Probabilistic constructions of computable objects

Andrei Rumyantsev, Alexander Shen*

August 20, 2012

Abstract

A nonconstructive proof can be used to prove the existence of an object with some properties without providing an explicit example of such an object. A special case is a probabilistic proof where we show that the object with required properties appears with some positive probability in a random process. Can we use such an argument to prove the existence of a *computable* infinite object? Sometimes yes: we show how the notion of a layerwise computable mapping can be used to prove a constructive version of Lovasz local lemma. This paper is a survey based on [2, 1, 4].

1 Probabilistic generation of infinite sequences

We want to show the one can use probabilistic arguments to prove the existence of inifinite objects (say, infinite sequences of zeros and ones) with some properties. So we should discuss first in which sense a probabilistic algorithm can generate such a sequence. The most natural approach: Turing machine has an output write-only tape where it can print bits sequentially. Using fair coin tosses, it generates a (finite or infinite) sequence of output bits. The output distribution of such a machine T is some probability distribution Q on the set of all finite and infinite sequences. Distributions of this class are determined by their values on sets Σ_x (of all finite and infinite extensions of a binary string x). The function $q(x) = Q(\Sigma_x)$ that corresponds to the measure Q satisfies the following conditions:

$$q(\Lambda) = 1; \quad q(x) \geq q(x0) + q(x1) \text{ for all } x.$$

(Here Λ denotes the empty string.) Any non-negative real function that satisfies these conditions corresponds to some probability distribution on the set of finite and infinite binary sequences. The output distributions of probabilistic machines correspond to functions q that are *lower semicomputable*; this means that some algorithm, given a binary string x, computes a non-decreasing sequence of rational numbers that converge to $q(x)$.

Now we are ready to look at the classical result of de Leeuw – Moore – Shannon – Shapiro: *if some individual sequence ω appears with positive probability as an output of a probabilistic Turing machine, this sequence is computable.* Indeed, let this probability be some $\varepsilon > 0$; take a rational threshold r such that $r < \varepsilon < 2r$, and consider some prefix w of ω such that $q(w) < 2r$. (Such a prefix exists, since $q(x)$ for prefixes of ω converges to

*LIRMM CNRS, Marseille; on leave from IITP RAS, Moscow. Supported by ANR NAFIT-008-01/2 grant. E-mail: azrumuyan@gmail.com, alexander.shen@lirmm.fr

ε.) Starting from w, we can select the next prefix of ω by finding a son where q exceeds r. The correct direction satisfies this condition, and no branching is possible: if for two sons the value exceeds r, then it should exceed $2r$ for the father.

This result can be interpreted as follows: if our task is to produce some specific infinite sequence of zeros and ones, randomization does not help (at least if we ignore the computational resources). However, if our goal is to produce *some* sequence with required properties, randomization can help: to construct a noncomputable sequence with probability 1 it is enough to output the random bits.

All these observations are well known, see, e.g., the classical paper of Zvonkin and Levin [5]. For a less trivial example, let us consider another result (of N.V. Petri) mentioned in this paper: *there exists a probabilistic machine that with positive probability generates a sequence with two properties:* (1) *it contains infinitely many ones but* (2) *the function $i \mapsto$ the position of i-th one has no computable upper bound.* (In the language of recursion theory, machine generates with positive probability a characteristic sequence of a hyperimmune set.) A nice proof of this result was given by Peter Gacs; it is reproduced in the next section (we thank M. Bondarko for some improvements in the presentation).

2 Fireworks and hyperimmune sequences

Consider the following "real-life" problem. We come to a shop where fireworks of some type are sold. After we buy one, we can test it in place (then we know whether it was good or not, but it is not usable anymore, so we have to buy a new one after that), or go home, taking the untested firework with us. We look for a probabilistic strategy in this game that with 99% probability either finds a bad firework during the testing or takes home a good one.

Solution: take a random number k in 0..99 range, make k tests (less if the failure happens earlier) and then take the next firework. The seller does not get any information from our behavior: he sees only that we are buying and testing the fireworks (when we take the next one home instead of testing, it is too late for him to do something). So his strategy is reduced to choosing some number K of good fireworks sold before the bad one. He wins only if $K = k$, i.e., with probability at most 1%.

Another description of the same strategy: we take the first firework home with probability 1/100 and test it with probability 99/100; if it is good, we take the second one with probability 1/99 and test it with probability 98/99 etc.

