Effective Computability — Summary

I. Introduction

A. Preliminaries
   1. This is the final week of Part III, on the effective computability construal.
   2. Next week we will start on Part IV, and look at the notion of *digitality*.
   3. This week
      a. We can't really *complete* a full reconstructive analysis of the EC construal: that would take months (or perhaps years, in order to complete a great deal of further research).
      b. Instead, we will (i) summarize what we've seen so far, and (ii) tie up some loose ends.

B. Review
   1. In a sense, we can summarise what we have discovered, in analysing this construal, in terms of three main results.
   2. Ultimately, our target, in looking at this second construal, has been the notion of (pure) **effectiveness**. But we couldn't start there, since we didn't know that that was "what the construal was ultimately about." So instead (with some malice aforethought), we looked at the (surprisingly unanalysed) notion of a *reasonable encoding*.
   3. After some tortuous analysis of the halting problem, we ended up reconstructing the notion of a reasonable encoding in two ways (together, these constitute the first major result):
      a. We turned it "upside down": claiming that the numbers represented the marks (instead of what is normally assumed: that the marks represent the numbers).
      b. We moved it "up":
         i. From being an (object-level) relationship located *within* the primary subject matter
         ii. To being a (meta-level) relationship between the theory and the subject matter, established for purposes of *theoretical modeling*.
   4. This first result—the analysis and reconstruction of the notion of encoding—is extremely important, on its own. But in terms of our overall project, it is only a partial result. What it allowed us to do—i.e., what we were able to achieve, once we had cleared up those issues —was to tease apart the two words ('effective' & 'computing) that constitute the EC label.
   5. In particular, we discovered that the "theory of effective computation" or "theory of effective computability" is not a theory of *computation* at all, but instead a **mathematical theory of effectiveness**, pure and simple. This is the second major result.
   6. The third result had to do with the *nature* of effectiveness, that that theory is a theory of.
   7. In particular, what we saw was that, contrary to the way in which the theory is framed, the notion of effectiveness is:
      a. Not abstract—in the sense in which one might think it is, given how the theory is often

expressed purely mathematically;

b. Not semantic—as you would be led to believe, if you think about it having to do with represented functions, arguments, and values (FAVs)

c. But instead genuinely concrete.

8. That is, it has to do with the *concrete, physical constraints* that govern the rearrangements of physical configurations of concrete, physical systems and machines.

C. Summary

1. In sum, is effectiveness:
   a. Abstract? No (✘)
   b. Semantic? No (✘)
   c. Concrete? Yes (✔)
2. When all these results are put together, they lead to what we are taking as the summary of this whole part of the course.
3. The so-called "theory of computation" = a *mathematical theory of causality*.

D. Plan

1. Today, I want to touch on three topics, in terms of this overall picture
   a. **Programs:** we started to look at programs last week; I want to say a few more things about this (admittedly messy) topic, to try to bring it to some kind of closure.
   b. **Effectiveness & physicality**: some remarks on what it is to take effectiveness to be a physical notion;
   c. **Formality:** how our analysis, in this third part of the course, touches on the general them of formality, which we focused on with such concentration in Part II (in the first critique)
2. Thursday, we will conclude the EC analysis by looking forward at work yet to be done: what it would (or will) be actually to pull off the reconstruction being suggested here: of recasting the theory of effective computability as a (mathematical) theory of physical (causal) efficacy.

II. Programs

A. Intro

1. The first loose end we need to look at is at the notion of a program.
2. What we saw, last week, is that programs are extremely complex entities, which fit in very complex ways into the overall (emerging) picture.
3. That complexity, I believe, is intrinsic to them; it doesn't reflect a simple failure on our part to *analyse* them properly.

B. Structure

1. The simplest picture of programs we used last week is given in Figure 1.
2. Two relationships are of special interest

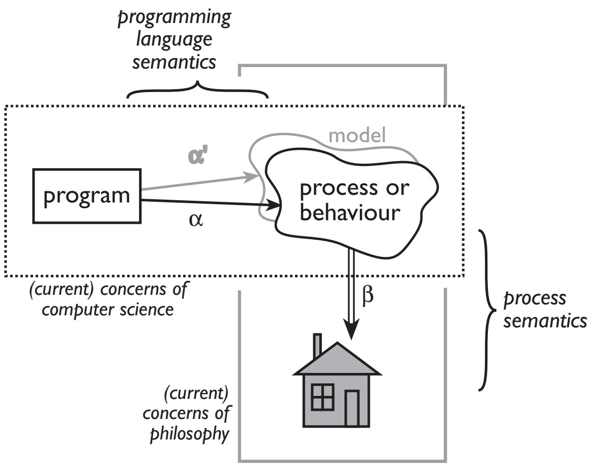

programming language semantics
model
α'
program
α
process or behaviour
(current) concerns of computer science
β
process semantics
(current) concerns of philosophy

