RANDOMIZING REDUCTIONS OF SEARCH PROBLEMS*

ANDREAS BLASS† AND YURI GUREVICH‡

Abstract. This paper closes a gap in the foundations of the theory of average case complexity. First, we clarify the notion of a feasible solution for a search problem and prove its robustness. Second, we give a general and usable notion of many-one randomizing reductions of search problems and prove that it has desirable properties. All reductions of search problems to search problems in the literature on average case complexity can be viewed as such many-one randomizing reductions; this includes those reductions in the literature that use iterations and therefore do not look many-one. As an illustration, we present a careful proof in our framework of a theorem of Impagliazzo and Levin.

Key words. Average case, search problems, reduction, randomization.

1. Introduction and results. Reduction theory for average case computational complexity was pioneered by Leonid Levin [?]. Recently, one of us wrote a survey on the subject [?], and we refer the reader there for a general background. However, the present paper is largely self-contained; we recall all necessary definitions.

We develop the foundations of the theory of many-one reductions of search problems in the context of randomizing algorithms. Many-one reductions are easier to use than Turing reductions, but one may wonder how restrictive they are. Indeed, there are cases in the literature on average-case complexity [?, ?] when reductions of search problems to search problems are not many-one; those reductions are iterations of many-one reductions. It turns out, however, that iteration is needed, not for reductions, but only for the resulting search algorithms. Thus, the theory of reductions can be simplified by treating reductions separately from iteration. Our notion of many-one reduction was influenced by the specific reductions used in [?, ?, ?].

As a general framework for the study of average case complexity, we use domains. Definitions of domains and polynomiality on average are recalled in Section 2. Essentially, a domain is a set of strings with a size function and a probability distribution. If X is a subset of positive probability of a domain A then the restriction of A to X is the domain obtained by assigning zero probability to elements of $A - X$ and renormalizing the probabilities on X. The phrase "polynomial on average" is abbreviated to "AP".

We consider search problems of the following sort. A domain A (of instances) is given, along with a binary relation W between elements of A and arbitrary strings in some fixed alphabet Σ. If $W(x, w)$ holds, the string w is a *witness for the instance x (with respect to W) and x is a positive instance; an instance x is negative if it has no witnesses. We assume that there exists an algorithm that, given an instance x of*

* An extended abstract of this paper has been published earlier [?].

† Partially supported by NSF grant DMR 88-01988. Address: Mathematics Department, University of Michigan, Ann Arbor, MI 48109-1003, USA. ablass@umich.edu

‡ Partially supported by NSF grant CCR 89-04728 and ONR grant N00014-91-J-11861. Address: Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI 48109-2122, USA. gurevich@umich.edu

non-zero probability and a witness w for x, computes $W(x,w)$ in time polynomial in $|x|+|w|$ where $|x| = |x|_A$ is the size of x with respect to A and $|w|$ is the ordinary length of w. The search problem SP(A,W) is: Given a positive instance $x \in A$ of non-zero probability, find a "witness" w such that $W(x,w)$.

Let A^+ be the restriction of domain A to positive instances. An algorithm for SP(A,W) is supposed to find witnesses for instances $x \in A^+$ of non-zero probability; it is immaterial what the algorithm does for other instances.

In the context of PTime (i.e. polynomial time) search algorithms, it is reasonable to assume that the length of a witness is bounded by a polynomial of the size of the instance and that $W(x,w)$ is computable in time polynomial in $|x|$, since such an algorithm, given x, could produce only w's that are polynomially bounded with respect to x. Search problem of this kind are called NP search problems; every NP decision problem gives rise to an NP search problem. We shall, however, work, not with polynomial-time algorithms, but with (randomizing) algorithms whose running time is polynomial on average (AP time algorithms). For such algorithms, there may be an occasional input x (and a random string r) for which the running time and the length of the output are very long. Therefore, we do not require that the length of a witness is bounded by a polynomial of the size of the instance or that $W(x,w)$ is computable in time polynomial in $|x|$. However, if we require that the length of a witness is bounded by a polynomial of the size of the instance or that $W(x,w)$ is computable in time polynomial in $|x|$, all our theorems remain true. On the other hand, the condition that $W(x,w)$ is PTime computable relative to $|x|+|w|$ can be relaxed to a hybrid — PTime with respect to w and AP time with respect to x — without affecting the results of this paper [?].

It is often useful to consider search problems where one seeks witnesses w for certain instances but doesn't care what the algorithm does for other instances even if those other instances have witnesses. For example, in the unique search problems, considered in [?], the algorithm applied to an instance x (and a random string) should produce a witness w when w is the unique witness for x, but it does not matter what the algorithm does when applied to an instance for which there is either no witness or more than one witness. Such problems can be incorporated into our framework by replacing A with its restriction to important instances.

One interesting case is when every instance of non-zero probability has a witness. This can be achieved by restricting the domain to positive instances, but there are other ways as well. Consider, for example, the usual Hamiltonian Circuit search problem with some probability distribution on graphs. Not every graph has a Hamiltonian circuit. However, we can require that, in the case of a graph without a Hamiltonian circuit, a witness establishes the nonexistence of a Hamiltonian circuit. For example, a witness may be a computation establishing the nonexistence of any Hamiltonian circuit. Such a witness may be long, but $W(x,w)$ is required to be computable in time polynomial in $|x|+|w|$, not in $|x|$, so long witnesses may be all right. Indeed, there exists an algorithm for a randomized version of Hamiltonian Circuit (with an arbitrary but constant edge probability) that finds a Hamiltonian circuit or establishes the nonexistence of one in linear, in the number of vertices, expected time [?].

Before we explain our notion of feasible solutions, let us recall a useful notion of random function [?]. Deterministic algorithms compute deterministic functions. Similarly, randomizing (coin flipping) algorithms compute random functions. Formally, a random function f on a domain A is a deterministic function on an extension D_f of A whose elements are pairs (x, s) where $x \in A$ and s is a binary string to be regarded as the sequence of coin flips used by the algorithm. In this introduction we allow ourselves to be sloppy about the distinction between a randomizing algorithm and the random function it computes. The definition of random functions is recalled in Section 2.

There are different approaches to defining what constitutes a good randomizing algorithm for a search problem. One approach, closer to the actual use of such algorithms, requires that a correct answer be obtained, with probability equal to 1 or at least very nearly 1, in reasonable time (i.e. in AP time). Another approach requires only that the algorithms have, for every input of positive probability, a reasonably high probability of success, say at least 1%. For many purposes, the two approaches are equivalent, since an algorithm of the second, weaker sort can be iterated with independent random strings to obtain a very high probability of success at the cost of a moderate increase in the running time. A polynomial number of iterations suffices to improve success probabilities from as low as a reciprocal of a polynomial of $|x|$ to as high as exponentially (relative to $|x|$) close to 1. For average-case complexity, the situation is even better. We can start with a success probability whose reciprocal is not polynomially bounded but only polynomial on average, and we can iterate the algorithm to obtain a success probability 1, without increasing the time beyond AP.

To make these ideas precise, we introduce, in Section 3, the notions of AP time randomizing algorithms and almost total randomizing algorithms. An almost total, AP time randomizing algorithm succeeds with probability 1, but an arbitrary AP time algorithm needs only to succeed often enough on each instance. The notion of AP time randomizing algorithms is the central notion of this paper. Roughly speaking, a randomizing algorithm M with inputs in a domain A is AP time, if it has a set of "good inputs" (x, s), where $x \in A$ and s is a finite sequence of coin flips, such that M terminates on every good input, the proportion of good inputs (among all inputs) is at least the reciprocal of an AP function, and the computation time of M on good inputs is not too big. It is not required that the set of good inputs is easily recognizable in any way.

DEFINITION 1.1. *A randomizing algorithm M is an AP time solution (resp. almost total, AP time solution) for a search problem $SP(A, W)$ if M is AP time (resp. almost total, AP time) on A^+ and every terminating computation of M with input $x \in A^+$ of non-zero probability (and arbitrary coin flips) produces as output a witness w with $(x, w) \in W$.*

It might seem more natural to require that an AP time algorithm produces appropriate outputs only for "enough", not all, of the terminating computations, but our definition is somewhat simpler and essentially equivalent: Every AP time algorithm M for a search problem Π that produces correct outputs on its good inputs can be converted to an AP time solution M' for Π.

Choose any iteration technique, subject to some natural fairness and carefulness

conditions specified in Section 4. Theorem **??**, *the main result of Section 4, implies*

THEOREM 1.2. *A randomizing algorithm M is an AP time solution for a search problem Π if and only if the iteration M^* of M is an AP time solution for Π that is almost total.*

Thus, an AP time solution is iterable to an almost total, AP time solution, and this is optimal. In addition, several natural iteration techniques are shown to satisfy the fairness and carefulness conditions in question.

Now let us turn our attention to reductions. What is a (randomizing) many-one reduction of a search problem $\Pi = SP(A, W)$ to another search problem $\Pi' = SP(A', W')$? Such a reduction consists of two parts, the first producing, for each instance $x \in A^+$ of non-zero probability and some random strings s, an instance $x' = f(x, s) \in A'$, and the second producing, for each witness w' with $(x', w') \in W'$, a witness $w = g(x, s, w')$ with $(x, w) \in W$. A reduction and any algorithm M' solving Π' yield, by composition, an algorithm M (equal to $g \circ M' \circ f$ if g depends only on w') that solves Π. We wish to determine conditions on randomizing algorithms f and g to ensure that, when M' has a good average-case solution, namely either almost total and AP time or simply AP time, then so does M. The conditions are to be independent of M', so that a reduction makes sense even when there is no algorithm M' available. Indeed, one of the most common uses of reductions is to prove that Π' is complete in one sense or another and therefore no good algorithm solving Π' is expected to exist. (Also, one may want to require that if M' is an efficient algorithm using some oracle, then M should be a similarly efficient algorithm using the same oracle.)

*The interesting parts of our conditions on f and g concern f; we shall simply require g to be polynomial time. Notice that it would make no sense to weaken this to, say AP time on the set $(\Sigma')^*$ of potential witnesses, because we are not given a probability distribution on $(\Sigma')^*$. Of course, M' together with the probability distribution $\boldsymbol{P}_{A'}$ of A' induces a distribution on $(\Sigma')^*$, but our conditions are to be independent of M'. And if we required g to be AP time with respect to some "natural" distribution on $(\Sigma')^*$, then some algorithm M' might, with high probability, produce witnesses w' from a small set to which our natural distribution assigned low probability and on which g takes very long to compute. Because of this situation, we assume that g is polynomial time, and then we can ignore g in the following sense: M is AP time or almost total and AP time if and only if $M' \circ f$ is. (There is a natural way to weaken the condition of PTime computability of $g(x, s, w')$ to a hybrid — Ptime with respect to w' and AP time with respect to (x, s) — preserving the results of this paper [**?**], but again we stick here to the simpler assumption of PTime computability.)*

*What conditions should f satisfy? Consider first deterministic reductions. Obviously, f should be AP time computable in this case. What else? Nontrivial examples of deterministic reductions may be found in [**?**, **?**]. Actually, the two papers are devoted to NP decision problems, but one can consider the search versions of those problems instead; the deterministic reductions of decision problems naturally transform to deterministic reductions of the corresponding search problems. We tried to give a natural motivation and analysis for the additional requirement (beyond AP time computability)*

in the notion of deterministic reductions.

THEOREM 1.3 ([?]). *For every (deterministic) function f from A to A', the following two conditions are equivalent:*

- *For every AP function T on A', the composition $T \circ f$ is AP.*
- *The function $x \mapsto |fx|_{A'}$ is AP and A' dominates A with respect to f, i.e., the ratio $\dfrac{\boldsymbol{P}_A[f^{-1}\{fx\}]}{\boldsymbol{P}_{A'}\{fx\}}$ is AP on A.*

Thus, if f is AP time computable and A' dominates A with respect to f, then, for every AP time algorithm M' on A', the composite algorithm $M' \circ f$ is AP on A.

This gives rise to the following reduction notions.

DEFINITION 1.4. *A deterministic AP time reduction of a domain A to a domain A' is a deterministic AP time computable function f from A to A' such that A' dominates A with respect to f.*

Since the domination relation is transitive [?], deterministic reducibility of domains is transitive.

DEFINITION 1.5. *A deterministic AP time reduction of a search problem $SP(A, W)$ to a search problem $SP(A', W')$ consists of*

- *A deterministic AP time reduction f of A^+ to $(A')^+$.*
- *A polynomial time computable function g such that if $x' = f(x)$ and w' is a W'-witness for x' then $g(x, w')$ is W-witness for x.*

COROLLARY 1.6. *Deterministic AP time reducibility of search problems is transitive. Further, a search problem Π is solvable by a deterministic AP time algorithm if it is deterministically AP time reducible to some Π' which is solvable in deterministic AP time.*

Unfortunately, deterministic reductions are too weak for many purposes [?] and stronger randomizing reductions were employed in [?, ?, ?, ?]. The task of "cleaning up" the notion of randomizing reductions was the main motivation of this paper.

