Electronic Colloquium on Computational Complexity, Report No. 138 (2010)

Exposition of the Muchnik-Positselsky Construction of a Prefix Free Entropy Function that is not Complete under Truth-Table Reductions

Eric Allender[*]
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08855, USA
allender@cs.rutgers.edu

Luke Friedman[*]
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08855, USA
lbfried@cs.rutgers.edu

William Gasarch
Dept. of Computer Science
University of Maryland
College Park, MD, 20742
gasarch@cs.umd.edu

August 25, 2010

1 Introduction

In this manuscript we give an exposition of a theorem by Muchnik and Positselsky (Theorem 2.7 of [MP02]). This exposition was developed in the course of writing an article [AFG10] that builds on the techniques of Muchnik and Positselsky. Some of the material here is lifted verbatim from [AFG10].

Definition 1.1 If f is a function then $ov(f)$, the overgraph of f, is

$$\{(x, y) \,:\, f(x) \leq y\}.$$

The following example gives a familiar overgraph, related to (plain) Kolmogorov complexity:

Example 1.2 Fix a Universal Turing machine U. Let $C(x)$ be the size of the shortest s such that $U(s) = x$. Note that

$$ov(C) = \{(x, y) \,:\, \text{ there is an } s,\ |s| \leq y \text{ such that } U(s) = x\}.$$

Note that $ov(C)$ is c.e.

This paper is concerned much more with *prefix complexity*.

Definition 1.3

[*]Supported in part by NSF Grants CCF-0830133 and CCF-0832787.

ISSN 1433-8092

1. A *prefix Turing machine* is a Turing machine M such that, for all x, if $M(x)$ halts then, for all $y \neq \lambda$, $M(xy)$ does not halt. That is, the domain of M is a prefix code.
2. Let M be a prefix Turing machine. Define $K_M(x)$ to be the size of the shortest s such that $M(s) = x$.
3. A *universal prefix Turing machine* is a prefix Turing machine U such that, for any prefix Turing machine M, there is a constant c such that for all $x, K_U(x) \leq K_M(x) + c$.

Example 1.4 Let M be a prefix Turing machine. Note that

$$ov(K_M) = \{(x, y) \ : \ \text{there is an } s, \, |s| \leq y \text{ such that } M(s) = x\}.$$

Note that $ov(K_M)$ is c.e.

Definition 1.5 We select some universal prefix Turing machine U and call $K_U(x)$ the *prefix complexity of* x. We will omit the subscript U. Note that the choice of U only affects $K(x)$ by an additive constant.

The following definition was used implicitly by Muchnik and Positselsky:

Definition 1.6 A *Prefix Free Entropy Function* f is a function from $\{0,1\}^*$ to $\mathbb{N}$ such that

- $\sum_{x \in \{0,1\}^*} 2^{-f(x)} \leq 1$ and
- $ov(f)$ is c.e.

The canonical example of a Prefix Free Entropy Function is $K(x)$. (K can be seen to be a prefix free entropy function by appeal to the Kraft Inequality; see e.g. [LV08, Theorem 1.11.1].)

Note that if f is a Prefix Free Entropy Function, then 2^{-f} is a special case of what Li and Vitányi call a *Lower Semicomputable Discrete Semimeasure* [LV08, Definition 4.2.2]. We recall the *Coding Theorem* (see [LV08, Theorem 4.3.3]), the proof of which yields the following important relationship between prefix free entropy functions and prefix complexity.

Theorem 1.7 *Let f be a prefix free entropy function. Given a machine computing f, one can construct a prefix machine M such that $f(x) = K_M(x) - 1$.*

Proof: The Coding Theorem, as stated and proved in [LV08, Theorem 4.3.3], gives only the *inequality* $f(x) \leq K_M(x) - 3$, where 2^{-f} is a lower semicomputable semimeasure. However, since we are dealing with the special case where f is a prefix free entropy function, we can dispense with some of the technicalities in the proof of [LV08, Theorem 4.3.3]. In particular (using the terminology utilized by Li and Vitányi in their proof) we use intervals I_x, and observe that each I_x contains a *binary* interval having length (exactly) half the length of I_x. Modifying the proof of [LV08, Lemma 4.3.3], to give the label of this interval (instead of the label of the *entire* interval in the case where I_x is itself a binary interval) yields the desired prefix machine. ▮

We will make use of the following easy propositions.

