Beyond Almost-Sure Termination

Thomas F. Icard (icard@stanford.edu)
Department of Philosophy, Stanford University

Abstract

The aim of this paper is to argue that models in cognitive science based on probabilistic computation should not be restricted to those procedures that almost surely (with probability 1) terminate. There are several reasons to consider non-terminating procedures as candidate components of cognitive models. One theoretical reason is that there is a perfect correspondence between the *enumerable semi-measures* and all probabilistic programs, as we demonstrate here (generalizing a better-known fact about computable measures and almost-surely halting programs). One practical reason is that the line between almost sure termination and non-termination is elusive, as well as arbitrary. We argue that this matters not only for theorists, but also potentially for a learner faced with the task of inducing programs from experience.

Introduction

The metaphor of cognition as computation provides a fruitful and flexible foundation for cognitive science. While computation can be understood broadly to encompass many different paradigms and formats (Rescorla, 2015), it is generally presumed that an upper bound on what can be computed by the human mind is that which can be computed by a Turing machine, or a program in any other *universal* model of computation, such as lambda calculus, recurrent neural networks with rational weights, combinators, Java programs, and so on.

Some early proponents of the computational theory of mind (e.g., Putnam 1967) focused attention on probabilistic computation, allowing randomization in state transitions; and random mechanisms have been central in psychological models going back at least to stimulus-response theory, which had formal connections to probabilistic automata (Suppes, 1969). In recent work, computation with random elements has taken on new significance, where mental representations themselves are characterized in terms of probabilistic procedures or programs (Goodman et al., 2014), and noise is seen not just as a nuisance, but as deeply tied to an agent's capacity for prediction and induction. Although probabilistic machines cannot compute any more functions than deterministic machines, this shift in emphasis raises new and distinct questions. For instance, how expressive is a given class of probabilistic machines for representing useful distributions?

Much of the recent theory of probabilistic computation—particularly that motivated by application to cognition—has focused on *computable* probability distributions, specifically restricting to procedures that terminate almost surely (a.s.), that is, with probability 1. This has given rise to a rich body of work. For instance, it can be shown that the computable distributions correspond to the a.s.-terminating probabilistic Turing machines (see, e.g., Freer et al. 2012), as well as to the a.s.-terminating stochastic lambda terms (Dal Lago and Zorzi, 2012). The limits of computability in the context of conditioning continuous distributions have also been thoroughly investigated (Ackerman et al., 2011).

These important advances notwithstanding, the aim of the present paper is to argue that in cognitive science the focus on a.s.-termination is overly restrictive. As a foundation for theorizing about cognitive processes we should consider the class of all probabilistic computations, not just those that a.s. halt. To use terminology introduced more formally below, cognition should be modeled on the more general class of *enumerable semi-measures*, rather than the smaller class of computable probability measures. We offer two arguments for this claim, one practical and one theoretical.

The practical argument is that the line between a.s. termination and non-termination is elusive and arbitrary. This point is illustrated with a simple example, where the boundary can be studied concretely. The theoretical argument is that the correspondence between the semi-measures definable by probabilistic machines and enumerable semi-measures is more basic and canonical than that between measures definable by a.s.-terminating machines and computable measures. We give a simple, self-contained proof of this first correspondence (Theorem 1), which subsumes the second as a special case (Corollary 1). This proof is elementary, and is arguably simpler than direct proofs of the corollary. We also discuss some connections to program induction, and possible repercussions for probabilistic inference.

Background on Probabilistic Computation

Consider any universal language for describing computations. Allowing programs in one of these languages access to an unlimited source of iid samples from a Bernoulli(0.5) distribution brings us to the setting of universal *probabilistic* languages. For instance, a Turing machine might have an additional read tape with an infinite sequence of random bits, while a lambda term might make use of a choice operator $\oplus$, where $M \oplus N$ reduces to M or N, each with probability 0.5. Just as the Church-Turing Thesis states that any two reasonable deterministic models of computation will be equivalent, one might hypothesize that any reasonable way of adding fair coin flips to these models will give rise to an equivalent model of probabilistic computation. For the rest of this paper we will remain neutral about which of these versions we adopt.