Figure 1 — Program and process semantics

   a. α—*program* to *process* (behaviour)
   b. β—*process* (behaviour) to *task domain*
3. The two are subject to almost diametrically opposed conditions:
   a. The **program–process** relation α, from *program* to *process* or *behaviour* (or α', the version that maps programs onto mathematical *models* of processes or behaviours) must be **effective**.
      i. That's why we've indicated it *horizontally*, with a single-tailed arrow ('→')
   b. The **process-world** relation β, according to what we extracted from the first (FSM) critique, is at least sometimes, and perhaps always, **non-effective**.
      i. That's why we've indicated it *vertically*, with a double-tailed arrow ('⇒')
4. Ironically, however—in spite of these differences—*both relations are called "semantic"* in their respective traditions. Because of this, I sometimes call them:
   a. **Program semantics**—for the program–process relation α; and
   b. **Process semantics**—for the process–world relation β.
5. However, as we also said last time, figure 1 is misleading, in that it doesn't indicate the relationship between the *phenomenon* of program–process relations and the *theory* of those program–process relations (of the sort that we have claimed that the theory of effective computability is a theory of).
6. So a better depiction is given in figure 2.
   a. θ_e is a *theory of effectiveness*: that is, a theory of the way in which a program engenders behaviour.
   b. θ_s is a *theory of semantics:* that is, a theory of how the process (behaviour) that the program engenders relates to its target subject matter/task domain.

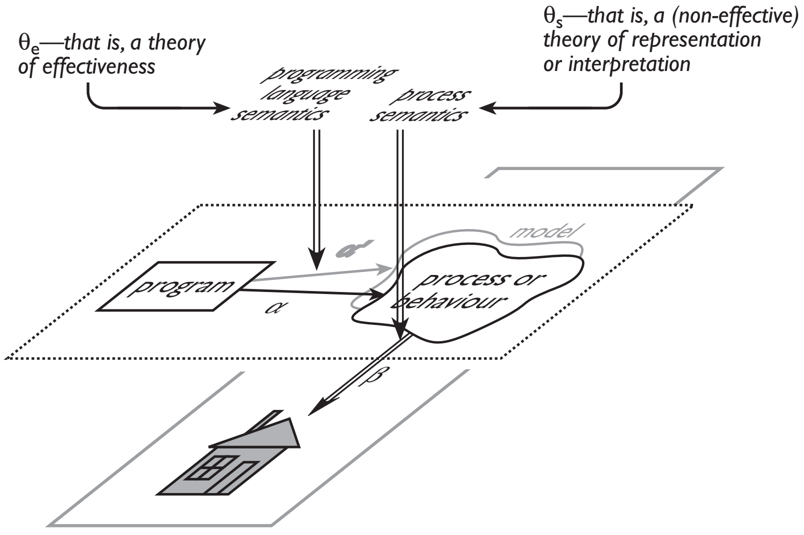

θ_e—that is, a theory of effectiveness
programming language semantics
process semantics
θ_s—that is, a (non-effective) theory of representation or interpretation
model
α'
program
α
process or behaviour
β

Figure 2 — Theory, program, and subject matter

7. What is tricky is to keep the θ_e–α relation and the program–process relation itself (α) distinct, in spite of the fact that they bear certain salient *similarities* (i.e., things true in both cases):
   a. Both need to be *"informationally complete"* in some appropriate sense
   b. Remember that the "target" of the program-process relation α is the *effective behavior*—not the interpretation!—of the (desired or represented) machine or behavior.

C. Discussion
   1. This analysis (i.e., the picture given in figure 2) explains something was indicated, at the beginning of the course, to need explanation: why the word 'semantics' is used in computer

science to designate something that, to a mathematical logician, looks more like *proof theory* or *inference* than like *model theory* or *semantics*.

a. E.g., if you ask a Lisp programmer for the semantics of the '(QUOTE …)' construct, they are likely to reply that "the QUOTE is stripped off and the CADR returned." (which to an outsider seems like a bizarre characterisation of quotation).

b. This is because of "semantic ascent" implicit in notion of a program: when viewed semantically, a program is at *one level of semantic remove* from process it engenders.

c. On the other hand, there is a fundamental contradiction in Lisp-like languages that are imagined to return program-like structures (such as s-expressions). Cf. the semantical reconstruction in 2-lisp.

