Copeland and Proudfoot on Computability

Michael Rescorla

Abstract: Many philosophers contend that Turing's work provides a conceptual analysis of numerical computability. In (Rescorla, 2007), I dissented. I argued that the problem of *deviant notations* stymies existing attempts at conceptual analysis. Copeland and Proudfoot respond to my critique. I argue that their putative solution does not succeed. We are still awaiting a genuine conceptual analysis.

§1. Deviant notations

Many philosophers contend that Turing's (1936) work provides a conceptual analysis of our intuitive concept *number-theoretic computability*. In (Rescorla, 2007), I dissented. I argued that Turing captures the intuitive concept's extension, but I questioned whether he successfully analyzes the concept itself. My argument exploited the phenomenon of *deviant notations*.[1] If we impose a deviant interpretation upon numerals manipulated during Turing computation, then Turing machines can "compute" intuitively non-computable numerical functions. For instance, given any set X of natural numbers, there is a semantic interpretation d_X relative to which some Turing machine "computes" X's characteristic function. One can avoid such deviant cases by

---

[1] For earlier discussion of deviant notations, see (Shapiro, 1982).

demanding that the semantic relation between numerals and numbers itself be intuitively computable. But this response introduces a circularity, since it elucidates intuitive computability of numerical functions by presupposing intuitive computability of semantic interpretations. I critiqued various attempts at avoiding this circularity. I argued that those attempts either engender circularity or else frustrate the aims of conceptual analysis in other ways.

Copeland and Proudfoot respond to my critique: "the problem of deviant encodings is easily solved --- and indeed *was* solved by Turing" (2010, p. 247). I will argue that the putative solution offered by Copeland and Proudfoot does not succeed. The proposal avoids circularity, but it fails in other ways. We are still awaiting a genuine conceptual analysis.

§2. The blank tape restriction

In (Rescorla, 2007), I employed what one might call *the input-output conception of numerical computation*: to compute a numerical function, one begins with a symbolic input (representing the function's input), and one executes a mechanical procedure that converts the symbolic input into a symbolic output (representing the function's output). I applied the input-output conception to Turing computation. I imagined a Turing machine that begins computation with a numeral inscribed on its machine tape (representing the input to a numerical function), eventually halting with a numeral inscribed on the machine tape (representing the numerical function's output).

Copeland and Proudfoot suggest that we should abandon this picture of computation. They endorse what they call *the blank tape restriction*, which demands that Turing machines begin computation with a blank tape. The input-output conception violates the blank tape restriction, by assuming that computation begins with a non-blank tape. Thus, as Copeland and

Proudfoot observe, the blank tape restriction precludes the particular deviant computations I considered in (Rescorla, 2007). Copeland and Proudfoot replace the input-output conception with what one might call *the enumeration conception*: a Turing machine begins with a blank tape and sequentially enumerates symbolic representations of a numerical function's outputs. As they note, Turing himself employs the enumeration conception.

The enumeration conception describes a class of legitimate numerical computations. But so does the input-output conception. Indeed, virtually all recursion theory textbooks employ the input-output conception. They describe Turing-computation of numerical functions as converting symbolic inputs (hence, non-blank tape configurations) into symbolic outputs (Rogers, 1987, pp. 13-16), (Soare, 1987, pp. 11-13). If we impose the blank tape restriction, then we must radically revise contemporary practice within recursion theory. I see no intuitive basis for such a revision. On the contrary, numerous intuitive algorithms (e.g. the grade school multiplication algorithm) describe how to convert symbolic inputs into symbolic outputs. Thus, I see no intuitive basis for demanding that computation begin with null input. The blank tape restriction is unacceptable, when regarded as a constraint upon computation *in general*.

More importantly, the blank tape restriction does not solve the problem of deviant notations. It avoids the particular examples discussed in (Rescorla, 2007). However, as Copeland and Proudfoot admit, one can readily construct alternative deviant notations that circumvent it (p. 251). Even if one employs the enumeration conception, there exists a deviant interpretation relative to which some Turing machine "computes" an uncomputable function.

§3. Turing's Notational Thesis

For this reason, Copeland and Proudfoot supplement the blank tape restriction with a further constraint on computation. Once again, they extract this constraint from Turing's original discussion. *Unary notation* uses a string of *n* strokes to denote the number *n*. As Copeland and Proudfoot note (p. 251),

(*)   A number-theoretic function is computable just in case a Turing machine (with an initially blank tape) is able to print, in order, unary representations of all the values of the function, separated by 0s.

