1 Introduction

Program checking was introduced in [2] as a way of certifying the correctness of program output. The original definition applied only to computational problems where correctness could be described as a relation between input and output. The idea was to ensure the correctness of the output of a program (with high probability) on each of its runs rather than on proving the program correct overall. In the process of checking the program on any one run the checker was allowed to make additional calls to the program and was allowed to declare the program buggy if the program produced wrong output on any of the calls.

There have been various extensions of the original program checking model. Blum, Luby, and Rubinfeld [3] defined the notion of checking a library of programs where while checking one program in the library one could make calls to other programs in the library which would themselves be checked as well. Blum et. al. [1] dealt with the problem of checking storage and retrieval from memory, where the specification of correctness of the retrieve operation depended on the state of the main memory rather than just on the input for the operation.

We are interested in extending the concept of program checking to programs that claim to generate objects according to probabilistic distributions.

For example, consider a program that claims to take as input a graph $G$ and produce as output a random spanning tree of $G$, where by 'random' we mean that every tree is equally likely to be generated. On a particular run of the program, it takes a specific graph $G$ and produces a tree $T$. We would like to design a checker that will either certify that $T$ is random in $G$ or declare that the program is buggy on some run. As in the standard checking model we want the checker to work against all programs that claim to generate a random spanning tree in $G$.

**Definition 1** The class of problems $RG$ (standing for random generation) consists of problems $\pi$ such that associated with each input $I$ of $\pi$ there is a sample space with distribution $D_I$ from which the output should be drawn.

We require a uniformity condition that the distribution $D_I$ have a short description in terms of its input. One could define a class of random generation problems in a non-uniform
model and the distinction does affect the checkability of a problem. However in this paper we will restrict attention to the uniform model defined above.

We will say that a probabilistic program $P$ solves $\pi$ if on input $I$ it produces output distributed as $D_I$. Our goal is to check such programs.

This looks like a fairly difficult task especially considering that no polynomial time test is known for the basic problem of certifying that a string of bits has been generated by independent Bernoulli trials with parameter $p$. The above problem of generating binomially distributed sequences of bits, with each bit having a probability $p$ of being a 0, will be called $B(p)$.

In order to make some headway we assume that a checker is available for $B(1/2)$ and ask what other problems in $RG$ can be checked using the checker for $B(1/2)$.

The justification for this approach is two-fold. It is theoretically interesting to study 'reductions' between problems in the program checking setting. For computational problems $\pi_1$ and $\pi_2$ all we know is that in order for a checker for $\pi_1$ to give rise to a checker for $\pi_2$ it is sufficient that $\pi_1$ and $\pi_2$ are polynomially equivalent. However the fact that $\pi_2$ reduces to $\pi_1$ alone does not seem to be sufficient. We would like to prove similar structural theorems in the program checking setting for problems in $RG$.

**Definition 2** For problems $\pi_1, \pi_2 \in RG$ We will say that problem $\pi_2$ polynomial time reduces to problem $\pi_1$ if there is a probabilistic polynomial time transformation which takes an output $O_2$ of $\pi_2$ on input $I_2$ and transforms it into an output $O_1$ of $\pi_1$ on input $I_1$ such that $O_1$ is distributed according to the output distribution of $\pi_1$ on $I_1$ iff $O_2$ is distributed according to the output distribution of $\pi_2$ on input $I_2$.

**Definition 3** We will say that $\pi_1$ and $\pi_2$ are equivalent if there are polynomial time reductions in both directions.

The second motivation for this approach comes from practical considerations. In practice various heuristics are used to certify that a string of bits is the output of independent, unbiased, Bernoulli trials. Reductions of other problems in $RG$ to $B(1/2)$ would allow such well-known heuristic checkers to apply to other problems in $RG$.

We will also consider the more general scenario where $\pi_1$ and $\pi_2$ are problems in $RG$, with $\pi_1$ having a checker and $\pi_2$ having a program $P$ that claims to solve it. Again the question will be, 'Is it possible to use the checker for $\pi_1$ to check $P$?'. In view of the suspect nature of the available randomness it is natural to assume that any reduction between problems in $RG$ be not allowed an independent source of randomness. This is what we will do. Thus in the definition of 'reduction' above, the randomness in the reduction must be derived from the randomness in the output of the program being checked. This assumption has the additional feature that in the spirit of program checking, the reduction (being deterministic) is essentially different from the program $P$ which must be probabilistic.