So suppose that f is a random function from a domain A to a domain A' and let T be an AP time random function on A'. There are four situations, according to whether T is assumed to be almost total or not and whether $T \circ f$ is required to be almost total and AP time or only AP time. It will be easy to see that one of the four situations is absurd; no f can convert arbitrary AP time functions T into almost total $T \circ f$, for if T succeeds with probability $1/2$ for every x', then $T \circ f$ will do no better.

Of the three reasonable situations, the easiest to analyze is when T is assumed to be almost total and $T \circ f$ is required to be almost total and AP time. It is clear that such an f must itself be AP time and almost total, for a computation of f is an initial segment of any successful computation of $T \circ f$, no matter how trivial T is. It turns out that Theorem ?? generalizes nicely in this case.

THEOREM 1.7. *Let f be an almost total, AP time computable random function from a domain A to a domain A' such that A' dominates D_f with respect to f. Then, for every almost total, AP time randomizing algorithm M' on A', the composite algorithm $M' \circ f$ is AP time and almost total.*

Theorem ?? follows from the more informative Theorem 5.1. It gives rise to the following reduction notions.

DEFINITION 1.8. *An almost total, AP time randomizing reduction of a domain A to a domain A' is an almost total, AP time computable random function f from A to A' such that A' dominates D_f with respect to f.*

DEFINITION 1.9. *An almost total, AP time randomizing reduction of a search problem $SP(A, W)$ to a search problem $SP(A', W')$ consists of*

- *An almost total, AP time randomizing reduction of A^+ to $(A')^+$.*
- *A polynomial time computable function g such that if $x' = f(x, s)$ and w' is a W'-witness for x' then $g(x, s, w')$ is a W-witness for x.*

COROLLARY 1.10. *Almost total, AP time randomizing reducibility on search problems is transitive. Further, a search problem Π is solvable by an almost total, AP time randomizing algorithm if it is reducible, by an almost total, AP time randomizing algorithm, to some Π' which is solvable by an almost total, AP time randomizing algorithm.*

Finally, we turn to the most complicated situations, where $T \circ f$ is only required be AP time. This sort of reduction has the advantage that $f(x, s)$ needs to be a reasonable instance of Π' only for so many random strings s. The random function f must be AP time computable. A set Γ of good inputs for f forms a domain in a natural way; see the notion of dilation in Section 2 in this connection.

THEOREM 1.11. *Let f be an AP time computable random function from a domain A to a domain A' with a domain Γ of good inputs dominated, with respect to f, by A'. Then, for every AP time algorithm M' on A', the composition $M' \circ f$ is AP time.*

Theorem ?? follows from the much more informative Theorem 5.2. It gives rise to the following reduction notions.

DEFINITION 1.12. *An AP time reduction of a domain A to a domain A' is an AP time computable random function f from A to A' with a set Γ of good inputs such that A' dominates Γ with respect to f.*

DEFINITION 1.13. *An AP time reduction of a search problem $SP(A, W)$ to a search problem $SP(A', W')$ consists of*

- *An AP time reduction (f, Γ) of A^+ to $(A')^+$.*
- *A polynomial time computable function g such that if $(x, s) \in \Gamma$, $x' = f(x, s)$ and w' is a W'-witness for x' then $g(x, s, w')$ is a W-witness for x.*

THEOREM 1.14. *AP time reducibility of search problems is transitive. Further, a search problem Π is solvable by an almost total, AP time algorithm if it is reducible, by an AP time algorithm, to some Π' which is solvable by an AP time algorithm.*

Proof. To prove the second part of the theorem, suppose that Π' is solvable by an AP time algorithm. By Theorem ??, Π has an AP time solution M. By Theorem ??, M^* is an almost total, AP time solution for Π. QED.

As an illustration of the theory of randomizing many-one reductions, we rewrite in Section 6 Impagliazzo-Levin's proof of a theorem of theirs [?]; we believe that this version of the proof is easier to comprehend.

An extended abstract of this paper has been published earlier [?].

2. Preliminaries. *For the reader's convenience, we recall here some definitions and facts.*

DEFINITION 2.1 ([?, ?]). *A domain A consists of*

- *an underlying set, often called A as well, whose members are strings over some finite alphabet,*
- *a size function, assigning to each $x \in A$ a positive integer $|x| = |x|_A$ called the size of x, and*
- *a probability distribution $\boldsymbol{P}_A$ on A.*

REMARK 1. *In [?], we required that the number of nonzero-probability elements of any given size is finite. The requirement seemed non-restricting and useful. However, it turned out to be too restrictive in the further analysis of randomizing computations, and in this paper we remove it.*

Because the elements of a domain A are strings, we can use the usual computation model based on the Turing machine. Traditional concepts of (worst-case) complexity for such functions are defined by means of the size function $|x|$ which is usually polynomially related to length of strings. For example, polynomial time would mean that there is a polynomial p such that $p(|x|)$ bounds the time needed on input x. Concepts of average-case complexity are defined by averaging with respect to the probability distribution $\boldsymbol{P}_A$.

As was pointed out by Levin [?] and discussed in some detail in [?, ?], the most obvious definition of the concept "polynomial time on average" has inappropriate consequences, and some care is needed to obtain a suitable definition. We use the following definition due to Levin [?], as modified in [?] to allow ∞ as a value.

DEFINITION 2.2. *Let T be a function from a domain A to the set $\bar{\mathcal{R}}^+$ of nonnegative reals augmented with ∞. T is linear on average if $T(x)/|x|$ has finite expectation,*

$$\mathop{E}_x \frac{1}{|x|} T(x) = \sum_x \boldsymbol{P}_A(x) \frac{1}{|x|} T(x) < \infty,$$

and T is polynomial on average, abbreviated AP, if it is bounded by a polynomial of a function that is linear on average. In other words, T is AP if, for some $\varepsilon > 0$,

$$\mathop{E}_x \frac{1}{|x|} (Tx)^\varepsilon = \sum_x \boldsymbol{P}_A(x) \frac{1}{|x|} (Tx)^\varepsilon < \infty \ .$$

We use the convention that $0 \cdot \infty = 0$; thus, an AP function can take the value ∞ but only at points of probability 0.

LEMMA 2.3 ([?]). *The collection of AP functions over a given domain is closed under addition and multiplication.*

A (deterministic) algorithm, taking elements of a domain A as inputs, is polynomial time on average or AP time, if its running time on input x is an AP function of x. We consider the running time to be ∞ if the algorithm fails to terminate, so an AP time algorithm must terminate on all inputs of positive probability.

DEFINITION 2.4. *Consider a set X with a size function. A probability distribution ν on X dominates a probability distribution μ on X if there exists a function g from X to $\bar{\mathcal{R}}^+$ such that g is AP with respect to X and μ and $\mu\{x\} \leq g(x) \cdot \nu\{x\}$ for all x in X.*

The original notion of domination is due to Levin [?]. It was analyzed and generalized in [?] and [?].

DEFINITION 2.5 ([?]). *Let f be a function from a domain A to a domain B. Then B dominates A with respect to f, written $A \leq_f B$ if the ratio $\dfrac{\boldsymbol{P}_A[f^{-1}\{fx\}]}{\boldsymbol{P}_B\{fx\}}$ is AP on A.*

We will use the following easy characterization of domination:

LEMMA 2.6 ([?]). *Suppose that f is function from a domain A to a domain B. Then $A \leq_f B$ if and only if $\boldsymbol{P}_A$ is dominated by a probability distribution ν such that the image of ν under f is $\boldsymbol{P}_B$, i.e. $\boldsymbol{P}_B\{y\} = \sum_{x \in f^{-1}\{y\}} \nu\{x\}$ for all $y \in B$.*

It will be convenient for us to restrict attention to domains satisfying the following proviso.

PROVISO. *$|x|$ is bounded by a polynomial of the length of x.*

The proviso is needed only to derive the following consequence.

COROLLARY 2.7. *If f is an AP time function from a domain A to a domain B, then the function $x \mapsto |f(x)|_B$ is AP.*

Proof. The length of the string $f(x)$ cannot exceed the time needed to compute $f(x)$. Therefore the length of $f(x)$ is an AP function of x. Now use Lemma ??. QED

Instead of adopting the proviso, we could change the formulations of some of our theorems by explicitly requiring that the reducing functions do not increase the size beyond AP. It seemed easier to adopt the proviso and get the problem of size blowup out of the way. We did not come across any need to consider a size function that is not bounded by a polynomial of the length. However, as the following definition shows, we did come across a need to consider a size function that may be much smaller than the length.

The notion of dilation was introduced in [?] and used in [?]. The idea is to combine the probability distribution on instances and the probability distribution on coin flips into one probability distribution. For this purpose, we introduce a new domain consisting of pairs (x, s), where x ranges over instances of the search problem under consideration, i.e., over elements of a given domain A, while s ranges over the possible finite sequences of coin flips used by some algorithm (in its terminating computations on input x). The size of (x, s) is taken to be $|x|_A$, not (as one might at first guess) $|x|_A + |s|$; this is so that the complexity of computations is measured relative to the size of the instance, not the number of coin flips. (If we used $|x|_A + |s|$, then a very inefficient computation could be made to look efficient — e.g., linear time — by appending a lot of unnecessary coin flips to make $|x|_A + |s|$ greater than half the running time.) The probability of (x, s) is, except for a normalization factor, the probability $\boldsymbol{P}_A(x) \cdot 2^{-|s|}$ for independent choices of x and

s. The normalization factor, i.e., the lack of actual independence, can be intuitively understood this way: Given x, the algorithm starts to flip coins (independent of x) until the computation terminates, having used random string s. But it may happen — and the probability of this does depend on x — that the algorithm doesn't terminate, in which case we re-start the algorithm with a new, independent sequence of coin flips. This re-starting leads to the denominator in the probability clause of the following definition.

DEFINITION 2.8. *A dilation of a domain A is a domain* $\Delta \subset A \times \{0,1\}^*$ *satisfying the following conditions, where* $\Delta(x) = \{\, s : (x,s) \in \Delta \,\}$*:*

- *For every x, no member of* $\Delta(x)$ *is a proper prefix of another member of* $\Delta(x)$*.*
- *For every x with* $\boldsymbol{P}_A(x) > 0$*,* $\Delta(x) \neq \emptyset$*.*
- $|(x,s)|_\Delta = |x|_A$*.*
- $\boldsymbol{P}_\Delta(x,s) = \boldsymbol{P}_A(x) \dfrac{2^{-|s|}}{\sum_{t \in \Delta(x)} 2^{-|t|}}$*.*

This definition is more general than the definition of dilation used in [?] because we do not require $\sum_{s \in \Delta(x)} 2^{-|s|} = 1$*. This sum occurs in the denominator of our definition of* $\boldsymbol{P}_\Delta$ *in order to ensure that* $\boldsymbol{P}_\Delta$ *is indeed a probability measure, i.e., that the total probability of* Δ *is 1. We say that a dilation* Δ_1 *of A is a subdilation of a dilation* Δ_2 *of A if* $\Delta_1 \subseteq \Delta_2$*.*

DEFINITION 2.9. *Let* Δ *be a dilation of A. The function*

$$U_\Delta(x) = \frac{1}{\sum_{s \in \Delta(x)} 2^{-|s|}}$$

is the rarity function of D. The dilation Δ *is almost total if* $U_\Delta(x) = 1$ *for every* $x \in A$ *of positive probability. This means that, if we repeatedly flip a fair coin to produce a string of 0's and 1's, then, with probability 1, we shall eventually obtain a string in* $\Delta(x)$*.*

The notion of randomizing (coin-flipping) algorithm motivates a useful notion of random function [?].

DEFINITION 2.10. *A random function on a domain A is a function f on a dilation* D_f *of A. Such a random function f is almost total if the dilation* D_f *is almost total and the probability, with respect to* D_f*, that the value of f is finite equals 1.*

Composition of randomizing algorithms motivates composition of random functions.

DEFINITION 2.11 ([?]). *Suppose that f is a random function from a domain A to a domain B such that* $\boldsymbol{P}_B(f(x,s) > 0$ *whenever* $\boldsymbol{P}_A(x) > 0$*, and let g be a random function on B. The composition* $g \circ f$ *of f and g is the random function h on A such that, for every* $x \in A$*,*

- $D_h(x) = \{st : s \in D_f(x)$ *and* $t \in D_g(f(x,s))\}$*,*
- *If* $s \in D_f(x)$ *and* $t \in D_g(f(x,s))$ *then* $h(x,st) = g(f(x,s),t)$*.*

It is easy to see that the composition of almost total random functions is almost total.