Why this game is relevant? Assume we have a program of some computable function f and want to construct probabilistically a total function g not bounded by f if f is total. (It is convenient to consider a machine that constructs a total integer-valued function and then use this function to determine the numbers of zeros between consecutive ones.) We consider $f(0), f(1), \ldots$ as "fireworks"; $f(i)$ is a good one if computation $f(i)$ terminates. First we buy $f(0)$; with probability 1/100 we "take" it and with probability 99/100 we "test" it. *Taking* $f(0)$ means that we run this computation until it terminates and then let $g(0) := f(0) + 1$. If this happens, we may relax and let all the other values of g be zeros. (If it does not, i.e., if we take a bad firework, we are out of luck.) *Testing* $f(0)$ means that we run this computation and at the same time let $g(0) := 0$, $g(1) := 0$, etc. until the computation terminates. If $f(0)$ is undefined, g will be zero function, and this is OK since we do not care about non-total f. But if $f(0)$ is defined, at some point testing stops, we have some initial fragment of zeros $g(0), g(1), \ldots, g(u)$, and then consider $f(u + 1)$ as the next firework bought (testing it with probability 98/99 and taking it with probability

1/99, etc.).

In this way we can beat one computable function f with probability arbitratily close to 1. To construct with positive probability a function not bounded by *any* total computable function, we consider all the functions as functions of two natural arguments (tables), and use ith row to beat ith potential upper bound with error probability $\varepsilon 2^{-i}$. To beat the upper bound, it is enough to beat it in some row, so we can deal with all the rows in parallel, and get error probability at most $\varepsilon \sum_i 2^{-i} = \varepsilon$.

3 Computable elements of closed sets

Let us return now to the original question: can we use probabilistic arguments to construct a computable sequence with some property? As we have seen, if we are able to construct a probabilistic machine that generates some *specific* sequence with positive probability, we can then conclude that this specific sequence is computable. However, we do not know arguments of this type, and it is difficult to imagine how one can describe a specific sequence that it is actually computable, and prove that it has positive probability — without actually constructing an algorithm that computes it.

Here is a more general statement that may be easier to apply: *If F is some closed set of infinite sequences, and if output of some probabilistic machine belongs to F with probability* 1*, then F contains a computable element.* Indeed, consider the output distribution and take a computable branch along which the probabilities remains positive (this is possible since the measure function is lower semicomputable). We get some computable sequence ω. If $\omega \notin F$, then some prefix of ω has no extensions in F (recall that F is closed). This prefix has positive probability by construction, so our machine cannot generate elements in F with probability 1. This contradiction shows that $\omega \in F$.

In the following sections we give a specific example when this approach (in a significantly modified form) can be used.

4 Computable Lovasz Local Lemma

Every closed set F of infinite sequences can be described by an infinite conjuctive normal form:

$$\omega_1\omega_2\omega_2 \ldots \in F \Leftrightarrow P(\omega_{i_1}, \ldots, \omega_{i_k}) \wedge Q(\omega_{j_1}, \ldots, \omega_{j_l}) \wedge \ldots,$$

where each clause in the right hand side excludes some combination of variables used in this clause. To show that F is not empty means to find an (infinite) assignment that makes all clauses true. Lovasz local lemma says that in some cases this can be established by "quantitative" reasons, just because there are few clauses, each involving a lot of variables and therefore only few combinations of values are prohibited.

We do not go here into the details of the full version of Lovasz local lemma (see [4, 2] for more details) and just state some corollary of it as an example. Assume that each clause (disjunction) contains exactly m variables for some m (and therefore is true for random assignment with probability $1 - 2^{-m}$). Assume also that each clause has at most 2^{m-2} neighbors (two clauses are neighbors if they have common variables).

Then Lovasz lemma guarantees the existence of a satisfying assignment.

To formulate the constructive version, we need to assume that the right hand side (infinite CNF) is presented effectively. Namely, we assume that (1) the set of variables is countable and they are indexed by natural numbers; (2) the clauses in the CNF are also

numbered; (3) for each clause, knowing its number, we can effectively compute the clause (the list of variables and the excluded combination of their values); (4) for each variable we can compute a list of clauses that include this variable. (By our assumption, this list is finite.)

