fred Dretske, Jon Barwise & John Perry, Stan Rosenshein, Joseph Halpern etc.
   b. Roughly: counterfactual-supporting correlation
   c. **x** carries the information that **y** (where **x** and **y** are propositions or states of affairs—things being a certain way) just in case **x** is the case when **y** is the case, but if **y** weren't the case, **x** wouldn't be the case, either
   d. Examples
      i. Keys in the hotel lobby: the keys are there, meaning that your friend is out; but if your friend were in, they wouldn't be there
      ii. Gasoline indicator gauge

C. This means that an intensive analysis of information processing would have to subdivide construal #5 into 4 **readings** (as shown in figure 1).
D. Same will hold for the other construals. So when we push hard on the FSM construal, for example, will end up subdividing it into two main readings, then a variety of sub-readings under that.
   1. This is an example of what I said last time: we will find that there is a grain of truth—a real insight—underlying each sub-reading (even if the reading or sub-reading to which it has given rise cannot ultimately be accepted, as stated).

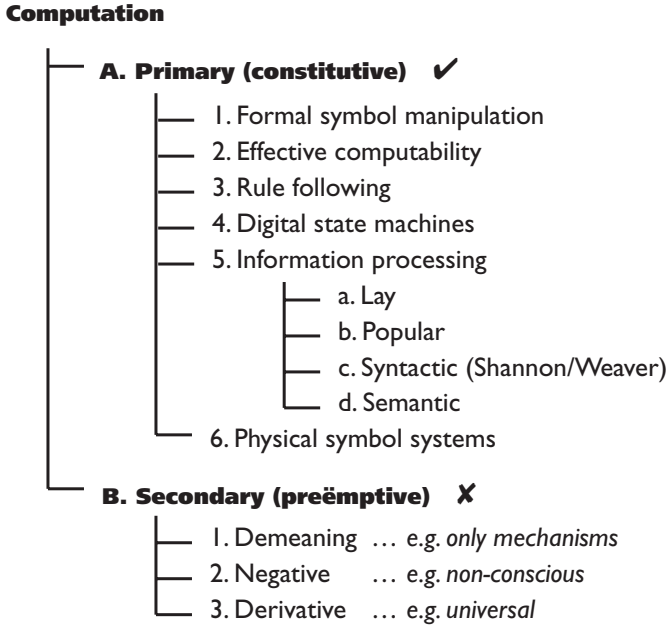


Figure 1 — Subdivisions within construals

IV. Digitality

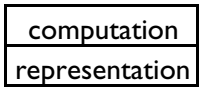
A. Look at one final example, in this introductory way: specifically, the digitality construal.
B. It is a common intuition, on the part of lay people, that AI must be impossible (and cognitive science false) because computers, but not people, are digital.
   1. That is, they instantiate the general form of anti-cog-sci argument that I sketched last time.
   2. That is:

People are continuous
Computers are digital (¬ continuous)

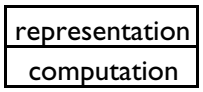
People ≠ Computers

3. Such claims are usually dismissed (and sometimes even treated with some scorn) by people who are computationally trained.
4. Look at this in a little detail, in terms of a debate between two conversants:
   a. *Person on the street (PoS)*, who makes this argument
   b. *Computational sophisticate (CS)*, who denies it

C. First reply: representation
1. Sophisticate: PoS's argument is wrong, because you can represent quantities to an arbitrary level of precision (e.g., in Fortran real-valued variables).
2. But that doesn't work, because the question was not about whether you could use a computer to *represent* thinking.
   a. Rather, the cog sci argument is that a computer can *be* a person or intelligence.
   b. Cf. Searle: you can *represent* the weather, without being the weather (finite-element analysis running on the PowerChallenge on the 4th floor wouldn't get people wet).
   c. In fact can probably represent or model *anything,* including the mind.[5]
   d. The fact that you can represent thinking is probably an interesting fact about *computers*, but it is not a claim about the *mind*.
3. NB: this is an interesting claim: that you can use a computer to model or represent anything. In particular: notion of *modelling*, *representing*, etc.
   a. Is this something that should figure in the comprehensive theory?
   b. If so, then do we need an account of it?
   c. What would that imply vis a vis the second (conceptual) criterion? Does it imply that a theory of computation will rest on a theory of representation?



   d. But what about Newell's claim, that computation would have the glory of defining a notion of symbol for itself:



   e. It's not clear, from what's been said so far, which would be right.
   f. Proposal, introduce something I will call the **master list**
      i. Put in it any word, notion, etc., that we will have to have an understanding of, before we really have what we want—a comprehensive theory of computing:

---

[5]The claim that a computer can represent the mind is (following Searle) often called **weak AI**; everyone (including Searle) believes that that is possible. The claim that a computer can actually *be* a mind is called **strong AI**; that is the thesis about which there is such vociferous debate.

ii. Start with 'computer,' 'computation,' 'computing,' etc.
iii. Also: 'program,' 'process,' 'procedure,' 'algorithm'
iv. Now: 'model' and 'representation'

4. Back to the debate
   a. Moral so far: can have digital representations of continuous phenomena
      i. Right: think of poems ("the endless lapping of the restless sea")—continuous phenomena represented by digital (discrete) words.
   b. But that wasn't the PoS's argument.
   c. Advantage: PoS
   d. Try again.

