Reviewer 1: "\mathsf R" refers to the base type for programs which reduce to real-number values. "r.v." means random variable, and “a.s.” means almost surely.

Reviewer 2 said that there were no examples of reduction strategies beyond CBV. As the other reduction strategies we’re mainly considering (i.e. those used with Theorem VI.16) are intended to be designed for particular programs, the other examples are given along with particular example programs, specifically the extended discussions of Examples V.14 and VI.17 in the appendices.

CBN is not considered a valid reduction strategy because it violates the condition that the argument of a beta redex must be a value, which follows from the definition of reduction strategy (p. 10) and redex (Definition VI.1).

Example V.1 is simply meant to illustrate that there are some terms which terminate too slowly to be rankable.

Labelling the samples with elements of a set A simply refers to picking traces s from I^A, then using s(a) for the sample reduction whenever the relevant position (or other label) is a.

Section 6:

The restrictions in reduction strategies are the extra conditions in Definition VI.1 on what reductions are valid (as described in the paragraph after Theorem VI.15).

The body of a Y subterm is effectively both the function and the argument in an application. If it contains multiple occurrences of the relevant variable, it will be duplicated by the Y reduction, potentially causing a similar issue to the case of (\lambda f. f 0 + f 0) (\lambda x. sample) if the sample were evaluated first.

Reading guide for Section 6: In order to be able to use alternative reduction strategies to prove properties of the program defined using the usual reduction strategy, it is necessary first to prove the relation between these different strategies, i.e. to provide a confluent version of the semantics. It is therefore necessary to have some way of relating the samples that will be used in runs of the program using different reduction strategies. Positions are defined as a way of addressing sample statements within a program independently of the reduction order. Next, a version of the reduction relation is presented that is nondeterministic, so that it allows a choice of what order to perform reductions in. The notion of positions is extended to potential positions, for samples which may appear later in the reduction sequence but not necessarily in the initial term. In order to allow potential positions in different reduction sequences to be considered equivalent, a relation ~* is defined, and finally, all of these are used to construct a confluent version of the trace semantics, ⇒, that is still nondeterministic in reduction order, but does specify the outcome of random choices. The desired properties of this semantics (confluence and equivalence to the standard semantics) are proved, then it is used to give extended versions of ranking functions and the related AST results.

Reviewer 3:

Regarding the conjecture, we are unaware of any particularly relevant previous literature. We did spend a while trying to solve the problem, and will try some more in future. Part of what makes this question so difficult is that you have to construct an antitone function that represents the rate of convergence of all of the different states the program could be in, and simultaneously assign all the program states ranking function values. The fact that, for constructing a ranking function, even branches with 0 probability must be considered is an additional cause of difficulty, as they can terminate arbitrarily much slower than the program itself.