Constructive version of Lovasz local lemma guarantees that *under these assumptions there exists a computable satisfying assignment.*

One can indeed prove this fact by showing that closed set of all satisfying assignments has measure 1 according to some computable distribution on the Cantor space of all infinite binary sequences. However, this distribution is constructed not as an output distribution of a randomized Turing machine (though it can be *a posteriori* represented in this way), but as an output distribution of a *rewriting Turing machine*, see the next section.

5 Rewriting machines and layerwise computability

Now we change the model and make the output tape (re)writable: the machine can change several times the bit (0 or 1) in a cell, and only the last value matters. (We may assume that initially all bits are zeros.) The last value is defined if a cell changes finitely many times. Of course, for some values of random bits it may happen that some cell gets infinitely many changes. We say that the output sequence of the machine is not defined in this case.

If the output sequence is defined with probability 1, we get an almost everywhere defined mapping from Cantor space (of random coin sequences) into itself (input sequence is mapped to the limit output sequence). This mapping defines the output distribution on the Cantor space (the image of the uniform distribution on fair coin bits). This distribution may be not computable (e.g., for every lower semicomputable real α it is easy to generate $0000\ldots$ with probability α and $1111\ldots$ with probability $1-\alpha$). However, we get a computable output distribution if we impose some restrictions on the machine.

The restriction: *for every output cell i and for every rational $\varepsilon > 0$ one can effectively compute integer $N = N(i, \varepsilon)$ such that the probability of the event "the contents of cell i changes after step N", taken over all possible random bit sequences, is bounded by ε.*

Lemma: *in this case the limit content of the output is well defined with probability* 1*, and the output distribution on the sequences of zeros and ones is computable.*

Indeed, to approximate the probability of the event "output starts with z" for a k-bit string z with error at most δ, we find $N(i, \delta/k)$ for k first cells, take n greater than all these values of N and simulate first n steps of the machine for all possible combinations of random bits.

An almost everywhere defined mappings of Cantor space into itself, defined by machines with described properties, are called *layerwise computable*. Initially they appeared in the context of algorithmic randomness [3]. One can show that such a mapping is defined on all Martin-Löf random inputs. Moreover, it can be computed by a machine with write-only output tape if the machine additionally gets the value of randomness deficiency for the input sequence. This property can be used (and was originally used) as an equivalent definition of layerwise computable mapping. (The word "layerwise" reflects this idea: the mapping is computable on all "randomness levels".)

6 How the computable version of Lovasz local lemma is proved

As we have said, to prove that some infinite CNF has a computable assignment, it is enough to construct a computable distribution where the set of all satisfying assignments for this CNF has probability 1. This is done by constructing a rewriting machine that defines a layerwise computable mapping, and taking the output distribution of such a machine. (One can prove that for some cases we need to take a layerwise computable machine, the standard machines with write-only output tapes cannot be used, see [1].)

The rewriting algorithm is straightforward. First, we need some ordering on clauses. For each clause we consider the maximal index of variables that appears in this clause, and call it the index of the clause. The requirements for the CNF guarantee that for every index there are only finitely many clauses of this index, so we can write all the clauses in a sequence with non-decreasing indices.

The algorithm starts by assigning random values to all variables. Then it reads the sequence of clauses until it finds some clause that is not satisfied. Then all the variables that appear in this clause are resampled (their values are replaced by fresh random bits).

Moser and Tardos [4] used this algorithm as an efficient probabilistic algorithm for finite Lovasz local lemma. However, as noted in [2], their upper bounds for the resampling probability can be used for the infinite case to prove that this algorithm determines a layerwise semicomputable mapping and therefore to prove the computable version of Lovasz local lemma.

References

[1] Rumyantsev A., *Everywhere complex sequences and probabilistic method*, `arxiv.org/abs/1009.3649`

[2] Rumyantsev A., *Infinite computable version of Lovasz Local Lemma*, `arxiv.org/abs/1012.0557`

[3] Hoyrup M., Rojas C., Applications of Effective Probability Theory to Martin-Löf Randomness. *Lectures Notes in Computer Science*, 2009, v. 5555, p. 549–561.

[4] Robin A. Moser, Gábor Tardos, A constructive proof of the general Lovasz Local Lemma, `http://arxiv.org/abs/0903.0544`

[5] Zvonkin A. K., Levin L. A., 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, 1970, v. 25, no. 6 (156), p. 85–127.