D. Second reply: analog computing
   1. Can we have continuous computational states—i.e., where the states *themselves*, not just what they mean or represent, are continuous?
      a. Cf. Carver Mead, Jonathon Mills, etc.: analog VLSI
      b. Surely nothing problematic about that.
   2. True—or perhaps true, depending on what computing is.
   3. But have given up DSM, in favour of something else.
   4. That's ok—but that wasn't the original problem. So try again.

E. Third reply: implementation
   1. Sophisticate: We can *implement* (to an arbitrary degree of precision) any continuous phenomenon on a digital substrate
      a. This is genuinely new—and different from the first and second replies—one on condition: that implementation ≠ representation
      b. Test, to tell whether **x** implements **y**, or represents **y**
         i. Blow **x** up
         ii. If **y** was destroyed, the relation was one of implementation
         iii. Otherwise, representation.
      c. Somehow, if you implement **x** as **y** (or on top of **y**, or whatever), then **x** *is* **y**, at a suitable (different) level of abstraction.
         i. Except of course in strange cases (a play in which you "play yourself")
         ii. Add 'implement' to the master list.
   2. Back to question at hand: can you implement continuous processes on top of discrete ones?
      a. Substantive question
      b. Various results in mathematical theory of computing suggest no.
         i. E.g., if you use chaos and turbulence crucially
         ii. Cf. results (Chris Moore, others) that continuous Turing machines can compute functions outside the standard class
         iii. Though there are subtleties: what about accuracy, for example—can you read off the results sufficiently for it to count?
      c. But to be fair, the sophisticate only needs to argue that *some* continuous processes—namely, those that needed for thought—can be digitally implemented.
      d. Maybe—so the sophisticate could argue—you *can* implement those, and so deny the

PoS's claim after all.

3. Does the sophisticate gain the advantage?
4. Depends. Look a little harder.
   a. This is what the sophisticate is claiming (cf. Figure 2)
   b. That is, that: *digitality doesn't cross implementation boundaries upward*[6]*s*
   c. In passing, note that *digitality doesn't cross implementation boundaries downwards*
      i. Cf. hardware: beta-functions of transistors (analysis in terms of Kirkhoff's laws)
      ii. All computers that we use today are analysed, at a sufficiently *low* level, in terms of continuous functions (electronics)

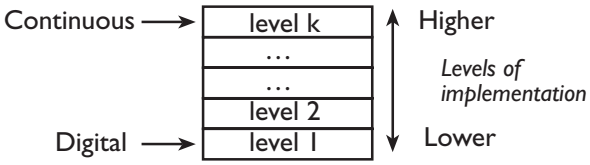


Figure 2 — Implementing Continuity

   d. That implies (an interesting result) that digitality is a *level-specific* notion (like equivalence and identity)
   e. I.e., that is what the sophisticate is committed to.
5. How is sophisticate doing?
   a. Did you have to ask? The answer is … not too well.
   b. There is a logical problem
   c. Up front, they are committed to thesis number 1:
      i. (T1) Digitality does *not* cross implementation (abstraction) boundaries upwards
   d. In the background, however, they are committed to two other theses:
      i. (T2) To be a computer is to be a digital machine (DSM); and
      ii. (T3) Being computational *does* cross implementation (abstraction) boundaries upwards
   e. That is not a possible view. Because T2 requires *property* (intensional) identity. Whereas T1 and T3 claim that the two properties behave differently.
   f. So can't work.
   g. Sophisticate in trouble.
   h. Advantage: PoS.

F. Final try: implementation (2)
   1. One more possibility
   2. Issue is whether *computation* crosses implementation boundaries upwards (Figure 3)

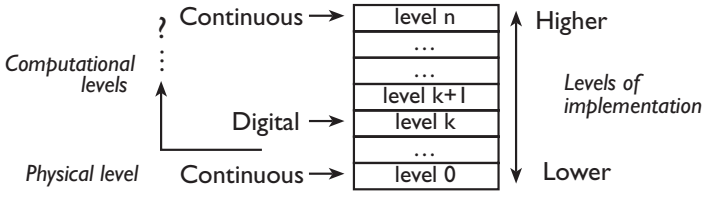


Figure 3 — Computation vs. Implementation

---

[6]We say that a property ϕ **crosses implementation boundaries upwards** just in case if system S is ϕ at level k (i.e., exemplifies or instantiates property ϕ, when analysed at level k), that implies that it necessarily exemplifies ϕ at all levels greater (higher, more abstract) than k. Similarly, ϕ **crosses implementation boundaries upwards** just in case, if S is ϕ at level k, that implies that S necessarily exemplifies ϕ at all levels *lower* (less abstract) than k. For example, being "concretely real" presumably crosses implementation boundaries downwards, since if any system is concretely real at some level of abstraction, it is presumably concretely real at all lower levels of abstraction.