2. Similarly, can explain things about (the practice of) programming language semantics:
   a. Remember: we (as a scientific community) have no uniform, ontologically secure way of describing processes or behavior.
   b. So we *model* processes or behavior in terms of other things—functions computed, messages passed, transaction files, etc.
   c. This is what generates the difference between denotational and operational semantics: they both analyse the *same* (program–process) relation, but use different models (classifications) of the thereby engendered behaviors or processes.
   d. Explains why operational and denotational account are considered proved *equivalent*.
   e. One wouldn't normally have such an equivalence if one were talking about reasoning (representation) and reference or description.
3. Similarly, this analysis explains the use of constructive mathematics (and intuitionistic type theories):
   a. Those are *constraints from the effective role showing up on the semantic side*.
   b. NB: such effectiveness constraints will not apply to semantic relations in general (such as the process-world relations we called "process semantics")
4. Similarly, the analysis also explains the popularity of Girard's linear logic:
   a. That is, it explains why linear logic
      i. Licenses
         α. $P \wedge (Q \vee \neg Q) \;\Rightarrow\; P$ ✔
      ii. But does not license
         α. $P \Rightarrow\; P \wedge P$ ✘
      iii. But (on the other hand) *does* license:
         α. $\oplus P \Rightarrow\; P \wedge P$ ✔
      iv. Except that a *cost* is associated with the duplication (copying).
   b. From a (genuinely) *semantical* point of view this behaviour is all *unimaginably strange!*
   c. But—to make a contentious claim we'll come back to—it all makes sense, ultimately, once one realizes that linear logic is a theory of *mechanism*, not a theory of *meaning.*

III. Process semantics

A. Intro

1. There are many questions that the foregoing picture doesn't answer

2. For example, it doesn't address the issue of whether the "program semantics" relation α is subject to semantic constraints as well as to effectiveness constraints, as identified here.
3. We will not pursue that question here, at least in that bald form. It would be addressed if we were to turn to the *third* construal: on "rule-following" (RF).

B. Process semantics

1. There is another important question, which would need to be addressed, if we had time to probe deeper into the nature of programs.
2. In particular, what can we say about what we called "process semantics:" the semantic relation (β) between processes and their embedding task domains?
   a. In general—as I have said many times—it need not be effective.
   b. However, the two relations (program–process and process–world) can be conflated under very special circumstances: when the *model* of the process is isomorphic to the *domain*, or there is a homomorphism from domain to model
      i. This most often happens when the computation is itself **context-independent**
      ii. Cf. initial and final models in algebraic semantics (e.g., Goguen and Meseguer)
   c. When do such assumptions fail?
   d. When the process is **context-dependent**!
   e. Context-dependence implies that the process-world relation is one-to-many, which in turn implies that a model of *process* cannot be used as a model of *domain*.

C. Open questions[1]

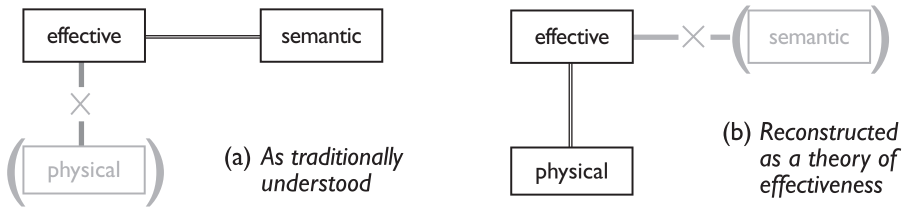
1. How should we go about studying process semantics (process–world relation)?
   a. This is a huge topic, which we don't have time to address here
   b. Among other things, it will involve expanding our (i.e., computer science's[2]) conception of semantics to deal with:
      i. Partial information
      ii. Error and misrepresentation
      iii. Non-effective relations
      iv. Perception and recognition
      v. I.e., general semantics
   c. One question this raises: will anything *specifically computational* be left?
2. What about semantics of *particular programs*, as opposed to semantics of *languages*?
   a. This is how programmers always use identifiers, in their programs (i.e., "move elevator to third floor," "remove employee's building authorization," etc.
   b. Cf. Mike Dixon's "Amala" project
   c. "Open" semantics: user specifies interpretations of inputs, outputs, and identifiers
   d. Changes things: cf. factorial being a *claim,* not a *definition*
   e. Have to move from correctness (of implementation) $\Rightarrow$ soundness
3. Etc. A million other issues

[1]Note: the notes in this section are very telegraphic. I may spend time in class spelling out what is intended, here.
[2]In philosophy, logic, natural language, semantics, etc., the ultimate need for semantics to address these sorts of issue is well recognised.