Copeland and Proudfoot call this encoding scheme *Turing-unary notation*. They propose that we analyze intuitive computability as Turing-computability relative to Turing-unary notation. To support their analysis, they articulate a doctrine that they call *Turing's Notational Thesis* (TNT):

Any job of work that can be done by a human computer engaged in numerical calculation can be carried out equivalently by a human computer employing Turing-unary notation.

They describe TNT as "the foundation upon which Turing's definition of a computable number-theoretic function rests" (p. 251).

In (Rescorla, 2007, p. 264), I critiqued a closely related proposal, geared towards the input-output rather than the enumeration conception. My critique evidently did not convince Copeland and Proudfoot, though they do not say why exactly they found it lacking. In any event, here are my reasons for rejecting the specific proposal offered by Copeland and Proudfoot.

I do not doubt that (*) is true. I question whether it yields a conceptual analysis. A conceptual analysis must achieve more than extensional adequacy. It must "capture the meaning" of the target concept. To illustrate, say that a numerical function is *Kleene-computable* iff we can obtain it from the primitive recursive functions through function composition and licensed application of the unbounded leastness operator μ. A numerical function is Kleene-computable

iff it is Turing-computable relative to unary notation.[2] *Church's thesis* states that a numerical function is intuitively computable iff it is Turing-computable relative to unary notation. Assuming Church's thesis, Kleene-computability is an extensionally adequate characterization of intuitive numerical computability. For many mathematical purposes, it is a very useful characterization. However, few commentators would recommend Kleene-computability as a conceptual analysis of the intuitive concept. Few would claim that it captures the meaning of the intuitive concept. Our question is whether (*) fares any better at capturing that meaning.

A basic problem here is that diverse notations yield equally true analogues to (*). For instance, a Turing machine might employ binary notation, Arabic decimal notation, Roman numeral notation, or any other linear notation. We need merely expand the "tape alphabet" accordingly. Copeland and Proudfoot provide no compelling reason to privilege unary notation over these alternative notations.

More carefully, let us say that a notation for the natural numbers is *intuitively computable* iff there is a mechanical procedure for computing which number each notational element denotes. Of course, executing such a mechanical procedure requires an ability to represent numbers, either in one's own thought or through some concrete notation. All the standard numerical notations employed throughout human history are intuitively computable. On the other hand, the deviant notations d_X considered in (Rescorla, 2007) are not intuitively computable.

I assume that all relevant notations are injective. I also assume a special symbol "¶" that is absent from each relevant notation N. For each notation N, I say that a Turing machine computes a numerical function in *Turing-N notation* iff the machine successively prints the function's values, as represented in notation N and separated by "¶"s. For uniformity, I tweak the definition of Turing-unary notation so that "¶" rather than "0" separates successive numerals.

---

[2] This result follows readily from Kleene's Normal Form Theorem.

Clearly, this tweak does not bias my case. Copeland and Proudfoot propose that we analyze intuitively computability as:

Comp(**unary**):   *Turing-computability relative to Turing-unary notation.*

But we could just as well propose one of the following analyses:

Comp(**binary**):   *Turing-computability relative to Turing-binary notation*

Comp(**Arabic**):   *Turing-computability relative to Turing-Arabic-decimal notation*

Comp(**Roman**):   *Turing-computability relative to Turing-Roman-Numeral notation*

or, more generally,

Comp(**N**):   *Turing-computability relative to Turing-N notation,*

for any linear, intuitively computable notation N. We can easily show that these rival analyses Comp(N) are extensionally equivalent:

Lemma: If notations N_1 and N_2 are linear and intuitively computable, then any function that satisfies *Comp*(N_1) also satisfies *Comp*(N_2).

Proof sketch: Since both notations are intuitively computable, there is a mechanical procedure for translating between them. By an analogue of Church's thesis for purely syntactic functions, we can construct a Turing machine that executes the translation: given some element from N_1 as input, the machine yields as output the corresponding element of N_2. (In any given case, we can directly construct this Turing machine without citing Church's thesis.) Now suppose that Turing machine T computes numerical function f relative to Turing-N_1 notation. In other words, T enumerates f's values in notation N_1, separated by "¶"s. Modify T by adding an additional tape (tape 2). Modify T's program as follows. Our modified machine executes T's original program

on tape 1, enumerating f's values. But whenever the scanner prints "¶" after some completed numeral, the new machine interpolates the following steps:

It copies the immediately proceeding numeral from tape 1 to the next free location on tape 2.