Non-termination, even for very simple, e.g., monitoring processes, is of course a desirable feature of many mechanisms involved in control, where inputs are continually processed (Botvinick and Cohen, 2014). However, our interest here is non-termination even for stand-alone programs without input, so we restrict attention to this setting.

A probabilistic program π, in any machine language, generates an output w—let us suppose outputs are (or at least

encode) binary sequences, so that $w \in \{0,1\}^*$—with some probability, which we will write $\mu_\pi(w)$. That is, $\mu_\pi(w)$ is the sum of the probabilities of all the execution sequences that halt with output w. The program π implicitly represents a distribution on binary strings; however, this distribution may not be a proper probability distribution on $\{0,1\}^*$, as it may be that $\sum_w \mu_\pi(w) < 1$. This will happen if the program fails to halt with some probability $1 - \sum_w \mu_\pi(w) > 0$. A function μ for which $\sum_w \mu(w) \leq 1$ is called a (discrete) *semi-measure*, and it is called a *probability measure* if this holds with equality.

A semi-measure μ is *(computably) enumerable* if it is approximable from below; that is, if for each $w \in \{0,1\}^*$ there is a computably enumerable weakly increasing sequence $q_0, q_1, q_2, \ldots$ of rational numbers, such that $\lim_{i\to\infty} q_i = \mu(w)$. Most semi-measures are not enumerable, but for any probabilistic program π, the semi-measure μ_π will be enumerable. To approximate $\mu_\pi(w)$ from below, consider the set W_i of strings v with length $l(v) \leq i$, such that π accesses (exactly) the bits of v before terminating with output w. Letting $q_i \stackrel{\Delta}{=} \sum_{v \in W_i} 2^{-l(v)}$, it is then evident that $\lim_{i\to\infty} q_i = \mu(w)$. Theorem 1 below states the converse of this observation, that in fact every enumerable semi-measure μ is μ_π for some π.

A semi-measure μ is called *computable* if it is enumerable and for each w there is also a computably enumerable weakly decreasing approximating sequence $q_0, q_1, q_2, \cdots \to \mu(w)$. There are computable semi-measures that are not probability measures, but every enumerable probability measure is computable: we can enumerate the sum $\sum_{w' \neq w} \mu(w')$ by dovetailing to obtain $q_0^*, q_1^*, q_2^*, \ldots$, and then $1 - q_0^*, 1 - q_1^*, 1 - q_2^*, \ldots$ converges from above to $1 - \sum_{w' \neq w} \mu(w') = \mu(w)$. Corollary 1 below states that the computable probabilities measures are exactly those of the form μ_π for some a.s.-terminating program π (see, e.g., Freer et al. 2012; Dal Lago and Zorzi 2012).

In addition to encompassing the space of randomized algorithms, probabilistic programs are of special interest in cognitive science because of their ability to provide compact representations of quite complex distributions, e.g., over combinatorially rich spaces (Goodman et al., 2014; Piantadosi et al., 2016). By encoding these distributions only implicitly through the program's objective probability of returning different outputs, they make an attractive candidate for plausible representations of subjective probability (see Icard 2016 for discussion). Moreover, it is often possible to define programs for automatically representing *conditional* distributions, and thus to apply and adapt the tools of Bayesian statistics to this setting (Tenenbaum et al., 2011; Freer et al., 2012).

Why Non-Terminating Programs?

The enumerable semi-measures form a larger, and arguably more natural, class than the computable measures, but what is the reason to include them in our study of cognitive agents?

The claim of this section is that there is a tension between allowing rich, interesting programs and ensuring those programs a.s. terminate. It is well known that testing whether a deterministic program halts is an undecidable (Σ_1^0) problem, and verifying a.s. termination of a probabilistic program is of even higher complexity (Π_2^0, see Kaminski and Katoen 2015). It follows that the only way to ensure a.s. termination is to restrict to smaller, controlled fragments. This is confining both for the cognitive scientist proposing psychological models, and for the learning agent who may need to construct and induce programs on the fly.