3. Randomizing algorithms. *A randomizing algorithm on a domain A can be formalized as a Turing machine M which takes as input an instance $x \in A$ and has access to an auxiliary read-only tape, called the random tape, containing an infinite sequence r of "random" bits (0's and 1's). The random tape is bounded on the left and unbounded on the right; its head can move only to the right. The set $\{0,1\}^\infty$ of all possible r's is endowed with the probability measure λ that is the product measure derived from the uniform measure on $\{0,1\}$. Both the measure and M's mode of access to the random tape can be described informally by saying that M is allowed to flip a fair coin at various stages of its computation and that these coin flips are independent.*

Consider the computation of a fixed randomizing algorithm M with a fixed input x but with varying sequence $r \in \{0,1\}^\infty$. For each r, let $Read(r)$ (or $Read_{M,x}(r)$ if necessary for clarity) be the initial segment of r that is actually read during the computation. If the computation terminates, then $Read(r)$ is finite; if not, then $Read(r)$ may be finite or equal to r. As the computation, with M and x fixed, depends only on $Read(r)$, it is clear that, if $r' \in \{0,1\}^\infty$ has $Read(r)$ as an initial segment, then the computation using r' is the same as that using r and, in particular, $Read(r') = Read(r)$. Thus, the set

$$R(x) = R_M(x) = \{Read_{M,x}(r) : r \in \{0,1\}^\infty\},$$

has the property that no member of $R(x)$ is a proper initial segment of another. We define the probability measure $\rho = \rho_{M,x}$ on $R(x)$ to be the image, under the function $Read_{M,x}$, of the standard measure λ on $\{0,1\}^\infty$. For any $E \subseteq R(x)$,

$$\rho(E) = \lambda\{r : Read(r) \in E\}.$$

Each $R(x)$ is the disjoint union of the sets $RF(x)$ and $RI(x)$ consisting of the finite and infinite strings in $R(x)$, respectively. The measure ρ gives the probability $2^{-|s|}$ to each $s \in RF(x)$ and agrees on subsets of $RI(x)$ with λ. Every $r \in \{0,1\}^\infty$ either is in $RI(x)$ or has an initial segment in $RF(x)$, namely $Read(r)$. In [?], randomizing algorithms were assumed to terminate for all x with $\boldsymbol{P}_A(x) > 0$ and all r; this ensured that, if $\boldsymbol{P}_A(x) > 0$, then $RI(x)$ is empty and every $r \in \{0,1\}^\infty$ has an initial segment in $RF(x)$ and therefore $RF(x)$ is finite. In the present paper, we consider algorithms that may fail to terminate even on inputs of positive probability, so we must consider both $RF(x)$ and $RI(x)$; furthermore, $RF(x)$ may well be infinite.

If $s \in \{0,1\}^*$ is a finite sequence with no proper initial segment in $RF(x)$, then s occurs (with probability $2^{-|s|}$) as the random string read by some stage in a computation of M on x. If, furthermore, $s \notin RF(x)$, then this computation will read at least one additional random bit, which is equally likely to be 0 or 1. If, on the other hand, $s \in RF(x)$, then the computation will read no additional random bits. Writing "s is a prefix" for the event that the random string actually read by M has s as an initial segment, we have (for future reference) the following

LEMMA 3.1. *The distribution ρ satisfies the equation*

$$\rho(s \text{ is a prefix}) = \begin{cases} 2^{-|s|} & \text{if } s \text{ has no proper initial segment in } RF(x), \\ 0 & \text{otherwise;} \end{cases}$$

furthermore, this equation completely determines ρ.

Proof. Obvious. QED

We restrict attention to randomizing algorithms M *such that* $RF_M(x) \neq \emptyset$ *whenever* $\boldsymbol{P}_A(x) > 0$. *It follows that the set*

$$RF_M = \{(x,s) : x \in A \text{ and } s \in RF_M(x)\}$$

forms a dilation of the domain A.

DEFINITION 3.2. *The random function* F_M *on* A *computed by* M *is the deterministic function on* $D_F = RF_M$ *with* $F(x,s)$ *equal to the output of* M *on* (x,s).

We write $Time(x,r)$ *or* $Time_M(x,r)$ *for the time taken by the computation of* M *on* x *and* r; *if this computation does not terminate, then* $Time(x,r) = \infty$. $Time(x,r)$ *depends only on* x *and* $Read(r)$, *since the unread part of* r *cannot influence the computation time.*

DEFINITION 3.3. *The restrained time function of a randomizing algorithm* M *on* A *is the random function* $T(x,s) = T_M(x,s)$ *on* A *such that* $D_T = RF_M$ *and*

$$Time(x,r) = T(x, Read(r))$$

for every r *with a finite* $Read(r)$ *part.*

$Time_M$ *is not the only function on* $A \times \{0,1\}^\infty$ *that we have to consider. In this paper, a functional is a measurable function* $\mathcal{F}$ *from* $A \times \{0,1\}^\infty$ *to* $\bar{\mathcal{R}}^+$ *such that every finite value of* $\mathcal{F}$ *is a positive integer. We say that a functional* $\mathcal{F}$ *is continuous if it is continuous with respect to the product topology on* $A \times \{0,1\}^\infty$ *and the natural topology on* $\bar{\mathcal{R}}^+$. *The continuity implies that, for every* x *and every* $r \in \{0,1\}^\infty$ *with* $\mathcal{F}(x,r) < \infty$, *there exists a finite initial segment* s *of* r *such that* $\mathcal{F}(x,r) = \mathcal{F}(x,r')$ *for every* r' *with prefix* s. *The shortest initial segment* s *with this property will be denoted* $Read_{\mathcal{F},x}(r)$. *Define the restrained version of a functional* $\mathcal{F}$ *to be the random function* F *on* A *such that* $D_F = \{(x, Read_{\mathcal{F},x}(r)) : \mathcal{F}(x,r) < \infty\}$ *and* $F(x, Read(r)) = \mathcal{F}(x,r)$ *Thus,* $Time_M$ *is a continuous functional and* T_M *is the restrained version of* $Time_M$.

DEFINITION 3.4. *A functional* $\mathcal{F}$ *on* A *is almost total if, for every* $x \in A$ *of positive probability,*

$$\lambda\{r \in \{0,1\}^\infty : \mathcal{F}(x,r) < \infty\} = 1.$$

LEMMA 3.5. *A functional is almost total if and only if its restrained version is so.*

Proof. Clear. QED

DEFINITION 3.6. *An almost total functional* $\mathcal{F}$ *on a domain* A *is AP if there is a positive* ε *such that*

$$\underset{x}{E}\,\underset{r}{E}\left(\frac{1}{|x|}\mathcal{F}(x,r)^\varepsilon\right) < \infty,$$

where E_x and E_r mean expectation with respect to x and r, the relevant probability distributions being $\boldsymbol{P}_A$ and λ.

The proof of Lemma ?? works for almost total functionals.

COROLLARY 3.7. *The collection of AP almost total functionals over a given domain is closed under addition and multiplication.*

By definition of ρ, the expectation of a continuous almost-total functional with respect to λ and the expectation of its restrained version with respect to ρ agree for every x.

LEMMA 3.8. *An almost total, continuous functional $\mathcal{F}$ is AP if and only if its restrained version F is AP as a deterministic function on D_F.*

Proof.

$$\underset{x}{E}\,\underset{r}{E}\left(\frac{1}{|x|}\mathcal{F}(x,r)^{\varepsilon}\right) = \underset{x}{E}\,\underset{s}{E}\left(\frac{1}{|x|}F(x,s)^{\varepsilon}\right) = \underset{(x,s)\in D_F}{E}\frac{1}{|x|}F(x,s)^{\varepsilon}$$

where the expectation E_s is with respect to ρ. QED

Not all almost total functionals of interest to us are continuous. This is why we have defined directly when an almost total functional is AP.

DEFINITION 3.9. *A randomizing algorithm M is almost total if the functional $Time_M$ is so.*

By Lemma ??, M is almost total if and only if the restrained time function T_M is almost total.

DEFINITION 3.10. *An almost total randomizing algorithm M is AP time if the continuous functional $Time_M$ is AP.*

The intuitive content of this definition of almost total AP time algorithm is that, for each input x with $\boldsymbol{P}_A(x) > 0$, the algorithm will almost surely (i.e., with probability 1 with respect to r) terminate and will usually (with respect to x and r) require only a reasonable amount of time.

Now we turn attention to the main notion of this section, the notion of AP time algorithms. We consider randomizing algorithms which terminate with not too small probability (rather than almost surely) and which in fact require reasonable time with not too small probability. To avoid confusion, we remark that we do not require all the terminating computations to take reasonable time, only enough of them to have reasonable probability.

DEFINITION 3.11. *Let F be a random function on a domain A. F is AP on A if there is a subdilation Γ of D_F such that*

1. *The rarity function U_Γ is AP, and*
2. *For some $\varepsilon > 0$,*

$$\sum_{(x,s)\in\Gamma}\frac{1}{|x|}\boldsymbol{P}_A(x)2^{-|s|}F(x,s)^{\varepsilon} < \infty.$$

The condition (1) formalizes the part of the informal requirement above about enough terminating computations having reasonable probability. The meaning of (2) is clarified below (Lemma ??). Notice that the statement "F is AP on A" is in general weaker than the statement "F is AP as a deterministic function on D_F". We shall not need to generalize the definition of AP random function to functionals.

If Γ satisfies these conditions, we call it a set of good inputs for F. The dilation of A formed by Γ may be called the domain of good inputs. The fact that F is AP means intuitively that reasonably often F will be defined by virtue of a good input and that these values of F are reasonably small on average. (F may also be defined on some inputs that are not good, and the resulting values of F need not be small at all.) It is not required that good inputs are easily recognizable in any way.

LEMMA 3.12. *Let F be a random function on a domain A and let Γ be a subdilation of D_F such that the rarity function U_Γ is AP. Then the following conditions are equivalent:*

1. *F is AP on A with Γ as a set of good inputs.*
2. *The restriction of F (viewed as a deterministic function on D_F) to Γ is AP, i.e., there exists $\varepsilon > 0$ such that*

$$\mathop{E}_{(x,s)\in\Gamma} \frac{1}{|x|} F(x,s)^\varepsilon = \sum_{(x,s)\in\Gamma} \frac{1}{|x|} \boldsymbol{P}_A(x) 2^{-|s|} U_\Gamma(x) F(x,s)^\varepsilon < \infty.$$

Proof. Since $U_\Gamma(x) \geq 1$, (2) implies (1). Given (1), multiply and divide each summand in part 2 of the definition of AP by $U(x) = U_\Gamma(x)$ to produce $\boldsymbol{P}_\Gamma$ as a factor; we obtain

$$\mathop{E}_{x,s} \frac{1}{|x|} \frac{F(x,s)^\varepsilon}{U(x)} < \infty$$

where the expectation over (x,s) is with respect to $\boldsymbol{P}_\Gamma$. Also, as U is AP on A, we have, for some $\delta > 0$,

$$\mathop{E}_{x,s} \frac{1}{|x|} U(x)^\delta = \mathop{E}_{x} \frac{1}{|x|} U(x)^\delta < \infty,$$

where the expectation over x is with respect to $\boldsymbol{P}_A$ and we used that $\boldsymbol{P}_A$ is the projection of $\boldsymbol{P}_\Gamma$. Adding the last two inequalities and using the fact that any weighted geometric mean of two positive numbers is less then their sum, we find

$$\mathop{E}_{x,s} \left[\left(\frac{1}{|x|} \frac{F(x,s)^\varepsilon}{U(x)} \right)^{\frac{\delta}{1+\delta}} \cdot \left(\frac{1}{|x|} U(x)^\delta \right)^{\frac{1}{1+\delta}} \right] < \infty.$$

Algebraically simplifying the expression in brackets, we get

$$\mathop{E}_{x,s} \left[\frac{1}{|x|} F(x,s)^{\frac{\varepsilon\delta}{1+\delta}} \right] < \infty,$$

which is the desired inequality with $\frac{\varepsilon\delta}{1+\delta}$ *in place of* ε. *QED*

COROLLARY 3.13. *An almost total continuous functional* $\mathcal{F}$ *is AP if and only if its restrained version* F *is AP with* D_F *as a set of good inputs.*

Proof. Use Lemmas **??** *and* **??**. *QED*

DEFINITION 3.14. *A randomizing algorithm* M *on* A *is AP time if the restrained time function* T_M *is AP. A set of good inputs for* T *is also called a set of good inputs for* M.