It uses the translation algorithm from N_1 to N_2 to replace the N_1-numeral on tape 2 with a corresponding N_2-numeral.

It prints "¶" on the next cell on tape 2.

It transfers back to tape 1 and continues executing T's original program.

The result is that the machine enumerates N_2 numerals for f's output values on tape 2, as it simultaneously enumerates N_1 numerals for f's output values on tape 1. Thus, the modified machine computes f relative to Turing-N_2 notation (on tape 2). □

Given the lemma, (*) implies an analogous principle for every linear, intuitively computable notation N:

A number-theoretic function is computable just in case a Turing machine (with an initially blank tape) is able to print, in order, representations (relative to N) of all the values of the function, separated by "¶"s.

Copeland and Proudfoot suggest that TNT supports *Comp*(Unary) as an analysis of intuitive computability. Yet we can isolate infinitely many rivals to TNT, each corresponding to some rival analysis *Comp*(*N*). For any linear, intuitively computable notation N, we can articulate:

TNT(*N*):   Any job of work that can be done by a human computer engaged in numerical calculation can be carried out equivalently by a human computer employing Turing-*N* notation,

which yields TNT for the special case where *N* is unary notation. I am not sure how Copeland and Proudfoot individuate "jobs of work." For instance, if we individuate "jobs" partly through the numerical notation employed during the "job," then TNT(*N*) is clearly false, for *every N*. Presumably, Copeland and Proudfoot individuate "jobs" without reference to the particular numerical notation employed during the "job." Under such an individuative scheme, TNT(unary) seems plausible. But so does TNT(*N*), for any linear, intuitively computable notation *N*.

Copeland and Proudfoot claim that "[t]hose who wish to press the circularity objection against Turing's analysis of computability must now focus their attack on TNT" (p. 252). I disagree. Assuming a suitable individuative scheme for "jobs of work," I accept TNT. Under that same assumption, I also accept TNT(*N*), for infinitely many *N*. My objection is not that TNT is *false*. My objection is that TNT isolates no privileged features that differentiate unary notation from infinitely many alternatives. It suggests no basis for choosing *Comp*(unary) over *Comp*(Arabic), *Comp*(binary), *Comp*(Roman-Numeral), or countless other candidates *Comp*(*N*).

In response to my objection, Copeland and Proudfoot might deny that the putative analyses *Comp*(*N*) are rivals. For instance, they might suggest that *Comp*(unary) and *Comp*(Roman) are trivial variants of one another.

I respond that, even when *Comp*(N_1) and *Comp*(N_2) are extensionally equivalent, they almost always express different meanings. Consider *Comp*(unary) and *Comp*(Roman). Someone might be familiar with unary notation but not Roman numeral notation. Thus, someone might believe that numerical function *f* satisfies *Comp*(unary) without having the conceptual resources to contemplate *Comp*(Roman). Even if one has the requisite conceptual resources, one might rationally believe that *f* satisfies *Comp*(unary) while doubting that it satisfies *Comp*(Roman), or

vice versa. By the classical Fregean substitution test, it follows that *Comp*(unary) and

Comp(Roman) express different meanings.

A similar argument applies to *Comp*(N_1) and *Comp*(N_2), for virtually any linear,

intuitively computable N_1 and N_2. I say "virtually" because some notations are so closely related

that rational doubts about extensional equivalence may be impossible. A plausible example:

unary notation versus a notation that maps a string of $n+1$ strokes to the number n. Such rare

examples aside, rational doubts about extensional equivalence seem possible. Of course, our

lemma shows that *Comp*(N_1) and *Comp*(N_2) *are* extensionally equivalent. But the proof is not

trivial. I took a few hours to formulate it. A rational agent who has yet to formulate the proof

may rationally doubt its conclusion.

An important moral ensues: for virtually any distinct N_1 and N_2, *Comp*(N_1) and *Comp*(N_2)

cannot both be good conceptual analyses of number-theoretic computability. For instance, if

Turing-computability relative to Turing-unary notation is a good conceptual analysis, then

Turing-computability relative to Turing-Roman-numeral notation is not. If *Comp*(N_1) and

Comp(N_2) express different meanings, then they cannot *both* capture the intuitive meaning of an

unambiguous pre-theoretic concept.