Between Termination and Non-Termination

The boundary between a.s.-terminating programs and non-terminating programs often looks quite arbitrary. To illustrate, we use a very simple example close to recent work on intuitive physics (e.g., Sanborn et al. 2013; Battaglia et al. 2013). This work models people's ability to understand and predict physical events using probabilistic programs for constructing internal "simulations" that operate in rough accordance with physical laws. The example here is far less sophisticated, merely concerning speed along a single spatial dimension. Consider a proverbial tortoise-and-hare scenario, where the tortoise is moving ahead at a constant rate following a slight head start, and the erratic rabbit is nonchalantly racing to catch up. We might suppose that the hare leaps forward some random distance about every fourth step that the tortoise takes. The question is when, if ever, the hare will catch up. Imagine this prediction arising from mental simulations of something like the following program $\pi^\downarrow$:

```
t = 5; h = 0
while (h < t):
  t = t + 1
  if (flip(0.25)):  h = h + Uniform(1,7)
return t
```

For instance, a person observing a rabbit chasing a tortoise might extract a program like $\pi^\downarrow$ in order to make predictions about what will happen some number of steps later.

Where H_k is the distance traveled by the hare at stage k, consider the random variable $X_k = (5+k) - H_k$, measuring the extent of the tortoise's current lead. It is easy to show that the sequence $\{X_k\}$ forms a random walk martingale, and specifically that $\mathbb{E}[X_{k+1} \mid X_1, \ldots, X_k] = X_k$, and so $\mathbb{E}[X_k] = 5$ for all k. By the recurrence property for symmetric random walks, we reach $X_k \leq 0$ at some stage k almost surely. Thus this program $\pi^\downarrow$ halts with probability 1. (Cf. Chakarov and Sankaranarayanan 2013 for powerful a.s.-termination proof techniques that cover examples like this.)

While $\pi^\downarrow$ as written terminates, small changes in the parameters of the program lead to positive probability of non-termination. For instance, if `t` is instead incremented by $1+\varepsilon$ at each step, or if the increase in `h` is drawn uniformly from an interval $(1, 7-\varepsilon)$, for $\varepsilon > 0$, then the resulting program $\pi^\uparrow$ may not halt because the expected distance between the tortoise and hare constantly increases. In particular, there will be a constant $C > 0$, such that for any fixed x_k, we have $\mathbb{E}(X_{k+1} | x_k) - x_k = C$. Hence $\mathbb{E}(X_{k+1}) = \mathbb{E}(\mathbb{E}(X_{k+1} \mid X_k)) = \mathbb{E}(X_k) + C$, from which it follows $\mathbb{E}(X_k) = 5 + kC$ for each k.

This means that the long run expected value of X_k is infinite, and the program fails to halt with some positive probability.

One might suspect that this theoretical distinction could have practical repercussions. Would we not want some guarantee that our program would eventually halt? The problem with this line of thought is that, from a practical perspective, non-termination is not any worse than eventual termination but only after an inordinate amount of time. Simulating the program $\pi^{\downarrow}$ above—and terminating computation whenever the number of steps reaches an upper bound of, say, 10^7—we see that the program reaches this upper bound about .01% of the time. Though a large majority ($\sim$75%) of runs end within 100 steps, the empirical average runtime is in the tens of thousands.[1] Thus, in some small number of cases we would presumably have to terminate computation anyway. From this simulation perspective, the behavior of $\pi^{\uparrow}$, taking $\varepsilon > 0$ to be very small, is empirically nearly indistinguishable. The fact that some of these runs might never terminate is immaterial, practically speaking.

This argument is about possible non-termination, and it does not distinguish between computable and merely computably enumerable distributions. If we increment `t` by a computable real number $1+\varepsilon$, then, though $\pi^{\uparrow}$ might never halt, $\mu_{\pi^{\uparrow}}$ is actually a computable semi-measure, with a computable probability of not halting. However, this situation again may be practically no different from a situation in which $1+\varepsilon$ can only be approximated from below. This paper is a plea mainly for non-terminating programs, and one could in principle accept non-termination but still insist on computability. There may be contexts where insistence on computability may be appropriate (see the section below on conditioning); the claim of this paper, however, is that we ought not make this restriction in general.