IV. Effectiveness and physicality

A. Status

1. Rather than focus more on programs, let's turn back to the notion of effectiveness.
2. As we said at the outset, we will spend some time, on Thursday, looking at what it would take, given the results we have come to, to redo or convert the theory of effective computability into a general mathematical theory of effectiveness.
3. We can start on that project today by saying a few words about why doing this would be a good idea


effective
semantic
physical
(a) As traditionally understood
effective
semantic
physical
(b) Reconstructed as a theory of effectiveness

   a. Because effectiveness is a fundamental metaphysical category
   b. Because it cuts deepest of all (twelve) "potency" properties identified in first critique
   c. Because it brings us closer to an integrated theory of nature (by showing how to tie computer science to physics, without foundering on the shoals of 'causation')
   d. Because (it turns out) it stands the test of time as a reconstruction of one leg of intentionality's essential dialectic.
4. In sum: *releasing* recursion theory and the theory of computability from the demands of dealing with intentionality—i.e., from the demands of being a theory of *computing*—is a liberating result. It will make it much easier, when the time comes, to incorporate the insights of this tradition into a full comprehensive theory of computation.
5. Allows it to be something that intellectual history has been wanting: a **general theory of mechanism**.
   a. Cf. the original characterisation of EC: "what can be done by a mere machine"

B. Physicality and semantics

1. To understand what "effectiveness" would come to, on such a reconstruction, consider it in relation to two classically familiar realms
   a. **Physical:** concrete, occurrent, material stuff
   b. **Semantics:** the realm of meaning, reference, truth, etc.
2. The way we have understood EC is indicated in part (a) of figure 3
3. To reduce 50 years of history to a sentence: recursion and computability theory take effectiveness to be *strongly related to semantics*, but *relatively isolated from physics*.
   a. As we said at the beginning of Part III of the course, the (relative) independence from physics is thought to be one of the great achievements of the 20th century: success in "lifting" the account of computability away from the details of physical realization
   b. Connection with semantics: this is necessary, remember, in order to say of it that it is a theory of *computing*
      i. Cf. "computing a function," "deciding a problem," etc.

ii. These are all interpreted notions.

4. The way I am recommending reinterpreting things is as indicated in part (b) of the figure 3.
   a. *Strengthen* the connection with physicality
   b. *Weaken* the connection with semantics
5. I.e., I want to make the following claim:

> ◆ The self-positioning of the theory of effective computability (second construal) is wrong on two counts. The issue it actually "theorises" (i.e., provides us with a theoretical understanding of) is:
> (i) *More* physical than it admits, but
> (ii) *Less* semantical than it claims.

V. Formality

A. Introduction

1. This raises a question about **formality**
2. Background
   a. I said at the beginning that I wanted to use "formality" as a general methodological tool, with which to get underneath all the various construals.
   b. In the first critique, we saw a particular reading of formality: the claim that computation was **independent of semantics**. But that reading was ultimately shown to be **false**.
   c. That is: FSM claimed to be independent of semantics, but wasn't.
   d. On the other hand, whereas that claim was negative, FSM also made a positive claim—one of its prime motivations. In particular (in Haugeland's phrase) it was claimed that it *made the world safe for semantics*.
   e. How so? Because of its efforts to provide a naturalistic grounding for the notion of "syntax" or "shape" or "form."
   f. That is: in its positive formulation, the FSM construal put a direct focus on the effective: on "shape", syntax, etc.—what in general we called "**potency**"
   g. Didn't have a *theory* of such properties (that's why we directed it to this 2nd account)
   h. Nevertheless, potent properties (syntax, grammar, shape, etc.) are widely recognized to be *physical* properties. Everyone in philosophy takes "syntactic" properties to be a (proper) subset of the physical properties.
3. In sum: FSM
   a. **Distances itself from semantics**
      i. This was its formality claim
   b. But it **aligns itself with physical embodiment** (at least with respect to syntax)
4. But now look at the second construal (EC). It:
   a. *Aligns* itself with semantics, in the sense of claiming to be a theory of *computing* (cf. its adjacency to logic, deduction, etc.)
   b. But *distances itself* from embodiment
      i. This is *its* formality claim!