Proposition 1.8 *Let U and U' be prefix Turing machines. Then there is a prefix machine U'' such that $K_{U''}(x) = \min(K_U(x), K_{U'}(x)) + 1$.*

Proof: The domain of U'' is $\{1x : x \text{ is in the domain of } U\} \cup \{0x : x \text{ is in the domain of } U'\}$. ▮

Proposition 1.9 *Given any machine U and constant c, there is a machine U' such that $K_U(x) + c = K_{U'}(x)$*

Proof: The domain of U' is $\{0^c x : x \text{ is in the domain of } U\}$. ▮

Notation 1.10 Let γ be a Turing machine or any process that works in steps.

- $\gamma(y)\downarrow$ means that $\gamma(y)$ halts.
- $\gamma(y)\uparrow$ means that $\gamma(y)$ does not halt.
- $\gamma_s(y)$ is the result of running $\gamma(y)$ for s steps.
- $\gamma_s(y)\downarrow$ means that $\gamma(y)$ halts within s steps.
- $\gamma_s(y)\uparrow$ means that $\gamma(y)$ has not halted within s steps. Note that we can determine this.

We will consider γ as a machine that (possibly) computes a truth-table reduction. We will interpret the output of γ to be an encoding of a Boolean circuit, with inputs labeled by atoms of the form "$(x, r) \in ov(H)$".

2 Main Theorem

Theorem 2.1 *There exists a prefix free entropy function H and a c.e. set A such that $A \not\leq_{tt} ov(H)$. The function H is also $K_{U'}$ for some universal prefix Turing machine.*

Proof:
We will first give the intuition before presenting the formal construction and proof.

Let $\gamma_0, \gamma_1, \ldots$ be a list of all tt-reductions. Actually this is a list of Turing machines using the convention stated in Notation 1.10.

Recall that $K(x)$ is the normal prefix complexity of x. Using K we will construct a c.e. set A and a function F, to form a function $H : \{0,1\}^* \to \mathbb{N}$ with the following properties.

1. F is a total function and $ov(F)$ is c.e.
2. $H(x) = \min(K(x) + 4, F(x) + 2)$.
3. $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq \frac{1}{4}$.
4. $A \not\leq_{tt} ov(H)$. This is broken up into an infinite number of requirements:

   R_e : γ_e is not a tt-reduction of A to $ov(H)$.

Claim 1: Given the above properties, (1) $H = K_{U'}$ for some universal prefix machine U', and $A \not\leq_{tt} ov(H)$ (this is just property 4).

Proof: By Property 3 we have that $\sum_{x\in\{0,1\}^*} 2^{-F(x)+2} \leq \frac{1}{4}$. Therefore $\sum_{x\in\{0,1\}^*} 2^{-F(x)} \leq 1$, which combined with Property 1 means that F is a prefix free entropy function. By Proposition 1.7 we then have that $F+1$ is K_M for some prefix machine M. Since $K = K_U$ for some universal prefix machine U, by Proposition 1.9 we have that $K+3$ is $K_{U''}$ for some universal prefix machine U''. Thus, by Lemma 1.8, $H(x) = \min(K(x)+4, F(x)+2) = \min(K(x)+3, F(x)+1)+1$ is $K_{U'}$ for some universal prefix machine U'. ∎

Our control over H comes from our freedom in constructing the function F. The construction will occur in stages – at any given time in the construction there will be a "current" version of F which we will denote by F^*. Similarly, there will be a "current" version of K denoted by K^*, which represents our knowledge of K at a given stage. At all times, H^*, our "current" version of H, will be defined as $\min(K^*(x)+4, F^*(x)+2)$.

Originally we set $F^*(x) = 2|x|+2$ and K^* as the empty function. At each stage of the construction we will assume that a new element (x,y) is enumerated into $ov(K)$ according to some fixed enumeration of $ov(K)$. (This is possible since $ov(K)$ is c.e.) When this occurs K^* is updated by setting $K^*(x) = \min(K^*(x), y)$. (Since K^* is a partial function, it is possible that $K^*(x)$ was previously undefined. In this case we set $K^*(x) = y$.) Similarly, during the construction at times we will modify F by enumerating elements into $ov(F)$. Whenever we enumerate an element (x,y) into $ov(F)$, F^* is updated by setting $F^*(x) = \min(F^*(x), y)$.