In the case of an almost total randomizing algorithm M *we have now two definitions of polynomiality on average. It is easy to see that they are equivalent.*

COROLLARY 3.15. *For every almost total randomizing algorithm* M, *the following are equivalent:*

- *The functional* Time_M *is AP.*
- *The random function* T_M *is AP on the input domain.*

Proof. Use Lemma **??**. *QED*

DEFINITION 3.16. *A randomizing algorithm* M *is an AP time solution (resp. almost total, AP time solution) to search problem* $SP(A,W)$ *if it is AP time (resp. almost total and AP time) and every terminating computation of* M *with input* $x \in A$ *(and arbitrary random string* r*) produces as output a witness* w *with* $(x,w) \in W$.

It might seem more natural to require appropriate output only for "enough", not all, of the terminating computations, but our definition is somewhat simpler and essentially equivalent.

LEMMA 3.17. *Every AP time algorithm* M *for a search problem* $\Pi = SP(A,W)$ *that produces correct outputs on its good arguments can be converted to an AP time solution* M' *for* Π.

Proof. Append to M *instructions saying that, when* M *terminates,* M' *should check that the output* w *satisfies* $W(x,w)$ *and* M' *should terminate if and only if the check succeeds. As our definition of "search problem" required* W *to be polynomial time computable, the time used by* M' *is bounded by a polynomial of the sum of* $|x|$ *and the time used by* M, *so replacing* M *with* M' *does not ruin complexity estimates of the sort we are considering. QED*

LEMMA 3.18. *Suppose that a search problem* Π *has an AP time solution. Then the length of the shortest witness is a (deterministic) AP function.*

Proof. Let M *be an AP time solution to* Π *with a set* Γ *of good inputs. Recall that the rarity function* U *of* Γ *is the reciprocal of* $\rho_{M,x}(\Gamma(x)) = \sum_{s \in \Gamma(x)} 2^{-|s|}$. *Write* $l(x)$ *for the length of the shortest witness for* x. *Clearly,* $T(x,s) \geq l(x)$ *for all* $(x,s) \in \Gamma$, *as it takes time* $l(x)$ *just to write a witness on the output tape. For every* $\varepsilon > 0$, *we have*

$$\sum_{x \in A} \frac{1}{|x|} l(x)^{\varepsilon} \boldsymbol{P}_A(x) \quad = \quad \sum_{x \in A} \frac{1}{|x|} l(x)^{\varepsilon} \boldsymbol{P}_A(x) U(x) \rho(\Gamma(x))$$

$$\leq \sum_{(x,s)\in\Gamma} \frac{1}{|x|} T(x,s)^{\varepsilon} \boldsymbol{P}_A(x) U(x) 2^{-|s|}.$$

Now use the fact that T is AP and apply Lemma **??**. *QED*

4. Iteration. *It is well-known that a randomizing algorithm that solves a search problem with a certain probability can be iterated, using independent sequences of coin flips, to obtain a much higher success probability. A polynomial number of iterations suffices to improve success probabilities from as low as a reciprocal of a polynomial of $|x|$ to as high as exponentially (relative to $|x|$) close to 1. For average-case complexity, the situation is even better. We can start with a success probability whose reciprocal is not polynomially bounded but only polynomial on average, and we can iterate the algorithm to obtain a success probability 1, without increasing the time beyond AP. The main result in this section will establish a precise version of this claim and will show that it is optimal.*

Let M be a randomizing algorithm on a domain A which may or may not terminate on a given input. In our application, M is an algorithm for a search problem $SP(A,W)$ whose successful computations produce desired witnesses. For simplicity, we assume in this section that every termination of M is successful. It is easy to remove this assumption. Also, see Lemma **??**.

For technical reasons, we start with what we call a perpetual iteration M^∞ of a given randomizing algorithm M. Given an input (x,r), where $x \in A$ and r is an infinite random sequence, M^∞ simulates many computations of M on inputs (x,s) for the same x but different random bit strings s (disjoint substrings of r); these computations of M will be called subcomputations of M^∞. M^∞ is called a perpetual iteration because it never halts. Even after some subcomputations have terminated, M^∞ continues to run other subcomputations and to start new subcomputations, and it continues to allocate its random bits to subcomputations. The corresponding iteration M^* of M is like M^∞ except M^* halts if and when one of the subcomputations terminates. The advantage of working with M^∞ rather than M^* is that, in a computation of M^∞, every subcomputation runs either forever or until it terminates according to the rules of M, whereas in a computation of M^*, a subcomputation can stop "unnaturally" because another subcomputation terminated and thus stopped the whole computation of M^*.

Presupposing a fixed perpetual iteration M^∞ and a fixed element $x \in A$, for each sequence $r \in \{0,1\}^\infty$, let $\hat{s}_j = \hat{s}_j(r)$ be the subsequence of r that M^∞ with input (x,r) passes to subcomputation number j and let $s_j = s_j(r)$ be the subsequence of $\hat{s}_j$ that is actually read by subcomputation number j. Here are some examples of perpetual iteration.

EXAMPLE 1. Eager Perpetual Iteration. *Stage n of the computation of M^∞ consists of one step in the first, second, ..., n-th subcomputations, in that order. So one new subcomputation is started at each stage, and each previously started subcomputation is carried one step further. Also, during stage n, M^∞ reads n new bits from r and distributes them, in order, to these n subcomputations. Thus, each projection $\hat{s}_j$ is infinite. Note that this projection operation on $\{0,1\}^\infty$ does not depend on M or on its*

input x.

EXAMPLE 2. Lazy Perpetual Iteration *is similar to the eager perpetual iteration except that it distributes random bits only as the subcomputations ask for them. There is no longer an a priori defined infinite sequence $\hat{s}_j$ for each subcomputation; in fact, $\hat{s}_j = s_j$. To decide which subcomputation will receive a particular bit from r, it is no longer sufficient to know the position of that bit in r; one needs to know both the input x and the preceding bits of r.*

EXAMPLE 3. *Another variation of the eager perpetual iteration is obtained as follows. Fix a polynomially bounded injection $P(j,t)$ from pairs of positive integers to positive integers and reserve random bits $P(j,1), P(j,2), P(j,3), \ldots$ for the j-th subcomputation. This perpetual iteration is executed more naturally by a random access machine than by a Turing machine.*

DEFINITION 4.1. *A perpetual iteration M^∞ is polynomially fair if the number $m(j,t)$ of steps of M^∞ required to achieve the t-th step or termination (whichever occurs first) of subcomputation j is bounded by a polynomial of $j+t$.*

It follows that new subcomputations are started fairly often: the number $m(j,1)$ of steps of M^∞ required to start the j-th computation is bounded by a polynomial of j. Notice also that each subcomputation gets enough random bits to proceed until termination or forever.

REMARK 2. *To deal with parallel iterations (in a sense, they are more natural), add to the definition of polynomial fairness the requirement that the number of subcomputations started during the first n steps of M^∞ is bounded by a polynomial of n.*

We call a perpetual iteration M^∞ careful if each subcomputation gets a sequence of fair coin flips, and the sequences for different subcomputations are independent (provided that M^∞ gets a sequence of fair coin flips). We formulate this definition more formally. Recall the probability distribution λ of Section 3.

DEFINITION 4.2. *A perpetual iteration M^∞ is careful if, for each input x of positive probability, the function*

$$r \mapsto (s_1, s_2, \ldots) \text{ from } \{0,1\}^\infty \text{ to } (\{0,1\}^* \cup \{0,1\}^\infty)^\infty$$

sends λ to the product measure on $(\{0,1\}^* \cup \{0,1\}^\infty)^\infty$ induced by the measure $\rho_{M,x}$ on $\{0,1\}^* \cup \{0,1\}^\infty$.

LEMMA 4.3. *M^∞ is careful if and only if, for any input $x \in A$ and any finite list $t_1, \ldots, t_k$ of binary strings, the probability*

$$\lambda\{\, r : \ t_j \text{ is a prefix of } s_j(r) \text{ for all } j = 1, \ldots, k \,\}$$

is 0 if some t_j has a proper initial segment in $RF_M(x)$, and otherwise is 2^{-n} where n is the sum of the lengths of the t_j's.

Proof. Use Lemma **??**. *QED*

THEOREM 4.4. *Let M^∞ be a careful and polynomially fair perpetual iteration of M. Then M is AP time if and only if the iteration M^* is AP time and almost total.*

Proof. Suppose that M is AP time, i.e., the restrained time function $T = T_M$ is AP. Let Γ and ε be as in the definition of AP random function for T, and let $U = U_\Gamma$ be the rarity function of Γ. We first check that, for any input x of positive probability in A, M^* terminates with probability 1 (with respect to r). In fact, we show more, namely that, for almost all r, some s_j is good, i.e., belongs to $\Gamma(x)$. (If $s_j \in \Gamma(x)$ and $\boldsymbol{P}_A(x) > 0$, then the computation of M on (x, s_j) terminates, so we are indeed proving more than originally claimed.) The event (= set of r's such) that no s_j is in $\Gamma(x)$ is the intersection of the infinitely many events "$s_j \notin \Gamma(x)$" ($j = 1, 2, 3, \ldots$), which are independent and have probability $1 - \rho(\Gamma(x)) = 1 - (1/U(x))$. So their intersection has probability 0, as required.

*We are now ready to estimate the computation time $T^*(x, r)$ of M^* on input (x, r) and to prove that M^* is AP time. For any $x \in A$ of positive probability and any $r \in \{0,1\}^\infty$, let $k = k(x, r)$ be the smallest positive integer j with $s_j \in \Gamma(x)$. We saw in the preceding paragraph that, for all $x \in A$ of positive probability, such a k exists for almost all r. (Define $k(x, r) = \infty$ on the measure zero set of pairs (x, r) where no such k exists.) Let $T' = T'(x, r)$ be the time taken by the computation of M with input $(x, s_{k(x,r)})$, which is the k-th subcomputation of M^∞ on input (x, r). Since M^∞ is polynomially fair, the computation time T^* of M^* is bounded by a polynomial of $(k+T')$. By Corollary **??**, it suffices to prove that $k(x, r)$ and $T'(x, r)$ are AP.*

First, we treat k. [1] *We begin with a well-known and quite general observation. Suppose an experiment succeeds with probability $p > 0$, and suppose we make many independent repetitions of this experiment until one, say the k-th, succeeds. Then k has expectation*

$$\sum_{i=1}^{\infty} i(1-p)^{i-1}p = 1/p.$$

In the situation at hand, the experiments are the subcomputations of $M^\infty(x, r)$ (with random r), which are independent by carefulness, "success" means that the string read by the subcomputation is good, and thus $1/p = U(x)$, $k = k(x, r)$ and $\underset{r}{E}\, k(x, r) = U(x)$.

*Choose positive $\delta < 1$ witnessing that U is AP: $\underset{x}{E}(\frac{1}{|x|}U(x)^\delta) < \infty$. We check that this δ also witnesses that k is AP. The function $f(y) = y^\delta$ is a concave function on the open real interval $I = (0, \infty)$. By Jensen's inequality (see Corollary **??**),*

$$\underset{x}{E}\,\underset{r}{E}\left(\frac{1}{|x|}k(x,r)^\delta\right) = \underset{x}{E}\left(\frac{1}{|x|}\,\underset{r}{E}\left[k(x,r)^\delta\right]\right) \leq$$

$$\leq \underset{x}{E}\left(\frac{1}{|x|}\left[\underset{r}{E}(k(x,r))\right]^\delta\right) = \underset{x}{E}\left(\frac{1}{|x|}U(x)^\delta\right) < \infty.$$

[1] Curiously, the useful functional $k(x, r)$ is not continuous.