A Remark on Levels of Analysis

The argument that $\pi^{\downarrow}$ and $\pi^{\uparrow}$ are practically indistinguishable assumes that we may have to terminate computation beyond a certain point no matter which one we run, and that the resulting behavior will look nearly indistinguishable. A possible objection at this stage is that by enforcing an upper bound on computation time, we are effectively only considering programs that a.s. (in fact, *always*) halt anyway. That is, the larger program encompassing both the simulation model itself and whatever controls the simulations always terminates after a bounded amount of time.

This objection is fine as far as it goes, but it undercuts the motivation for considering rich, e.g., recursive, probabilistic programs to begin with. When we write the program $\pi^{\downarrow}$ above in Java, for example, we understand it as encoding an abstract procedure that could in principle run for any amount of time, even though we know no concrete implementation of $\pi^{\downarrow}$ has this property. Indeed, $\pi^{\downarrow}$ abstracts away from many details about how the program might be implemented. The idea that we can construe some psychological models in a similar manner is very familiar in cognitive science (Marr and Poggio, 1976). Characterizations of mental phenomena using grammars, recursive constructions, and other devices that license unbounded computations are legitimized by potential gain in conceptual clarity and modularity. We understand `while`-loops, models of Newtonian mechanics, and so on, in a very general way: we have a good sense of what they can do, what problems they can be used to solve, and how they can be combined with other tools to form even more powerful devices. From this perspective it is unsurprising that such devices would make their way into our cognitive models.

Such issues about levels of analysis are beyond the scope of this paper. The present suggestion is simply that the best arguments favoring liberal use of a.s.-terminating probabilistic machines as cognitive models should extend to the class of all probabilistic machines. Just as there may be practical reasons to avoid computable, but algorithmically intractable, procedures in practice, so it may make sense in many cases to avoid use of procedures that might not terminate. That does not delegitimize their use in cognitive modeling.

Program Induction

The argument up to this point has been largely negative, that there is no reason to exclude effective semi-measures as possible components of a cognitive model. But there also may be good positive reasons to include them when we consider the learning problem of *inducing* programs from observations (see, e.g., Lake et al. 2017 for application of this idea to human cognition). Given the high complexity of verifying a.s.-termination, the learner seems to be faced with a dilemma: either restrict search to a small fragment of possible programs or risk hypothesizing programs that may not halt.

For example, it is difficult to imagine a sufficiently flexible class of programs—say, a class built out of a few primitive constructions such as `while`-loops and simple arithmetical operations like addition—from which one could easily obtain the program $\pi^{\downarrow}$ above, but not one of the variants $\pi^{\uparrow}$ that might have some probability of not halting. It is not that one would prefer to construct $\pi^{\uparrow}$ over $\pi^{\downarrow}$, but that they are equally preferable and that separating them in a principled way might be difficult, no matter what method is used to perform the induction. That is, even if the goal is to construct an a.s.-halting program, flexibility in program construction might require the possible construction of non-halting programs.

In light of this possibility, a natural suggestion is to consider enriching program induction frameworks with more expansive classes of programs. Consider Bayesian approaches to program induction. Where $\mathcal{C}$ is some class of (semi-)measures on a space $\mathcal{X}$, e.g., on $\{0,1\}^*$—so that we can ask about $P(X)$ for any $P \in \mathcal{C}$ and $X \in \mathcal{X}$—we could have a prior measure ν over $\mathcal{C}$ that induces a mixture distribution P_ν on $\mathcal{X}$:

$$P_\nu(X) \quad \triangleq \quad \sum_{P \in \mathcal{C}} \nu(P)\, P(X).$$

[1] It is even possible for an a.s.-terminating program to have infinite expected run time. Consider a program defining a geometric distribution that repeatedly flips a fair coin until first flipping a heads after n steps, then continutes for 2^n more steps thereafter.

Thanks to Theorem 1, we can always think of each element of $\mathcal{C}$ as a semi-measure μ_π defined by a probabilistic program π from some class Π, so ν defines a prior on programs in Π.