Hence, the analyses *Comp*(N) are genuine rivals. They cannot all be correct. Is *any one* of

them correct? Copeland and Proudfoot emphatically endorse *Comp*(unary). In support of this

position, they note that TNT "is fundamental to the differing developments of computability set

out by the three great founders of the subject" (p. 251) --- namely Turing, Church, and Gödel.

I agree that historical and contemporary developments of recursion theory place great

emphasis upon unary notation. Recursion theory textbooks usually employ this notation or else a

fairly trivial variant, such as the mapping from a string of $n+1$ strokes to the number n. Other

potential notations would yield a less elegant theory. However, such observations do not suggest that *Comp*(unary) provides a compelling conceptual analysis. The question is not whether unary notation subserves an elegant mathematical theory of computation. The question is whether unary notation displays a privileged connection to our *intuitive, pre-theoretic* concept of numerical computation. Only if we answer the latter question affirmatively are we entitled to assign unary notation a privileged role in our conceptual analysis. A convincing case must show that unary notation plays a privileged role in securing our representational access to the natural numbers, at least for purposes of number-theoretic computation.

I am doubtful. Unary notation is a poor vehicle for numerical computation. For instance, if forced to compute over a large number as presented in unary notation, any normal human would immediately translate into some more legible notation, such as Arabic decimal notation, scientific notation, etc. So far from occupying a privileged role in our computations, unary representation is basically useless for normal computation involving large numbers.

Copeland and Proudfoot may respond that unary notation occupies a privileged role in the numerical computation of an idealized human agent, if not an actual human. But I see no reason to agree. Unary notation is inefficient, because its demands upon storage space rise alarmingly with the size of numerical inputs. Other notations allow the thinker to represent large numbers much more efficiently. Why should such an inefficient representational scheme occupy a privileged role in even the most idealized numerical computation?

I conclude that *Comp*(unary) is not a good conceptual analysis of numerical computability. There is no clear reason to favor it over countless alternatives *Comp*(*N*). I do not say that *all* putative analyses *Comp*(*N*) are equally compelling. In many cases, an intuitively computable notation *N* may be unnatural or impractical. For instance, Kripke's unpublished

Whitehead Lectures ("Logicism, Wittgenstein, and *De Re* Beliefs about the Numbers") demand

that any acceptable notation satisfy certain complexity desiderata. Kripke's proposal eliminates

many candidate analyses *Comp*(*N*). But it leaves infinitely many other candidates unscathed, so

it does not uniquely favor *Comp*(unary).

§4. What do all numerical computations have in common?

Should we endorse some other analysis *Comp*(*N*)? Arabic decimal notation may occupy a

privileged role in numerical computation, *for those initiated into Arabic decimal notation*.[3] But

the Romans and the Mayans were unfamiliar with Arabic decimal notation. It hardly seems

plausible that Arabic decimal notation occupied a privileged role in *their* numerical

computations. Thus, *Comp*(decimal) does not even aspire towards sufficient generality. A similar

objection applies to other specific candidates *Comp*(*N*).

A numerical function is computable just in case there exists a computation that computes

it. If we want to explain what it is for a numerical function to be comput*able*, we should explain

what it is to *compute* a numerical function. A good analysis of numerical computability should

provide non-circular necessary and sufficient conditions upon computation of numerical

functions.[4] Beginning with Turing himself, most commentators have recognized this

desideratum. Turing's strategy for satisfying the desideratum is to treat numerical computation as

syntactic manipulation of numerals. He offers constraints that purportedly capture any possible

syntactic manipulation by an idealized human computing agent. As Sieg (2009) notes, Turing's

constraints are somewhat too narrow. For instance, Turing focuses entirely on linear arrays of

symbols, whereas actual human computation frequently occurs in two dimensions. Sieg attempts

---

[3] Kripke argues as much in the aforementioned unpublished Whitehead lectures.
[4] This is one reason why Kleene-computability is not a good candidate for a conceptual analysis. The
characterization says nothing about what it is to compute a numerical function.

to generalize Turing's constraints accordingly. For the sake of argument, I concede that Turing's work, as extended by Sieg, yields non-circular necessary and sufficient conditions upon mechanical syntactic manipulation.