Next, we treat $T'(x,r)$. Clearly, $T'(x,r) = T(x,s_k)$, where $k = k(x,r)$. Recall that a positive ε, one of the witnesses that M is AP, was chosen. We begin by computing the expectation of $T'(x,r)^\varepsilon = T(x,s_k)^\varepsilon$ with respect to r.

$$\underset{r}{E}(T(x,s_k)^\varepsilon) = \sum_{j=1}^{\infty} \boldsymbol{P}(k(x,r)=j) \cdot \underset{r}{E}\left(T(x,s_j)^\varepsilon \mid k(x,r)=j\right).$$

The event $k(x,r) = j$ whose probability occurs here is the intersection of the event $s_j \in \Gamma(x)$ and all events $s_i \notin \Gamma(x)$ with $i < j$. Since M^∞ is careful, these are independent, so the product $(1-\frac{1}{U(x)})^{j-1} \cdot \frac{1}{U(x)}$ of their probabilities equals $\boldsymbol{P}(k(x,r)=j)$. The conditional expectation of $T(x,s_j)^\varepsilon$ relative to this intersection of events equals the conditional expectation of $T(x,s_j)$ relative to $s_j \in \Gamma(x)$, since, by the carefulness of M^∞, $T(x,s_j)^\varepsilon$ is independent of the events $s_i \notin \Gamma(x)$ with $i \neq j$. Further,

$$\underset{r}{E}\left(T(x,s_j)^\varepsilon \mid s_j \in \Gamma(x)\right) = \underset{s\in\Gamma(x)}{E} T(x,s)^\varepsilon = \sum_{s\in\Gamma(x)} T(x,s)^\varepsilon \cdot 2^{-|s|} \cdot U(x).$$

Thus, we have

$$\begin{aligned}
\underset{r}{E}(T'(x,r)^\varepsilon) &= \sum_{j=1}^{\infty}(1-\frac{1}{U(x)})^{j-1} \cdot \frac{1}{U(x)} \cdot \sum_{s\in\Gamma(x)} \left(T(x,s)^\varepsilon \cdot 2^{-|s|} \cdot U(x)\right) = \\
&= \left(\sum_{j=1}^{\infty}(1-\frac{1}{U(x)})^{j-1}\right) \cdot \sum_{s\in\Gamma(x)} \left(T(x,s)^\varepsilon \cdot 2^{-|s|}\right) = \\
&= U(x) \cdot \sum_{s\in\Gamma(x)} \left(T(x,s)^\varepsilon \cdot 2^{-|s|}\right).
\end{aligned}$$

Recall that Γ, U and ε were chosen to witness that M is AP. Thus, U is AP and the factor multiplying it is AP as well. (In fact, this factor is average linear.) So the product is AP. Choose a positive $\delta < 1$ to witness this. Using Jensen's inequality again, we have

$$\begin{aligned}
\underset{x}{E}\underset{r}{E}\left(\frac{1}{|x|}T'(x,r)^{\varepsilon\delta}\right) &= \underset{x}{E}\left(\frac{1}{|x|}\underset{r}{E}\left[T'(x,r)^{\varepsilon\delta}\right]\right) \\
&\leq \underset{x}{E}\left(\frac{1}{|x|}\left[\underset{r}{E}(T'(x,r)^\varepsilon)\right]^\delta\right) < \infty.
\end{aligned}$$

Thus, we have the desired estimate for T', which completes the proof of one direction of the theorem.

To prove the converse, assume M^* is an almost total AP time algorithm. We must produce a Γ and a positive ε witnessing that M is AP. Fix a positive $\varepsilon < 1$ witnessing that T^* is AP: $\underset{x}{E}\underset{r}{E}\left(\frac{1}{|x|}T^*(x,r)^\varepsilon\right) < \infty$. That is,

$$\underset{x}{E}\left(\frac{1}{|x|}F(x)\right) < \infty, \text{ where } F(x) = \underset{r}{E}\left(T^*(x,r)^\varepsilon\right).$$

Observe that, quite generally, if X is a nonnegative random variable with expectation m, then, by Markov's inequality, $\boldsymbol{P}(X \leq 2m) \geq 1/2$. (Indeed, if $\boldsymbol{P}(X \leq 2m) = p$, then

$$\begin{aligned} m = \boldsymbol{E}(X) &= p \cdot \boldsymbol{E}(X \mid X \leq 2m) + (1-p)\, \boldsymbol{E}(X \mid X > 2m) \\ &\geq p \cdot 0 + (1-p) \cdot 2m, \end{aligned}$$

so $1 - p \leq 1/2$, as claimed.)

Applying this observation to the random variable $T^*(x,r)^\varepsilon$, for a fixed x and random $r \in \{0,1\}^\infty$, we find that the set

$$\Gamma'(x) = \{r \in \{0,1\}^\infty : T^*(x,r)^\varepsilon \leq 2F(x)\}$$

has probability $\geq 1/2$. For each $r \in \Gamma'(x)$, let $s(x,r)$ be the string of random bits actually read by the subcomputation of M^* on (x,r) that produced a witness for x. We define

$$\Gamma = \{(x, s(x,r)) : x \in A \text{ and } r \in \Gamma'(x)\},$$

so that $\Gamma(x) = \{s(x,r) : r \in \Gamma(x)\}$, and we claim that this Γ and the ε fixed above are as required by the definition of M being AP. By the definition of $s(x,r)$, Γ forms a dilation of A.

To check that the rarity function $U = U_\Gamma$ is AP, recall that for $r \in \Gamma(x)$, $T^*(x,r) \leq (2F(x))^{1/\varepsilon}$. Therefore the number $q = q(x)$ of subcomputations started by M^* on input (x,r) is bounded by a polynomial of $F(x)$. Since $F(x)$ is AP (in fact, linear on average), $q(x)$ is AP. We have:

$$\begin{aligned} \frac{1}{2} &= \sum_{s \in \Gamma(x)} \sum_{j=1}^{q} \boldsymbol{P}\{r \in \Gamma'(x) : s(x,r) = s \text{ and} \\ &\qquad\qquad \text{subcomputation } j \text{ is the one that succeeds}\} \leq \\ &\leq \sum_{s \in \Gamma(x)} \sum_{j=1}^{q} \boldsymbol{P}\{r \in \Gamma'(x) : \text{subcomputation } j \text{ reads exactly } s\} = \\ &= \sum_{s \in \Gamma(x)} \sum_{j=1}^{q} 2^{-|s|} = q(x) \cdot \frac{1}{U(x)} \end{aligned}$$

So $U(x)$ is bounded by the AP function $2q(x)$.

To prove the convergence condition in the definition of T being AP, recall that each s in $\Gamma(x)$ is $s(x,r)$ for some $r \in \Gamma'(x)$. Hence $T(x,s)^\varepsilon \leq T^*(x,r)^\varepsilon \leq 2F(x)$. So

$$\sum_{s \in \Gamma(x)} T(x,s)^\varepsilon 2^{-|s|} \leq 2F(x) \cdot \sum_{s \in \Gamma(x)} 2^{-|s|} \leq 2F(x),$$

which implies the desired convergence, because F is linear on average. This completes the proof of Theorem **??**. *QED*

It is easy to see that the perpetual iterations of Examples 1–3 are polynomially fair and that the perpetual iterations of Example 1 and Example 3 are careful as well. It

is not quite obvious that the lazy perpetual iteration is careful. In the case of the lazy perpetual iteration, some information about a given subcomputation j can be learned from watching other subcomputations. Imagine, for example, that the first m random bits received by subcomputation $j > 1$ happened to have the same values as the first m random bits received by subcomputation 1 and subcomputation 1 requested another random bit. We know then that subcomputation j will request another random bit as well.

THEOREM 4.5. *Suppose that a perpetual iteration M^∞ has the following properties:*

- *M^∞ starts infinitely many subcomputations on every input (x, r).*
- *No subcomputation of M^∞ is ever stalled: either it terminates or else it executes infinitely many steps.*
- *M^∞ serves random bits on the "first come, first served" basis.*

Then M^∞ is careful.

*Proof. Fix an arbitrary $x \in A$ of positive probability. Recall that $s_j = s_j(r)$ is the string of random bits passed to the j-th subcomputation. By Lemma **??**, we must show that, for any input $x \in A$ and any list $t_1, \ldots, t_k$ of binary strings, the probability*

$$\lambda\{r : \ t_j \text{ is a prefix of } s_j(r) \text{ for all } j = 1, \ldots, k\}$$

is 0 if some t_j has a proper initial segment in $RF_M(x)$, and otherwise is 2^{-n} where n is the sum of the lengths of the t_j's.

The first half of this is clear, for if t_j has a proper initial segment in $RF_M(x)$, then no subcomputation can read all of t_j, so $s_j \neq t_j$. To prove the second half, consider a list $t_1, \ldots, t_k$ of cumulative length n and with no proper initial segments of any t_j being in $RF_M(x)$. Note that $s_j(r)$ cannot, for any r, be a proper initial segment of t_j because any finite s_j is in $RF_M(x)$ whereas no t_j has a proper initial segment in $RF_M(x)$.

Call a finite binary string q, which we regard as a possible initial segment of $r \in \{0,1\}^\infty$, pertinent (to the fixed list $t_1, \ldots, t_k$) if, when M^∞ reads q from its random tape, the strings $s_j(q)$ that it passes to subcomputations $j = 1, \ldots, k$ are consistent with the t_j's. ("Consistent" means one is an initial segment, not necessarily proper, of the other.) For pertinent q, write $s'_j(q)$ for $s_j(q)$ or t_j, whichever is an initial segment of the other; this is the part of t_j already read by subcomputation j when M^∞ has read q. Write $m(q)$ for the sum of the lengths of the strings $s'_j(q)$. We have $0 \leq m(q) \leq n$.

Let E denote the event

$$\{\, r : \ t_j \text{ is a prefix of } s_j(r) \text{ for all } j = 1, \ldots, k \,\}.$$

Recall that our objective is to prove that $\lambda(E) = 2^{-n}$. Call $q \in \{0,1\}^*$ strange if it is pertinent and $\lambda[E \mid q \text{ is a prefix of } r] \neq 2^{-n+m(q)}$. Observe that, since the empty string e is pertinent and prefixes every r, e is strange if and only if $\lambda(E) \neq 2^{-n}$, which is the negation of what we want to prove. We therefore assume e is strange and attempt to deduce a contradiction.

LEMMA 4.6. *If q is strange, then so is at least one of its one-bit extensions $q0$ or $q1$.*

Proof. Consider a computation of M^∞ (on our fixed input x) that has read q (and passed its bits to the appropriate subcomputations).

Case 1. The next request for a bit comes from subcomputation i with $i > k$ or with $i \leq k$ and $s'_i(q) = t_i$.

Then both $q0$ and $q1$ are pertinent, and $s'_j(q0) = s'_j(q1) = s'_j(q)$ for all $j = 1, \ldots, k$, so $m(q0) = m(q1) = m(q)$. As q is strange,

$$2^{-n+m(q)} \neq \lambda[E \mid q \text{ is a prefix}] = \frac{1}{2}\left(\lambda[E \mid q0 \text{ is a prefix}] + \lambda[E \mid q1 \text{ is a prefix}]\right),$$

so the conditional probabilities on the right cannot both equal $2^{-n+m(q)}$. This means that one of $q0$ and $q1$ is strange.

Case 2. The next request for a random bit comes from subcomputation i such that $i \leq k$ and $s'_i(q)$ is a proper initial segment of t_i.

Then the next random bit read by M^∞ will go to subcomputation i, and so $q0$ (resp. $q1$) will be pertinent if and only if the next bit in t_i after s'_i is 0 (resp. 1). Suppose, without loss of generality, that this bit is 0, so $q0$ is pertinent. We have $s'_i(q0) = s'_i(q)0$ and $s'_j(q0) = s'_j(q)$ for j in $\{1, \ldots, k\} - \{i\}$. We also have, as q is strange and $\lambda[E \mid q1 \text{ is a prefix}] = 0$ (as $q1$ is impertinent),

$$2^{-n+m(q)} \neq \lambda[E \mid q \text{ is a prefix}] = \frac{1}{2} \cdot \lambda[E \mid q0 \text{ is a prefix}],$$

so

$$\lambda[E \mid q0 \text{ is a prefix}] \neq 2^{-n+m(q)+1} = 2^{-n+m(q0)},$$

and $q0$ is strange. QED

Now we obtain a contradiction from the assumption that e is strange, as follows. Repeatedly apply the lemma, starting with $q = e$, to obtain a sequence of strange strings, each a one-bit extension of the previous one. Thus, we have an $r \in \{0,1\}^\infty$ each of whose finite initial segments is strange. Fix such an r. By fairness, each subcomputation j, for $j = 1, \ldots, k$, either terminates or is infinite on r.

For $j = 1, \ldots, k$, each $s_j(r)$ is consistent with t_j, because all finite initial segments of r, being strange, are pertinent. We saw earlier that $s_j(r)$ cannot be a proper initial segment of t_j, so t_j must be an initial segment of $s_j(r)$, for each $j = 1, \ldots, k$. Thus, for a sufficiently long finite initial segment q of r, each t_j is an initial segment of the corresponding $s_j(q)$, and therefore $s'_j(q) = t_j$ and $m(q) = n$. But then the event "q is a prefix" is a part of E, so

$$\lambda[E \mid q \text{ is a prefix}] = 1 = 2^{-n+m(q)}.$$

This contradicts the fact that q, a finite initial segment of r, is strange. QED

5. Randomizing Many-One Reductions. *Let us clarify the notion of composition $M = M_2 \circ M_1$ of randomizing algorithms. Suppose that M_1 computes a random function f from a domain A to a domain B, and M_2 is a randomizing algorithm on B. Given an instance $x \in A$ and a "random" sequence $r \in \{0,1\}^\infty$, M begins by simulating M_1 on x and r. If this computation terminates with output x', and if s is the finite initial segment of r read during this computation, then M continues by simulating M_2 on x' and the part of r after s. Thus,*

$$R_M(x) = \{st : s \in RF_{M_1}(x) \text{ and } t \in R_{M_2}(f(x,s))\} \cup RI_{M_1}(x).$$

LEMMA 5.1. *Suppose that $\boldsymbol{P}_B(f(x,s)) > 0$ whenever $\boldsymbol{P}_A(x) > 0$, and let $T = T_{M_2}$ be the restrained time function of M_2. Then*

- *M is almost total if f and T are almost total, and*
- *M is AP time if f is AP time and the composition $T \circ f$ is AP.*

Proof. The first claim is clear. To prove the second claim, recall that, by definition, a randomizing algorithm is AP if and only if its restrained time function is so. Let Γ be a set of good inputs for $T \circ f$. It is easy to see that Γ is also a set of good inputs for f. Except for a small bookkeeping overhead, T_M is the sum of T_{M_1} and $T \circ f$ on Γ. Thus, Γ is a set of good inputs for T_M, so that T_M is AP. QED.