Hierarchical Bayesian models fit this description, where $\mathcal{C}$ is typically a parametrized family of distributions and ν is a hyperprior over those parameters (though hierarchical models may include more levels), and they are often explicitly encoded as probabilistic programs. Provided one can define appropriate likelihood functions $\nu(Y|P)$ and $P(Y|X)$, it makes sense to condition such a mixture distribution on data Y using Bayesian inference:

$$P_\nu(X|Y) \quad = \quad \sum_{P \in \mathcal{C}} \nu(P|Y)\, P(X|Y). \tag{1}$$

By updating ν alongside candidate ground-level distributions P, such methods capture effects of learning at multiple levels of abstraction, such as the ability to transfer general principles inferred in one domain to novel but related domains.

Probabilistic programs generalize hierarchical Bayesian models to allow wider classes $\mathcal{C}$ of measures. For instance, work by Piantadosi et al. (2016) considers learning in a context where $\mathcal{C}$ is defined by logical expressions of the sort typically used in natural language semantics. Evidently, there is no reason we could not consider classes that include enumerable semi-measures as well. An alluring possibility is to take $\mathcal{C}$ to be the class of *all* enumerable semi-measures—i.e., all programs—with ν assigning a weight to each. Because $\mathcal{C}$ is then computably enumerable, there are many effective semi-measures ν assigning positive weight to all probabilistic programs, and even here P_ν is guaranteed to be an enumerable semi-measure, and thus definable by a program. Learning in this setting is somewhat fraught (see below), but at least such a semi-measure can be represented. By contrast, when $\mathcal{C}$ is the class of computable measures there is no computable ν with support exactly $\mathcal{C}$, since that set is undecidable.

Simplicity Bias

In this broader setting of program induction, as in hierarchical models, it is presumed that a good prior on $\mathcal{C}$ is one that favors *simpler* hypotheses. This might be achieved, for instance, by defining ν with a probabilistic grammar so that shorter programs are automatically given higher probability. A very general proposal for biasing simpler functions, known as Solomonoff induction, is based on ideas from Kolmogorov complexity. In brief, the proposal is to assign probability to a string w in proportion to the shortest (deterministic) program that, when run on a universal Turing machine U, produces w as output. The intuition is, data that could be produced by simpler mechanisms should be a priori more likely.

As an aside, there are other applications of simplicity-based constructions inspired by Kolmogorov complexity that make use of enumerable semi-measures. As an example, in their generalization of Shepard's Universal Law of Generalization, Chater and Vitányi (2003) assume enumerable "confusability" probabilities, $P(R_a|S_b)$—specifying how likely it is that a subject will give a response appropriate to a when presented with b—to develop a notion of similarity between arbitrary representations. The basic idea is that similarity is roughly proportional to the length of the shortest (deterministic) program that would be required to transform one representation into the other. Enumerability is exactly what is needed to derive (a generalized version of) the Universal Law.

But what about simplicity-based Solomonoff induction? There are several issues with Solomonoff induction (including the variant here for semi-measures, due to Zvonkin and Levin 1970). One well known problem is that the resulting prior is very sensitive to the choice of universal Turing machine U. In fact, it has been shown that there is a perfect correspondence between weightings ν on the class of enumerable semi-measures and choices of universal Turing machines U for the Solomonoff prior (Wood et al., 2011). In other words, the class of Solomonoff priors just is the class of mixture semi-measures P_ν in which ν assigns positive weight to all the enumerable semi-measures. It is therefore questionable whether this framework really does provide a foundation for understanding simplicity, since it is not clear what an "unbiased" choice of U or ν would be (see Sterkenburg 2016).

A larger problem with Solomonoff induction, however, concerns the complexity of conditional inference. Whereas each Solomonoff prior is itself computably enumerable, conditioning on data leads to a function that is not even enumerable (specifically, we go from Σ^0_1 to Δ^0_2, see Leike and Hutter 2015, cf. also the next section). Since the whole point of Solomonoff induction is to learn, this is a disappointing result. In particular, it means that no probabilistic program can represent a conditioned Solomonoff prior. Given the centrality of induction, we would like to understand better what we can do with conditioning, and whether Bayesian program induction is even possible when some of the candidate programs have positive probability of not halting. This leads us to the next section.