5. I.e., there is an *ironically symmetrical situation: with two independence claims:*

	Construal	**Independence claim**	**Reading of formality**
1	Formal symbol manipulation (FSM)	*Independent of semantics*	Syntactic (antisemantical)
2	Recursion-theoretic theory of effective computability (EC)	*Independent of embodiment*	Abstract

a. Corresponding claims
   i. FSM: get *semantics* out of the way, and everything will be fine (take care of itself)
   ii. EC: get *embodiment* out of the way; & everything will similarly be fine

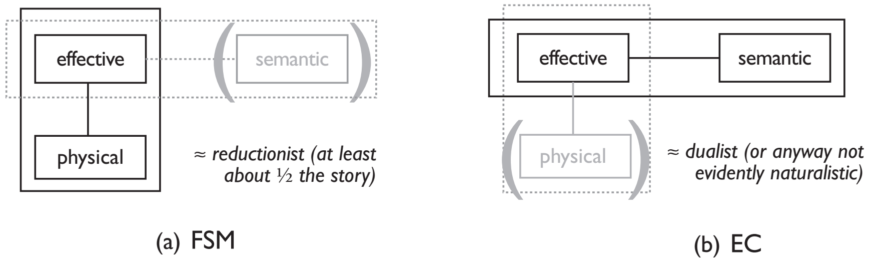

effective
semantic
physical
≈ reductionist (at least about ½ the story)
(a) FSM
effective
semantic
physical
≈ dualist (or anyway not evidently naturalistic)
(b) EC

Figure 4 — Metaphysical biases of the first two construals

b. Corresponding metaphysical biases:[3]
   i. FSM: while setting semantics aside (though recognizing it to be present), it pushed towards a **reductionist** account of how things work
   ii. EC: contrapuntally, set embodiment aside, doesn't worry about the ontological status of numbers, functions, semantics, etc. This is much more **dualist** (at least not evidently naturalistic).
c. Corresponding *fears* kept at bay

[3]In fact the situation is more complex (and even more ironic) than this (admittedly glib) caricature suggests. For it turns out that both formulations (FSM and EC) can support both metaphysical commitments at the same time.

a. The physicalist or reductionist can take 'syntax' to refer to the physically effective, and 'semantics' to refer to the behavioural (causal) consequences of that physically effective (in the "programming language semantics" sense), and thus adopt the terminological structure of the entire FSM, logical, and recursion-theoretic tradition in a way that is wholly compatible with their underlying metaphysical biases.
b. By the same token, the abstract dualist—who, it should be said, stakes a better historical claim to the use of the vocabulary—can employ the *very same terminology*, taking 'effectiveness' to be an abstract (grammatical) property, and similarly taking 'semantics' to refer to something genuinely transcendent of the "merely material," and thereby leave *their* metaphysical biases intact as well.

So there is a facade of agreement—without any deeper consonance on what is being said.

As it happens, it will be a condition of the form of theory to which I am headed that *there will be no room for such metaphysical variation in interpretation (by different theorists)*. Part of the job of a theory of computing—or of general significance, as it will by then be—is to take a stand on such issues. Because computation is inherently intentional, that is, theories of computation cannot be metaphysically neutral. Radical theories cannot be liberal.

   i. FSM: Reductionists, looking at computing through their antisemantical glasses, believe that the (potentially mysterious) threat of the potentially non-effective semantics has been banished, and thus feel reassured that the formal tradition has captured what matters to them about physicalism.
   ii. EC: Dualists, by the same token, interpreting the same tradition through abstract glasses, can believe that the distractions of the body have been kept appropriately out of view, and feel assured, for their part, that the formal tradition captures the essence of what is right about a metaphysical orientation of disembodiment.
  d. These alignments are indicated in figure 4.

6. Critique
  a. In both cases, the analysis is much the same: *the formality assumption won't fly.*
  b. First critique: we showed that its formality claim—the claim that behaviour was independent of semantics—was false.
   i. Computing is *much more interdependent with semantics than* FSM *admits*.
  c. Second critique: we have claimed that the independence (of embodiment) claim is false.
   i. Computing is *much more dependent on physical realization than* EC *admits.*
  d. Ultimately, I will argue that this is no coincidence: that the two interpretations of the two construals are far more similar (in terms of underlying metaphysics) than it looks.
   i. Hint: it is not just that their common allegiance (but dual readings) of formality betrays what they have in common: **formality is what they have in common**.
   ii. But this is an extraordinarily abstract sense of "formality"
   iii. We will get to unpack this more at the end of the fifth construal (on **digitality**).
   iv. Indeed, that is why we will be turning to digitality next. Of all of the construals, there is a sense in which it cuts the deepest.

—— *end of file* ——