In the rest of this section, f is a random function from a domain A to a domain B and T ranges over random functions on B. If Γ is a dilation of A and Δ is a dilation of B, define $\Gamma * \Delta$ to be the dilation of A comprising pairs (x, st) such that $(x,s) \in \Gamma$ and $(f(x,s),t) \in \Delta$. The notion of domination is recalled in Section 2.

THEOREM 5.2. *Suppose that f is almost total. Then the following are equivalent:*

1. *For every almost total, AP random T, the composition $S = T \circ f$ is almost total and AP.*
2. *The almost total function $(x,s) \mapsto |f(x,s)|_B$ is AP and $D_f \leq_f B$.*

*Proof. First, suppose (1). Every deterministic function on B is an almost total, random function on B with good inputs (y,e) where $\boldsymbol{P}_B(y) > 0$ and e is the empty string. By Theorem **??**, $D_f \leq_f B$ and the restriction of the function $(x,s) \mapsto |f(x,s)|_B$ to D_f is AP. This implies (2).*

*Second, suppose (2) and let T be an almost total, AP random function on B. The domain $\Gamma = D_f$ of the AP function $(x,s) \mapsto |f(x,s)|_B$, is an almost total dilation of A. Similarly, the domain Δ of T is an almost total dilation of B. These two facts and the domination condition imply that the dilation $D_S = \Gamma * \Delta$ is almost total. It remains to check that S, as a deterministic function $S(x,st) = T(f(x,s),t)$ on $\Gamma * \Delta$, is AP. This function is the composite of T with the function $j(x,st) = (f(x,s),t)$ from $\Gamma * \Delta$ to Δ. By Theorem **??**, it suffices to prove that Δ dominates $\Gamma * \Delta$ with respect to j. We have:*

$$\begin{aligned} \boldsymbol{P}_{\Gamma*\Delta}[j^{-1}\{j(x,st)\}] &= \boldsymbol{P}_{\Gamma*\Delta}\{(x_0,s_0t) : f(x_0,s_0) = f(x,s)\} \\ &= \boldsymbol{P}_\Gamma[f^{-1}\{f(x,s)\}] \cdot 2^{-|t|} \end{aligned}$$

and

$$\boldsymbol{P}_\Delta\{j(x,st)\} = \boldsymbol{P}_B\{f(x,s)\} \cdot 2^{-|t|},$$

so the ratio in the definition of domination for j is $\dfrac{\boldsymbol{P}_\Gamma[f^{-1}\{f(x,s)\}]}{\boldsymbol{P}_B\{f(x,s)\}}$, *which is AP on* Γ *because* $\Gamma \leq_f B$. *It follows immediately that this ratio is AP as a function on* $\Gamma * \Delta$ *(a function not depending on the* t *part of its argument). QED*

COROLLARY 5.3. *Let f be a function from a domain A to a domain B, computed by an almost total, AP time randomizing algorithm M_1. Suppose that $D_f \leq_f B$. Then, for every almost total, AP time randomizing algorithm M_2 on B, the composite algorithm $M = M_2 \circ M_1$ is almost total and AP time.*

Now we turn to the case when the composite is not required to be almost total.

THEOREM 5.4. *Suppose that Γ is a dilation of A. The following four statements are equivalent.*

(a) *For every AP random function T on B with Δ as a set of good inputs, $T \circ f$ is AP with $\Gamma * \Delta$ as a set of good inputs.*

(b) *For every almost total AP random function T on B with Δ as a set of good inputs, $T \circ f$ is AP with $\Gamma * \Delta$ as a set of good inputs.*

(c) *For every deterministic AP function T on B, $T \circ f$ is AP with Γ as a set of good inputs.*

(d) *The function $x \mapsto |f(x)|_B$ is AP with Γ as a set of good inputs and $\Gamma \leq_f B$.*

Proof. Clearly, (a) implies (b).

To derive (c) from (b), suppose (b) and let T be as in (c). Consider the dilation Δ of B comprising pairs (y, e) where y ranges over B and e is the empty string. Now apply (b).

To derive (d) from (c), use Lemma **??**, *to restate (c) as follows: If T is any AP (deterministic) function on B, then $T \circ f$ is an AP deterministic function on Γ. By Theorem* **??** *(and another application of Lemma* **??***), this implies (d).*

It remains to derive (a) from (d). Assume (d), and let T and Δ be as in (a). Our first task is to show that the rarity function $U_{\Gamma*\Delta}$ for the composite dilation $\Gamma * \Delta$ of A is AP. It follows from (d) that U_Γ is AP. By the hypothesis of (a), U_Δ is AP. We compute

$$U_{\Gamma*\Delta}(x) = \left(\sum_{s\in\Gamma(x)} \sum_{t\in\Delta(f(x,s))} 2^{-|s|}2^{-|t|}\right)^{-1} = \left(\sum_{s\in\Gamma(x)} 2^{-|s|} \cdot \sum_{t\in\Delta(f(x,s))} 2^{-|t|}\right)^{-1} =$$

$$= \left(\sum_{s\in\Gamma(x)} 2^{-|s|} \cdot \frac{1}{U_\Delta(f(x,s))}\right)^{-1}$$

and therefore, for any δ with $0 < \delta < 1$,

$$\sum_{x\in A} \frac{1}{|x|}\boldsymbol{P}_A(x)U^\delta_{\Gamma*\Delta}(x) = \sum_{x\in A} \frac{1}{|x|}\boldsymbol{P}_A(x)\left(\sum_{s\in\Gamma(x)} 2^{-|s|} \cdot \frac{1}{U_\Delta(f(x,s))}\right)^{-\delta} =$$

$$= \sum_{x\in A} \frac{1}{|x|}\boldsymbol{P}_A(x)\left(\sum_{s\in\Gamma(x)} 2^{-|s|} \cdot U_\Gamma(x)\frac{1}{U_\Gamma(x)U_\Delta(f(x,s))}\right)^{-\delta} =$$

$$= \sum_{x\in A} \frac{1}{|x|} \boldsymbol{P}_A(x) \left(\underset{s\in\Gamma(x)}{E} \left[\frac{1}{U_\Gamma(x) U_\Delta(f(x,s))} \right] \right)^{-\delta},$$

where the expectation over $s \in \Gamma(x)$ is with respect to the probability distribution $\boldsymbol{P}(s) = 2^{-|s|} \cdot U_\Gamma(x)$.

Apply Jensen's inequality (in the form of Corollary ??) to the last part of the computation above:

$$\sum_{x\in A} \frac{1}{|x|} \boldsymbol{P}_A(x) U^\delta_{\Gamma*\Delta}(x) \le \sum_{x\in A} \frac{1}{|x|} \boldsymbol{P}_A(x) \underset{s\in\Gamma(x)}{E} \left[(U_\Gamma(x) \cdot U_\Delta(f(x,s))^\delta \right] =$$

$$= \sum_{x\in A} \sum_{s\in\Gamma(x)} \frac{1}{|x|} \boldsymbol{P}_A(x) \cdot 2^{-|s|} \cdot U_\Gamma(x) \left[(U_\Gamma(x) \cdot U_\Delta(f(x,s)))^\delta \right] =$$

$$= \sum_{(x,s)\in\Gamma} \frac{1}{|(x,s)|_\Gamma} \boldsymbol{P}_\Gamma(x,s) \left[(U_\Gamma(x) \cdot U_\Delta(f(x,s))^\delta \right].$$

Thus, $U_{\Gamma*\Delta}$ is AP if the product of $U_\Gamma(x)$ and $U_\Delta(f(x,s))$ is AP on Γ, i.e., if both factors are AP on Γ. Since $U_\Gamma(x)$ is AP on A, it is AP on Γ. Applying Theorem ??, we find that the composition $U_\Delta(f(x,s))$ is AP. Hence $U_{\Gamma*\Delta}$ is AP.

It remains to verify the convergence condition of the definition "$T \circ f$ is AP with $\Gamma * \Delta$ as a set of good points", which is, by Lemma ??, equivalent to the statement that $T \circ f$, as a deterministic function on $\Gamma * \Delta$, is AP. By the hypothesis of (a) plus Lemma ??, T, as a deterministic function on Δ, is AP. Since $T \circ f$ is obtained from T by composition with the function $g(x,st) = (f(x,s),t)$ from $\Gamma * \Delta$ to Δ and $|g|$ is AP on on $\Gamma * \Delta$, the desired conclusion will follow, by Theorem ??, if we show that Δ dominates $\Gamma * \Delta$ with respect to g.

For this purpose, we use Lemma ?? and seek a measure ν on $\Gamma * \Delta$ that projects by g to $\boldsymbol{P}_\Delta$ and that dominates $\boldsymbol{P}_{\Gamma*\Delta}$. The assumption (d) that B dominates Γ with respect to f provides a measure μ on Γ that projects to $\boldsymbol{P}_B$ by f and dominates $\boldsymbol{P}_\Gamma$. We use μ to define ν by

$$\nu(x,st) = \mu(x,s) \cdot 2^{-|t|} \cdot U_\Delta(f(x,s)).$$

This projects to $\boldsymbol{P}_\Delta$ via g, because, for any $(x',t') \in \Delta$,

$$\sum_{(x,st)\in g^{-1}(x',t')} \nu(x,st) = \sum_{(x,s)\in f^{-1}(x')} \nu(x,st') =$$

$$\sum_{(x,s)\in f^{-1}(x')} \mu(x,s) \cdot 2^{-|t'|} \cdot U_\Delta(f(x,s)) = 2^{-|t'|} U_\Delta(x') \sum_{(x,s)\in f^{-1}(x')} \mu(x,s) =$$

$$2^{-|t'|} U_\Delta(x') \boldsymbol{P}_B(x') = \boldsymbol{P}_\Delta(x',t').$$

So it remains to prove that ν dominates $\boldsymbol{P}_{\Gamma*\Delta}$, that is, that the ratio $\dfrac{\boldsymbol{P}_{\Gamma*\Delta}(x,st)}{\nu(x,st)}$ is an AP function on $\Gamma * \Delta$. But this ratio is

$$\frac{\boldsymbol{P}_A(x) \cdot 2^{-|s|} \cdot 2^{-|t|} \cdot U_{\Gamma*\Delta}(x)}{\mu(x,s) \cdot 2^{-|t|} \cdot U_\Delta(f(x,s))} = \frac{\boldsymbol{P}_\Gamma(x,s) \cdot U_{\Gamma*\Delta}(x)}{U_\Gamma(x) \mu(x,s) U_\Delta(f(x,s))} \le \frac{\boldsymbol{P}_\Gamma(x,s)}{\mu(x,s)} \cdot U_{\Gamma*\Delta}(x).$$

To show that this is AP on $\Gamma * \Delta$, it suffices to show that both $\dfrac{\boldsymbol{P}_\Gamma(x,s)}{\mu(x,s)}$ and $U_{\Gamma*\Delta}(x)$, considered as functions of (x,st) are AP on $\Gamma * \Delta$. Of course, the former is AP as a function of (x,s) on Γ, by our choice of μ, and the latter is AP as a function of x on A, as proved above, but neither of these is exactly what is needed. The desired information about $U_{\Gamma*\Delta}(x)$ follows easily because $\boldsymbol{P}_{\Gamma*\Delta}$ projects to $\boldsymbol{P}_A$ under the map $(x,st) \mapsto x$, which preserves sizes. Indeed, we have

$$\sum_{(x,st)\in\Gamma*\Delta} \frac{1}{|(x,st)|_{\Gamma*\Delta}} \cdot \boldsymbol{P}_{\Gamma*\Delta}(x,st) \cdot (U_{\Gamma*\Delta}(x))^\varepsilon =$$

$$\sum_{x\in A} \left(\frac{1}{|x|} \cdot (U_{\Gamma*\Delta}(x))^\varepsilon \cdot \sum_{(x,st)\in\Gamma*\Delta} \boldsymbol{P}_{\Gamma*\Delta}(x,st) \right) =$$

$$\sum_{x\in A} \frac{1}{|x|} \cdot (U_{\Gamma*\Delta}(x))^\varepsilon \boldsymbol{P}_A(x) < \infty$$

for sufficiently small $\varepsilon > 0$, since $U_{\Gamma*\Delta}$ is AP on A.