Conditioning Enumerable Semi-Measures

The fact previously mentioned—that effective semi-measures are not closed under conditioning—appears problematic, at least for Bayesian applications of probabilistic programs. It is especially noteworthy given that the computable measures are closed under conditioning in the discrete setting. While a full discussion of conditionalization is beyond the scope of this paper, it is worth briefly clarifying the issue. (Of course, for non-Bayesian approaches to learning programs, e.g., Neelakantan et al. 2016, this may not even be problematic.)

Conditioning an effective semi-measure may produce a function that is only "limit computable" (Leike and Hutter, 2015), meaning the conditional probability of each string can be approximated, but the sequence of rationals need not approach its limit (even weakly) monotonically. The intuition behind this is clear. To determine $\mu(X|Y)$ we must compute $\mu(X,Y)/\mu(Y)$. If all we can do is approximate each of $\mu(X,Y)$ and $\mu(Y)$ from below, and we know nothing about how fast we are converging to the correct values, then we know absolutely

nothing about the ratio $\mu(X,Y)/\mu(Y)$ at any finite stage.

In the computable setting, as Freer et al. (2012) explain, it is straightforward to define a single Turing machine QUERY that takes a program π and a Boolean condition κ (also represented as a probabilistic program), and produces a representation of the posterior distribution $\mathsf{QUERY}(\pi,\kappa)$. The idea is to divide the infinite random bit stream into infinitely many random bit streams, and find the first one that satisfies κ. Then run π using this bit stream to generate an output w. As long as κ a.s. terminates and returns 'true' with positive probability, $\mu_{\mathsf{QUERY}(\pi,\kappa)}$ correctly defines the posterior distribution.

By the aforementioned result, we know there can be no machine that conditions an arbitrary semi-measure μ_π on an arbitrary condition κ. Nonetheless, provided κ stipulates a computable condition, even when μ_π is merely computably enumerable, $\mathsf{QUERY}(\pi,\kappa)$ will correctly represent the conditioned semi-measure, which shows that the conditional distribution is also enumerable. Thus, enumerable semi-measures are closed under conditioning with computable queries.

In this sense the situation for enumerable semi-measures is no worse than that for computable measures: both can be conditioned with computable observations in a uniform way. For many settings this does not seem at all limitative. As a simple example, we could imagine conditioning the program $\pi^\uparrow$ on the statement that the tortoise reached at least 15 steps. This is an easily, indeed finitely, verifiable proposition.

The main limitative result is rather the one we already mentioned: though we can represent complex semi-measures such as Solomonoff priors, the probabilities of even basic observations like "the first object is a 0" are not computable. Nonetheless, we can hope for something in this direction. Sufficient conditions for a conditional mixture semi-measure P_ν to be semi-computable are not terribly stringent. First, the prior ν over semi-measures $\mu \in \mathcal{C}$ should be computable (e.g., this holds if the semi-measures/programs are generated by a probabilistic grammar). Second, as above, the specific data Y must be computably verifiable for each $\mu \in \mathcal{C}$. If both of these are satisfied, then the adapted version of (1)

$$\begin{aligned} P_\nu(X|Y) &= \sum_\mu \Big(\frac{\nu(\mu)\mu(Y)}{\sum_{\mu'}\nu(\mu')\mu'(Y)}\frac{\mu(X,Y)}{\mu(Y)}\Big) \\ &= \frac{\sum_\mu \nu(\mu)\,\mu(X,Y)}{\sum_\mu \nu(\mu)\,\mu(Y)} \end{aligned}$$

is enumerable, and thus representable by a single program, e.g., using an operation like QUERY. Though this falls short of full Solomonoff-style induction, it does generalize what is usually done with Bayesian program induction. It also reveals a distinctive positive reason to entertain specific effective semi-measures as candidate cognitive models. Granted our previous suggestion that it might be beneficial for learning to consider wide classes of programs, putting a computable prior on such a class will result in a enumerable mixture semi-measure P_ν that can be conditioned.

Questions about conditioning in this more general setting clearly merit further attention, especially in relation to more realistic inference methods such as MCMC (see, e.g., Goodman et al. 2008). Algorithmic tractability is an obvious worry, but this is already a worry when everything is computable, and it is not obvious that including effective semi-measures exacerbates the problem. Moreover, even in the computable case, for the continuous setting conditionalization is not in general a computable operation (Ackerman et al., 2011). Consequently, as computational-level models of learning and inference, it appears that the effective semi-measures fare no worse than the computable measures.