Yet a lacuna remains. Syntactic manipulations compute numerical functions only relative to semantic interpretations. Thus, a complete account of numerical computation must offer general constraints not just upon syntactic manipulation, but also upon semantic interpretation. It must provide non-circular necessary and sufficient conditions upon notations that are "legitimate" or "acceptable" for numerical computation. Just as Turing's work illuminates what all mechanical syntactic manipulations have in common, we should illuminate what all acceptable notations have in common. The tradition stemming from Turing does not accomplish the latter task. For instance, Sieg's recent (2009) discussion offers Turing-inspired constraints upon syntactic manipulations executed by an idealized human computing agent. Sieg calls an agent satisfying those constraints a "Turing computor." Sieg then defines: "[a] function **F** is (Turing) computable if and only if there is a Turing computor **M** whose computation results determine --- under a suitable encoding and decoding --- the values of **F** for any of its arguments" (2009, p. 599). Sieg says nothing about what makes an encoding "suitable." He says nothing to differentiate acceptable and unacceptable notations. Thus, he says nothing to rule out deviant notations such as d_X.[5]

---

[5] In response to the foregoing difficulties, Dershowitz and Gurevich (2008, p. 341) suggest that we should model computation directly over the natural numbers, without intercession by syntactic entities that represent the natural numbers. They propose an axiomatization that reflects this methodology. The axiomatization is "notation-free," in the sense that it allows us to describe computations directly over represented entities (such as numbers), not over the notations through we which represent those entities. Dershowitz and Gurevich thereby circumvent the problem of deviant notations. Detailed discussion lies beyond the scope of this note. Briefly, however, Dershowitz and Gurevich assume (pp. 325-327) that any intuitively computable numerical function case can be computed through iterated application of standard "grade school" arithmetical operations: addition, subtraction, multiplication, and so on. I agree that the assumption is correct. However, someone who fully grasps the concept of computability might reasonably contest the assumption. Thus, we should not simply help ourselves to this crucial assumption when providing a conceptual analysis of computability or when trying to establish Church's thesis.

Until we fill this lacuna, we cannot claim to have provided a general account of *what it is* to compute a numerical function. We may have specified non-circularly the class of intuitively computable functions. But we have not specified non-circularly what all numerical computations have in common. The lacuna remains unfilled as long as we restrict attention to a single numerical notation, whether unary, binary, decimal, or otherwise. Numerical computation can employ a wide variety of notations. Restricting attention to a single notation deprives our putative analysis of any pretense to suitable generality. No analysis *Comp*(*N*) even attempts a suitably general demarcation of numerical computations. A general demarcation will address what all acceptable notations have in common.

What do they have in common? At present, I see no compelling way to answer this question without citing intuitive computability (perhaps supplemented with further constraints, such as a Kripkean complexity restriction).

In sum, a conceptual analysis of numerical computability must describe in non-circular terms what all numerical computations have in common. Otherwise, the putative analysis does not elucidate what it is for a numerical function to be comput*able* (i.e. what it is for there to exist a computation that computes the function). No analysis in the vicinity of *Comp*(unary) provides a sufficiently general description of what all legitimate numerical computations have in common. Thus, no analysis in the vicinity of *Comp*(unary) seems promising.

References

Copeland, B. J., and Proudfoot, D. (2010). Deviant encodings and Turing's analysis of
computability. *Studies in History and Philosophy of Science, 41*, 247-252.

Dershowitz, N., and Gurevich, Y. (2008). A natural axiomatization of computability and proof

of Church's thesis." *The Bulletin of Symbolic Logic, 14*, 299-350.

Rescorla, M. (2007). Church's thesis and the conceptual analysis of computability. *Notre Dame Journal of Logic, 48*, 253-280.

Rogers, H. (1987). *Theory of Recursive Functions and Effective Computability*. Cambridge, MA: MIT Press.

Shapiro, S. (1982). Acceptable notation. *Notre Dame Journal of Formal Logic*, *23*, 14-20.

Sieg, W. (2009). On computability. In A. Irvine (Ed.), *Philosophy of Mathematics*. Burlington: Elsevier.

Soare, R. (1987). *Recursively Enumerable Sets and Degrees*. New York: Springer-Verlag.

Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 42*, 230-265.