The same approach does not succeed with the function $\dfrac{\boldsymbol{P}_\Gamma}{\mu}$ on Γ, because the projection $(x,st) \mapsto (x,s)$ from $\Gamma * \Delta$ to Γ need not send $\boldsymbol{P}_{\Gamma*\Delta}$ to $\boldsymbol{P}_\Gamma$; factors of $U_{\Gamma*\Delta}$ and U_Γ get in the way. Fortunately, Lemma ?? allows us to ignore those factors. Since $\dfrac{\boldsymbol{P}_\Gamma}{\mu}$ is AP on Γ, the lemma guarantees an $\varepsilon > 0$ such that

$$\sum_{(x,s)\in\Gamma} \frac{1}{|x|} \boldsymbol{P}_A(x) \cdot 2^{-|s|} \left(\frac{\boldsymbol{P}_\Gamma(x,s)}{\mu(x,s)} \right)^\varepsilon < \infty.$$

For any fixed $(x,s) \in \Gamma$, the numbers $2^{-|t|}$, where t ranges over $\Delta(f(x,s))$, add up to at most 1, since no such t is a proper initial segment of another. So

$$\sum_{(x,st)\in\Gamma*\Delta} \frac{1}{|x|} \boldsymbol{P}_A(x) \cdot 2^{-|s|} 2^{-|t|} \left(\frac{\boldsymbol{P}_\Gamma(x,s)}{\mu(x,s)} \right)^\varepsilon < \infty.$$

By Lemma ??, $\dfrac{\boldsymbol{P}_\Gamma}{\mu}$ is AP on $\Gamma * \Delta$, and the proof of Theorem ?? is complete. QED

QUESTION. *Do parts (a), (b) and (c) of Theorem 5.3 remain equivalent if they are weakened to assert only that $T \circ f$ is AP, without specifying a particular set of good inputs? Is there an analog of (d) for this situation?*

COROLLARY 5.5. *Let f be a random function from a domain A to a domain B computed by an AP time randomizing algorithm M_1. If there exists a domain of good inputs for f dominated, with respect to f, by B then, for every AP randomizing algorithm M_2 on B, the composite algorithm $M = M_2 \circ M_1$ is AP time.*

6. Impagliazzo-Levin's theorem. *In order to formulate the theorem in question, we need a couple of definitions. We start with the definition of uniform domains.*

In the case of domains with finite many elements, it would be natural to call a domain uniform if all elements have the same probability. This definition makes no sense in the case of infinite domains, which is the only case of interest to us. Another natural way to define uniform domains requires a default probability distribution on positive integers; it is customary to assign the probability $\frac{1}{n(n+1)}$ *to a positive integer* n*.*

DEFINITION 6.1. *A domain is* uniform *if it has a finite number of elements of any given size, all elements of a given size have the same probability, and* $\boldsymbol{P}\{x : |x| = n\} = \frac{1}{n(n+1)}$.

DEFINITION 6.2 ([?]). *A domain* A *is* samplable *if there exists a randomized algorithm* S *such that*

- S *takes no input (but tosses coins) and outputs an element of* A *if it converges,*
- $\boldsymbol{P}_A(x)$ *is proportional to the probability that* S *outputs* x*, and*
- *the computation time of* S *is bounded by a polynomial of the size of the output.*

The restriction on the computation time of S *can be relaxed [?].*

Impagliazzo and Levin [?] deal with domains (though they do not use the term) where the size $|x|$ *of an element* x *is its length (recall that domain elements are strings); this restriction is not necessary [?] but it simplifies the exposition and we stick to it in this section. Call a sampling algorithm* S length preserving *if it uses exactly* n *coin tosses to produce a string of length* n*. The following fact is well known and we omit the proof.*

LEMMA 6.3 ([?, ?, ?]). *Every search problem on a samplable domain reduces to a search problem on a domain sampled by a length-preserving algorithm.*

Implagliazzo and Levin prove that every NP search problem on a samplable domain reduces to an NP search problem on a uniform domain. Recall that a search problem $SP(A, W)$ *is NP if the length of a witness is bounded by a polynomial of the size of the instance and the witness relation* $W(x, w)$ *is PTime computable relative to* $|x|$*. In fact, the Impagliazzo-Levin argument does not require the restriction to NP problems.*

THEOREM 6.4 ([?]). *Every search problem* $\Pi_1 = SP(A_1, W_1)$ *on a samplable domain* A_1 *reduces to a search problem on the uniform domain BS of binary strings. If the source problem is NP then the target problem may be chosen to be NP as well.*

Proof. First, we indicate the motivation behind the proof. There are two simple but unsuccessful attempts at reducing $SP(A_1, W_1)$ *to a search problem on a uniform domain. One is to simply redefine the probability distribution on* A_1 *to be uniform. This fails because the obvious instance-transformer, the identity map, violates the domination condition. It is entirely possible for some instances* x *to have vastly larger probability in the given samplable* A_1 *than in the uniform domain; this occurs when* $S^{-1}(x)$ *is large (as* S *preserves length). A second attempt is to use the search problem on the domain BS defined by* $W'(u) = W(S(u))$*. The trouble with this is that an instance-transformer should produce, for any* x*, some* $u \in S^{-1}(x)$*, and this may be difficult, even for a randomizing algorithm. Indeed, one could imagine that each such* u *encodes a witness* w *for* x*, so that the new search problem is trivial. Notice, however, that this second*

attempt fails only when $S^{-1}(x)$ is rather small; when it is large, an element of it can be found by guessing random elements and checking them (as S is quickly computable).

The strategy of the proof is to interpolate between these two attempts, leaning toward the first (resp. second) when $S^{-1}(x)$ is small (resp. large). More precisely, an instance of the new problem should be (approximately) $n = |x|$ bits long, consisting of $l = \lceil \log(||S^{-1}(x)||) \rceil$ bits of information about some $u \in S^{-1}(x)$ and $n - l$ bits about x. (Here $||S^{-1}(x)||$ is the cardinality of the set $S^{-1}(x)$.)

There are some obvious difficulties with this. For one thing, we can't efficiently compute l from x. But we can guess it; it lies between 0 and n, so the probability of guessing correctly is the reciprocal of a polynomial, which is good enough to give a nonrare dilation. Another issue is how to select the right bits of information about u and x. This, too, is solved by randomization. We randomly choose two matrices L and M (with entries in the two-element field $\{0,1\}$) of the appropriate size to hash u and x to vectors Lu and Mx of lengths l and $n - l$, respectively. A key point in the proof is that randomly chosen hash matrices have a reasonable probability of working the way we want.

Thus, finally, an instance of the new problem will consist of the two hash matrices L and M and the results Lu and Mx of hashing u and v. A witness for such an instance will consist of a u and a w such that Lu is as specified in the instance, $S(u)$ is an x whose hashing Mx is as specified in the instance, and $W(x, w)$ holds. (Actually, there are a couple of minor technical modifications in the actual proof, but this is the essential idea.)

Now we give the actual proof. We construct search problems $\Pi_i = SP(A_i, W_i)$ for $i = 2, 3, 4, 5$ with $A_5 = BS$ and reduce each Π_i with $i < 5$ to Π_{i+1}. Accordingly, we have 4 reduction lemmas. In each lemma, the desired reduction is called (Γ, f, g). The probability distribution of A_i is denoted $\boldsymbol{P}_i$.

By Lemma **??**, *we may suppose that S is size-preserving.*

NOTATION. *In the rest of this section, x is an instance of Π_1 with $W_1(x) \neq \emptyset$, $n = |x|$, m is the cardinality of the set $S^{-1}(x)$, and $l = \lceil \log m \rceil$. If j is a natural number then $b(j)$ is the (shortest) binary notation for j.*

We define Π_2. A_2 comprises pairs (x, l) where x is an element of A_1 and l is as above. The size and probability of (x, l) are the size and probability of x in A_1. Further, $W_2(x, l) = W_1(x)$.

LEMMA 6.5. *Π_1 reduces to Π_2.*

Proof. Define $\Gamma(x)$ to contain one string, namely, $b(l)$. We have: $Rarity_\Gamma(x) = 2^{|b(l)|} \leq 2l \leq 2n$, so that Γ is not rare.

Define $f(x, b(l)) = (x, l)$. To check the domination property, take into account that f is injective:

$$\frac{\boldsymbol{P}_\Gamma(x, b(l))}{\boldsymbol{P}_2(x, l)} = \frac{\boldsymbol{P}_1(x) \cdot 2^{-|b(l)|}}{\boldsymbol{P}_1(x)} \leq 1.$$

Finally, define $g((x, b(l)), w) = w$. QED

NOTATION: *u ranges over binary strings of length n, u' ranges over binary strings of length l, L ranges over $l \times n$ matrices over the field of two elements. If I is a matrix over the field of two elements then $b(I)$ is the binary string obtained from I by writing down the first row of I, then the second row of I, and so on. If s and s' are strings then $s * s'$ denotes the concatenation of s and s'.*

We define Π_3. A_3 comprises triples (x, L, u') such that L has full rank (i.e. rank l) and there exists $u \in S^{-1}(x)$ with $Lu = u'$. A_3 is isomorphic to a subdomain of $A_2 \times BS$; the isomorphism is $\iota_3(x, L, u') = ((x, l), b(L) * u'))$. Further,

$$W_3(x, L, u') = \{(u, w) : S(u) = x, Lu = u' \text{ and } w \in W_1(x)\}.$$

LEMMA 6.6. *Π_2 reduces to Π_3.*

Proof. Define Γ to be the ι_3-image of A_3. To prove that Γ is not rare, fix an instance (x, l) of Π_2.

CLAIM 6.7. *In the uniform probability space of $l \times n$ matrices, the probability that a matrix has full rank exceeds a positive constant independent of l and n, e.g., $1/4$.*

Proof. Only one row (namely, the row of zeroes) cannot serve as the first row of a full rank matrix. Given the first row u_1, only two rows (namely, u_1 and the zero row) cannot serve as the second row of a full rank matrix. Given the first two rows u_1 and u_2, only four rows (namely, the four linear combinations of u_1 and u_2) cannot serve as the third row of a full rank matrix. And so on. Thus the number of full rank matrices is

$$(2^n - 1)(2^n - 2)(2^n - 4)\ldots(2^n - 2^{l-1}) = \prod_{i=0}^{l-1}(2^n - 2^i).$$

The total number of $l \times n$ matrices is 2^{ln}. Hence the probability of full rank is

$$\frac{\prod_{i=0}^{l-1}(2^n - 2^i)}{2^{ln}} = \prod_{i=0}^{l-1}\frac{2^n - 2^i}{2^n} = \prod_{i=0}^{l-1}(1 - \frac{1}{2^{n-i}}) >$$
$$\prod_{i=0}^{\infty}(1 - \frac{1}{2^i}) = (1 - \frac{1}{2})(1 - \frac{1}{4})(1 - \frac{1}{8})\ldots.$$

To estimate this product, consider the following probabilistic experiment. For each i, let U_i be an urn with 2^i balls such that exactly one of the balls is red and the others are green. Let A_i be the event of selecting a green ball from U_i, and B_i be the event of selecting the red ball, so that $\boldsymbol{P}(A_i) = 1 - \boldsymbol{P}(B_i) = 1 - \frac{1}{2^i}$. We have

$$\prod_{i=2}^{\infty}(1 - \frac{1}{2^i}) = \boldsymbol{P}\left[\bigcap_{i=2}^{\infty} A_i\right] = 1 - \boldsymbol{P}\left[\bigcup_{i=2}^{\infty} B_i\right] >$$
$$1 - \sum_{i=2}^{\infty}\boldsymbol{P}(B_i) = 1 - (\frac{1}{4} + \frac{1}{8} + \ldots) = 1 - \frac{1}{2} = 1/2.$$

Hence the probability of full rank exceeds $(1 - \frac{1}{2})\frac{1}{2} = 1/4$. QED

CLAIM 6.8. *Consider the probability space of pairs (L, u') where L is of full rank. For each x, the probability of the event*

$$\{(L, u') : (\exists u \in S^{-1}(x))(Lu = u')\}$$

is at least $3/8$.