Universality

In this final section we offer a proof of the correspondence between probabilistic machines and enumerable semi-measures. Specifically, we show that every enumerable semi-measure can be represented by a probabilistic machine. The proof is similar to proofs for the computable case (e.g., Freer et al. 2012), but we can only use enumerability and must allow for probability of non-halting executions. The exposition is intended to be accessible and intuitive.

Let μ be an enumerable semi-measure on $\{0,1\}^*$. That is, for each word $w \in \{0,1\}^*$, there is a computably enumerable weakly increasing sequence $q_0, q_1, q_2, \ldots$ of rational numbers such that $\lim_{i\to\infty} q_i = \mu(w)$. Assume without loss that $q_0 = 0$. Note then that $\mu(w) = \sum_{i=0}^\infty (q_{i+1} - q_i)$. Our aim is to show that $\mu = \mu_\pi$ for some machine π.

Let $\langle \_,\_ \rangle : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ be a fixed (computable) bijective pairing function with first projection $\rho_1(n) = k$ when $n = \langle k,i\rangle$. Let $w_0, w_1, w_2, \ldots$ be a fixed enumeration of $\{0,1\}^*$, each with a fixed enumeration of approximating rationals $q_0^k, q_1^k, \ldots$ converging from below to $\mu(w_k)$. We define a sequence of rational (thus computable) numbers as follows:

$$\begin{aligned} r_0 &\triangleq q_0^0 = 0 \\ r_{n+1} &\triangleq r_n + \big(q_{i+1}^k - q_i^k\big) \end{aligned}$$

where we assume $0 = \langle 0,0\rangle$ and $n+1 = \langle k,i\rangle$.

Our machine π works in stages, observing a random sequence of bits $a_0,\ldots,a_{j-1}$ while producing an enumeration $r_0,\ldots,r_{j-1}$. At each stage j, we observe a bit a_j and add a rational r_j, then check whether, for any n with $0 \le n < j$, the following condition (2) is satisfied:

$$r_n < \sum_{i=0}^{j} a_i 2^{-i} - 2^{-j} \text{ and } r_{n+1} > \sum_{i=0}^{j} a_i 2^{-i} + 2^{-j} \qquad (2)$$

That is, where $\tilde{p} = \sum_{i=0}^j a_i 2^{-i}$ is the rational generated so far, we know our randomly generated real number will lie somewhere in the interval $(\tilde{p}-\varepsilon, \tilde{p}+\varepsilon)$, and (2) tells us that this interval sits inside the interval (r_n, r_{n+1}). If this holds, output $w_{\rho_1(n+1)}$. Otherwise, move on to stage $j+1$.

Each word w has its probability mass $\mu(w)$ distributed

across different intervals in $[0,1]$. Specifically:

$$\begin{aligned}\mu(w_k) &= \sum_{n:\rho_1(n+1)=k} r_{n+1} - r_n \\ &= \sum_{i=0}^{\infty} (q_{i+1}^k - q_i^k).\end{aligned}$$

The procedure generates approximations $\tilde{p} = \sum_{i=0}^{j} a_i 2^{-i}$ to a random real number, and as soon as we are guaranteed that this random number is in one of our intervals between r_n and $r_{n+1} = r_n + (q_{i+1}^k - q_i^k)$, i.e., that no further bits will take us out of that interval (condition (2) above), we halt and output the string w_k corresponding to the interval, with $k = \rho_1(n+1)$. Clearly, the probability of outputting w is exactly $\mu(w)$, and the probability of not halting at all is $1 - \sum_w \mu(w)$.

Theorem 1. Probabilistic machines correspond exactly with the enumerable semi-measures.

As every enumerable probability measure is computable, we have the following well-known corollary.

Corollary 1. A.s.-terminating probabilistic machines correspond exactly with the computable measures.