3. Sophisticate changes their definition of computing.
   a. Relinquishes DSM: being computational = being digital
   b. Instead embraces DSM: being computational = being digitally *implementable*
4. But this is an odd position.
5. The question is whether being computational is a *restriction* (constraint)
   a. If it is not a restriction, then "being computational" would become content-free. Computer scientists could go fishing; the computational claim on mind (CCOM—i.e., cognitivism, AI, etc.) would be trivially true; Fodor could retire in disgrace, etc. Not a good outcome.
   b. If it is a restriction, then we need to know what restriction it is (since the sum and substance of the computational claim on mind, for example, would devolve to exactly that question).
   c. One possibility: that you can't implement turbulent (chaotic) processes.
   d. That seems easy enough to imagine: that you can digitally implement any "approximable" process—i.e., any process for which, for any δ there exists ε, such that …
   e. But then the CCOM would amount to a claim that thinking is not turbulent (in spite of what we know from personal experience).
   f. If that is what it amounts to, then that is what we should call it.

V. Conclusion

A. Where have we gotten?
   1. I am not going to answer what I think is *right* (certainly not this early in the course)
   2. What I do want to do is to draw out some morals.
B. First moral: be tough!
   1. This wasn't an especially hard time that I have given the sophisticate (I don't plan to be easy to satisfy)
   2. In this course, as I said last time, we are going to focus on FSM, EC, and (then again) on DSM.
   3. Plan to be fair: I will push on them just as hard as on the sophisticate (harder, in fact—and slower).
C. Second moral: new intellectual debts
   1. Maybe to semantics, representation, modelling, etc.
   2. Also to *implementation*
   3. Have talked about whether digitality crosses implementation boundaries.
      a. Cf. figure 2 what about other properties?
         i. Real-time support?
         ii. Universality?
         iii. Being abstract (virtual)?
         iv. Soundness / correctness?
         v. Semantics in general?
      b. We need a theory of implementation that can give us answers to such things.
      c. So underline "implementation" in the master list.
D. For philosophers: what does all of this have to do with reduction, supervenience, etc.?

1. Very quickly: a (higher-level) theory is *reducible* if it regularities can be expressed in the terms of a lower-level theory.
    a. Lots of variants:
        i. Type-reducible (properties and relations can be directly related)
        ii. Token-reducible (only individual objects)
        iii. Supervenience (no difference at upper level without a difference at the lower level)
    b. What does all this have to do with implementation (in computer science)?
        i. If I construct an implementation of Lisp or Scheme (say in C++), what kind of "reduction" is that?
2. Leads to several more questions:
3. If I can make it, then it is physical?
    a. No—cf. money, STOP signs, etc.
    b. But that raises questions re computation.
    c. Just because we can build a computer, does that mean that it is "exhausted" by its physical reality?
    d. Maybe it's like money; maybe we need society around *in order for a computer to be a computer.*
4. What about the quantifiers?
    a. It seems as if, to implement **x** in **y**, you only have to have *an* organization of **y**, such that **y** is **x** (in the appropriate way).
    b. For theory reduction, though, aren't we supposed to say something about *all* implementations (realisations, reductions, etc.) of **x**?
    c. I.e., is computer science interest in *one* or *any*; philosophy, in *all*?
        i. Cf. Cussins' "construction condition"
5. When we implement, do we understand?
    a. Lots of people think yes. But why?
    b. We can "implement" a kid, without understanding.
    c. Most people would say that building a computer (or programming) is different.
    d. But do we (you) know why it is different? What are we assuming about computers, in thinking that they are not kids? That they are *machines*? If so, what is a machine, such that a computer is one of them? And how do we know that computers *are* machines? Should we add a "theory of machines" to the master list—the list of things we need to understand? Should the c.s. dept require a qualifying exam in machines?

E. Here is a PhD thesis for someone: write a thesis that explains 3 things:
    1. For philosophers: just what implementation, and abstraction boundaries, etc., are.
    2. For computer scientists: what reduction, supervenience, type and token reduction, etc.
    3. Shows how (or whether) the two traditions could be unified.

F. Enough! On Tuesday we'll come back to this whole set of issues, but this time not by looking at the construals themselves, but instead at some of the conceptual issues that are at stake in them: issues like meaning, mechanism, representation, and the like.

—— end of file ——