Proof. *Let u, v range over $S^{-1}(x)$. For each u, let $E(u) = \{(L, u') : Lu = u'\}$. Then the cardinality $||E(u)||$ of $E(u)$ is the number F of full rank matrices L. The event in question is $\bigcup E(u)$, and the probability in question is $||\bigcup E(u)||/(F2^l)$.*

We check that if $u \neq v$ then the set $E(u, v) = E(u) \cap E(v)$ contains at most $F/2^l$ elements. Notice that

$$[(L, u') \in E(u, v)] \Longleftrightarrow [Lu = u' = Lv] \Longrightarrow [L(u - v) = 0]$$

and, for each L that annihilates $u-v$, there exists a unique u' such that $(L, u') \in E(u, v)$. Thus $||E(u, v)||$ equals the number of full rank matrices L which annihilate $u-v$. Among all $l \times n$ matrices, the probability of annihilating $u - v$ is exactly $(1/2)^l$; among full-rank matrices, the probability is $\leq (1/2)^l$. Let us make this precise.

If $l = n$ then the probability that a full-rank matrix annihilates $u - v$ equals 0; so assume that $l < n$. The probability p that a random full-rank matrix L annihilates a specified nonzero vector $z \in \{0, 1\}^n$ is independent of z. For any z, z' there is a non-singular $n \times n$ matrix A such that $Az = z'$. Then L annihilates z' if and only if LA annihilates z, and LA has full rank if and only if L does.

So, without loss of generality, $z = (0, 0, \ldots, 0, 1)$. It is clear now that p equals the number of full-rank $l \times n$ matrices with the last column of zeroes divided by the total number of full-rank $l \times n$ matrices. In other words, p equals the number of full-rank $l \times (n-1)$ matrices divided by the number of full-rank $l \times n$ matrices. Recall our counting of full-rank matrices above. We have:

$$p = \frac{\prod_{i=0}^{l-1}(2^{n-1} - 2^i)}{\prod_{i=0}^{l-1}(2^n - 2^i)} \leq \frac{\prod_{i=0}^{l-1}(2^{n-1} - 2^{i-1})}{\prod_{i=0}^{l-1}(2^n - 2^i)} = \prod_{i=0}^{l-1} \frac{1}{2} = (\frac{1}{2})^l.$$

*By the inclusion-exclusion principle [**?**],*

$$\frac{1}{F2^l}||\bigcup_u E(u)|| \geq \frac{1}{F2^l}\left(\sum_u ||E(u)|| - \sum_{u \neq v} ||E(u) \cap E(v)||\right).$$

Since the number of u's is m, $\frac{1}{F2^l}\sum_u ||E(u)|| = \frac{1}{F2^l} mF = \frac{m}{2^l}$. Further,

$$\frac{1}{F2^l}\sum_{u \neq v} ||E(u) \cap E(v)|| \leq \frac{1}{F2^l}\frac{m(m-1)}{2} F/2^l = \frac{m(m-1)}{2^{2l+1}}.$$

Thus, the probability we want is bounded below by $\frac{m}{2^l} - \frac{m(m-1)}{2^{2l+1}} > t - \frac{1}{2}t^2$ where $t = \frac{m}{2^l}$ is between one-half and one (because of the definition of l). But the minimum of

the quadratic function $t - \frac{1}{2}t^2$ *on the interval* $[\frac{1}{2}, 1]$ *is* $\frac{3}{8}$. *So the probability we want is greater than* $\frac{3}{8}$. *QED*

The two claims imply that Γ *is non-rare. Define*
$f = \iota_3^{-1}$ *and* $g(((x,l), b(L) * u'), (u,w)) = w$.
It is easy to see (Γ, f, g) *is indeed the desired reduction. QED*

NOTATION: $k = n + 1 - l$ *and* M *ranges over* $k \times n$ *matrices over the 2-element field.*

We define Π_4. A_4 *comprises 5-tuples* (x, L, u', M) *such that* $(x, L, u') \in A_3$, M *is as above (a* $k \times n$ *matrix) and there is no* $u \in \{0,1\}^n - S^{-1}(x)$ *such that* $Lu = u'$ *and* $M(S(u) - x) = 0$. *It is isomorphic to a subdomain of* $A_3 \times BS$ *via the isomorphism* $\iota_4(x, L, u', M) = ((x, L, u'), b(M))$. *Further,* $W_4(x, L, u', M) = W_3(x, L, u')$.

LEMMA 6.9. Π_3 *reduces to* Π_4.

Proof. Define Γ *to be the* ι_4*-image of* A_4. *To prove that* Γ *is not rare, fix an element* (x, L, u') *of* A_3 *and call a matrix* M bad *if there exists* $u \in \{0,1\}^n - S^{-1}(x)$ *such that* $Lu = u'$ *and* $M(S(u) - x) = 0$.

CLAIM 6.10. *The fraction of bad matrices* M *is at most 1/2.*

Proof. The number of strings u *satisfying the equation* $Lu = u'$ *is* $2^{n-l} = 2^{k-1}$. *For each* $u \in \{0,1\}^n - S^{-1}(x)$, *there are exactly* 2^{kn-k} *matrices* M *satisfying the equation* $M(S(u) - x) = 0$. *Thus the number of bad matrices* M *is at most* $2^{k-1}2^{kn-k} = 2^{kn-1}$, *which is exactly one half of the total number of matrices* M. *QED*

The claim implies that Γ *is not rare. Define* $f = \iota_4^{-1}$ *and*
$g(((x, L, u'), b(M))(u, w)) = (u, w)$. *It is easy to see that* Γ, f, g *is indeed a reduction. QED*

The domain A_5 *of our final problem* Π_5 *is BS. We do some work before defining* W_5. *Consider the function*

$$f(x, L, u', M) = b(L) * u' * b(M) * b(Mx) * b(l).$$

from A_4 *to* A_5. *(It will serve eventually as the instance transformer of the desired reduction of* Π_4 *to* Π_5.*)*

CLAIM 6.11. f *is injective, and the components* L, u' *and* M *as well as the numbers* n, l *and* k *are PTime computable from* $f(x, L, u', M)$

Proof. As $|f(x, L, u', M)| = ln + l + kn + k + l = (n+1)^2 + l$ *and* $l \leq n$, *the numbers* n, l *and* k *are easily computable from* $|f(x, L, u', M)|$. *It follows that* L, u' *and* M *are easily computable from* $f(x, L, u', M)$. *To prove that* f *is injective, suppose that* $f(x_2, L, u', M) = f(x, L, u', M)$. *Then* $M(x_2 - x) = 0$ *and there exists* $u \in S^{-1}(x_2)$ *such that* $Lu = u'$ *and* $M(S(u) - x) = 0$. *Since* M *isn't bad for* (x, L, u') *(in the sense of the proof of Lemma* **??***),* $x_2 = S(u) = x$. *QED*

CLAIM 6.12. f *deterministically reduces* A_4 *to* A_5.

Proof. Clearly, f is PTime computable. We need only to check that f satisfies the domination condition. Since f is injective,

$$\begin{aligned}\frac{\boldsymbol{P}_4(f^{-1}(f(x,L,u',M))}{\boldsymbol{P}_5(f(x,L,u',M))} &= \frac{\boldsymbol{P}_4(x,L,u',M)}{\boldsymbol{P}_5(f(x,L,u',M))} \\ &= \frac{\boldsymbol{P}_1(x)\boldsymbol{P}_{BS}(b(L)*u'*b(M))}{\boldsymbol{P}_{BS}(b(L)*u'*b(M)*b(Mx)*b(l))}.\end{aligned}$$

We may ignore factors polynomial in $|(x,L,u')|$, i.e., polynomial in n. In that sense,

$$\frac{\boldsymbol{P}_1(x)\boldsymbol{P}_{BS}(b(L)*u'*b(M))}{\boldsymbol{P}_{BS}(b(L)*u'*b(M)*b(Mx)*b(l))} \approx \frac{2^{-(n-l)}2^{-(ln+l+kn)}}{2^{-(ln+l+kn+k+\log l)}} \approx 1. \quad QED$$

Now we are ready to define W_5.

$$W_5(s) = \begin{cases} W_4(x,L,u',M) & \text{if } s = f(x,L,u',M) \\ \emptyset & \text{otherwise.}\end{cases}$$

Since S is PTime computable and Π_1 is an NP search problem, the search problem Π_5 is NP as well.

LEMMA 6.13. *Π_4 deterministically reduces to Π_5.*

Proof. The instance transformer f of the desired reduction is already defined. The witness transformer is $g((x,L,u',M),(u,w)) = (u,w)$. It is easy to see that (f,g) is indeed a reduction. QED

Theorem **??** *is proved. QED*

7. Appendix. Jensen's inequality. *For the reader's convenience, we prove here two forms of Jensen's inequality that we need. The proof of Jensen's inequality from [**?**] is used.*

THEOREM 7.1. *Consider an increasing concave function f on an interval (a,∞) of the real line and set $f(\infty) = \lim_{x\uparrow\infty} f(x)$. For every random variable X with values in the interval $(a,\infty]$ of the real line extended with ∞,*

$$\mathbb{E}(f(X)) \le f(\mathbb{E}(X)).$$

Proof. If $E(X) = \infty$ then $f(E(X)) = \lim_{x\uparrow\infty} f(x) \ge E(f(X))$. Suppose that $E(X) < \infty$.

The fact that f is concave means that for $a < u < v < w$,

$$\frac{f(v)-f(u)}{v-u} \ge \frac{f(w)-f(v)}{w-v}.$$

It is clear that the monotone limits

$$A(v) = \downarrow \lim_{u\uparrow v} \frac{f(v)-f(u)}{v-u}, B(v) = \uparrow \lim_{w\downarrow v} \frac{f(w)-f(v)}{w-v}$$

exist and $A(v) \geq B(v)$. For all positive real v, x and every c in $[B(v), A(v)]$, we have $f(x) \leq c(x-v)+f(v)$. In particular, there is c such that $f(X) \leq c(X-E(X))+f(E(x))$ and the desired inequality follows on taking expectations. QED

COROLLARY 7.2. *Suppose $0 < \delta < 1$ and let X be any random variable with values in $(0, \infty]$. Then*

$$E(X^\delta) \leq (E(X))^\delta.$$

THEOREM 7.3. *Consider a decreasing convex function g on an interval $(0, b)$ of the real line and set $g(0) = \lim_{x \downarrow 0} g(x)$. For every random variable X with values in the interval $[0, b)$ of the real line,*

$$E(g(X)) \geq g(E(X)).$$

Proof. First suppose $E(X) = 0$. Since all values of X are ≥ 0, we must have with probability 1 that $X = 0$ and therefore $g(X) = g(0)$. Then $E(g(X)) = g(0) = g(E(X))$.

In the case $E(X) > 0$, the proof is similar to the part of the proof of Theorem ?? for the case $E(X) < \infty$; just reverse some arrows and inequalities. QED

COROLLARY 7.4. *Suppose $0 < \delta < 1$ and let X be any random variable with values in $[0, b)$. Then*

$$E(X^{-\delta}) \geq (E(X))^{-\delta}.$$

REFERENCES

[1] *Shai Ben-David, Benny Chor, Oded Goldreich and Michael Luby, On the Theory of Average Case Complexity, Symposium on Theory of Computing, ACM, 1989, 204–216.*

[2] *Andreas Blass and Yuri Gurevich, On the Reduction Theory for Average-Case Complexity, CSL'90, 4th Workshop on Computer Science Logic (Eds. E. Börger, H. Kleine Büning and M. Richter), Springer LNCS, 1991.*

[3] *Andreas Blass and Yuri Gurevich, Randomizing Reductions of Search Problems, 11th Conference on Foundations of Software Technology and Theoretical Computer Science, New Delhi, India, Springer Lecture Notes in Computer Science 560 (1991), 10–24.*

[4] *Andreas Blass and Yuri Gurevich, Randomizing Reductions of Decision Problems (tentative title), in preparation.*

[5] *Yuri Gurevich, Average Case Complexity, J. Computer and System Sciences 42:3, June 1991 (a special issue on FOCS'87), 346–398.*

[6] *Yuri Gurevich, Average Case Complexity, International Colloquium on Automata, Languages and Programming, Springer Lecture Notes in Computer Science 510, 1991, 615–628.*

[7] *Yuri Gurevich and Saharon Shelah, Expected computation time for Hamiltonian Path Problem, SIAM J. on Computing 16:3 (1987) 486–502.*

[8] *Russel Impagliazzo and Leonid A. Levin, No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random, Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1990, 812–821.*

[9] *Leonid A. Levin, Average Case Complete Problems, SIAM Journal of Computing 15 (1986), 285–286.*

[10] *Richard P. Stanley, Enumerative Combinatorics I, Wadsworth & Brooks/Cole, 1986.*

[11] *Ramarathnam Venkatesan and Leonid Levin, Random Instances of a Graph Coloring Problem are Hard, Symposium on Theory of Computing, ACM, 1988, 217–222.*

[12] *David Williams, Probability with Martingales, Cambridge University Press, 1991.*