Conclusion

Defining distributions by means of programs in a universal probabilistic language yields exactly the computably enumerable semi-measures. Our claim has been that this wider class, going beyond a.s.-terminating programs, provides a sensible foundation for theorizing about representation, inference, and learning in cognitive science. Assuming we want to make a clean separation between computational and algorithmic levels of analysis—which is evidently necessary to justify use of anything beyond (probabilistic) finite-state automata in the first place—we see no reason to restrict attention to programs that a.s. terminate, neither for the theorist nor for the learner.

References

Ackerman, N. L., Freer, C. E., and Roy, D. M. (2011). Noncomputable conditional distributions. In *Proceedings of Logic in Computer Science (LICS)*.

Battaglia, P. W., Hamrick, J. B., and Tenenbaum, J. B. (2013). Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332.

Botvinick, M. M. and Cohen, J. D. (2014). The computational and neural basis of cognitive control: Charted territory and new frontiers. *Cognitive Science*, 38:1249–1285.

Chakarov, A. and Sankaranarayanan, S. (2013). Probabilistic program analysis with martingales. In *International Conference on Computer Aided Verification (CAV)*.

Chater, N. and Vitányi, P. (2003). The generalized universal law of generalization. *Journal of Mathematical Psychology*, 47:346–369.

Dal Lago, H. and Zorzi, M. (2012). Probabilistic operational semantics for the lambda calculus. *RAIRO - Theoretical Informatics and Applications*, 46(3):413–450.

Freer, C., Roy, D., and Tenenbaum, J. B. (2012). Towards common-sense reasoning via conditional simulation: Legacies of Turing in artificial intelligence. In Downey, R., editor, *Turing's Legacy*. ASL Lecture Notes.

Goodman, N. D., Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. B. (2008). Church: A language for generative models. In *Uncertainty in Artificial Intelligence*.

Goodman, N. D., Tenenbaum, J. B., and Gerstenberg, T. (2014). Concepts in a probabilistic language of thought. In Margolis, E. and Laurence, S., editors, *The Conceptual Mind: New Directions in the Study of Concepts*. MIT Press.

Icard, T. F. (2016). Subjective probability as sampling propensity. *Review of Philosophy and Psychology*, 7(4):863–903.

Kaminski, B. L. and Katoen, J.-P. (2015). On the hardness of almost-sure termination. In *Mathematical Foundations of Computer Science*, pages 307–318.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*. forthcoming.

Leike, J. and Hutter, M. (2015). On the computability of Solomonoff induction and knowledge-seeking. In *26th International Conference on Algorithmic Learning Theory*.

Marr, D. and Poggio, T. (1976). From understanding computation to understanding neural circuitry. MIT Memo 357.

Neelakantan, A., Le, Q. V., and Sutskever, I. (2016). Neural programmer: Inducing latent programs with gradient descent. In *International Conference on Learning Representations (ICLR)*.

Piantadosi, S. T., Tenenbaum, J. B., and Goodman, N. D. (2016). The logical primitives of thought. *Psychological Review*, 123(4):392–424.

Putnam, H. (1967). Psychophysical predicates. In Capitan, W. H. and Merrill, D. D., editors, *Art, Mind, and Religion*. Pittsburgh University Press.

Rescorla, M. (2015). The computational theory of mind. In Zalta, E. N., editor, *Stanford Encyclopedia of Philosophy*.

Sanborn, A. N., Mansinghka, V. K., and Griffiths, T. L. (2013). Reconciling intuitive physics and Newtonian mechanics for colliding objects. *Psych. Rev.*, 120(2):411–437.

Sterkenburg, T. F. (2016). Solomonoff prediction and Occam's razor. *Philosophy of Science*, 83:459–479.

Suppes, P. (1969). Stimulus-response theory of finite automata. *Journal of Math. Psych.*, 6(3):327–355.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., and Goodman, N. D. (2011). How to grow a mind: Statistics, structure, and abstraction. *Science*, 331:1279–1285.

Wood, I., Sunehag, P., and Hutter, M. (2011). (Non-) Equivalence of Universal Priors. In Dowe, D., editor, *Solomonoff 85th Memorial Conference*, pages 417–425. LNCS.

Zvonkin, A. K. and Levin, L. A. (1970). The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Uspekhi Matematicheskikh Nauk*, 25(6):85–127.