2.1 How to Satisfy a Particular Requirement

We discuss how to satisfy a particular requirement. Let $e \in \mathbb{N}$. Assume we have a witness x picked out for the purpose of satisfying R_e. (During the construction a particular witness x will be used at most once in this way). If $\gamma_e(x)\uparrow$ then R_e is satisfied, though we may never know it. This will lead to taking no action.

Assume that at some stage s we see that $\gamma_{e,s}(x)\downarrow$. By Notation 1.10 $\gamma_e(x)$ is an encoding of a circuit $\lambda_{e,x}$. The output of the circuit $\lambda_{e,x}$ is determined by the truth values of the atoms "$z \in ov(H)$" that label the inputs to the circuit. Define $\lambda_{e,x}[H']$ to be the truth value obtained by taking the circuit $\lambda_{e,x}$ and for each atom "$(z,r) \in ov(H)$" using the truth value of "$(z,r) \in ov(H')$ in its place. In order to satisfy the requirement R_e we would like to put x in A iff $\lambda_{e,x}[H]$ says NO. The problem is that at a given stage s we can "guess" at the value of $\lambda_{e,x}[H]$ by computing $\lambda_{e,x}[H^*]$, but in general we cannot know the value of $\lambda_{e,x}[H]$ for sure, because as H^* evolves the value of $\lambda_{e,x}[H^*]$ may change. The main difficulty is that the function K is out of our control and determining whether $(z,r) \in ov(K)$ is in general an uncomputable task.

We do have *some* influence over the situation, though, due to our control of F. Indeed, for any atom "$(z,r) \in ov(H)$", we can ensure that the truth value of the atom is 1 by enumerating $(z, r-2)$ into $ov(F)$. (Note that for all x, the value of $H^*(x)$ can only decrease over time). We have to be careful about making these types of changes though; if we are too liberal in modifying F we may violate the condition $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq 1/4$ in the process. Thus the construction becomes a balancing act – at stage s we will make a decision as to whether to put x in A or not, and from then on try to use F to ensure that $\lambda_{e,x}[H^*] \neq A(x)$ while at the same time maintaining the invariant that $\sum_{x\in\{0,1\}^*} 2^{-H^*(x)} \leq 1/4$. (In particular, if F_s is the function F^* at the beginning of stage s, for all x we will not want $\lim_{s\to\infty} F_s(x)$ to be very much smaller than $K(x)$).

As part of our solution, for each R_e we will find a suitable witness x and set up a game $\mathcal{G}_{e,x}$ played between us (making moves by enumerating elements into $ov(F)$), and K, who makes moves by enumerating elements into $ov(K)$. (Even though elements are obliviously enumerated into $ov(K)$ according to some fixed enumeration we will treat K as if it is a willful adversary). We will always make our choice about whether x is placed in A or not in order to assure that we have a winning strategy: as long as K continues to make legal moves we can respond with changes to F (our own legal moves) that both assure that R_e is satisfied and that $\sum_{x\in\{0,1\}^*} 2^{-H^*(x)} \leq 1/4$.

It is possible that K will cheat by enumerating elements into $ov(K)$ in such a way that it plays an illegal move. In this case we will simply destroy the game $\mathcal{G}_{e,x}$ and start all over again with a new game $\mathcal{G}_{e,x'}$, using a different witness x'. However we will be able to show that if K cheats infinitely often on games associated with a particular requirement R_e, then $\sum_{x\in\{0,1\}^*} 2^{-K(x)}$ diverges; therefore it can only happen finitely often.

The requirements $R_1, R_2, R_3, \dots$ are listed in priority ordering. If during stage s a move is played on a game $\mathcal{G}_{e,x}$, we say that R_e is "acting". In this case for all $e < e' \leq s$, if $\mathcal{G}_{e',y}$ is the game associated with $R_{e'}$ currently being played, we destroy this game and start a new game $\mathcal{G}_{e',y'}$ with some new witness y'. When this happens we say that each of the $R_{e'}$ has been "injured" by R_e. The reason this works in the end is that at some point $R_1, R_2, \dots, R_{e-1}$ have stopped acting, so R_e will no longer ever be injured by some higher priority requirement.

2.2 Description of the Game

Before we present the other details of the formal construction, let us describe one of the games $\mathcal{G}_{e,x}$ in more depth and provide some analysis of the game. Let the inputs to the Boolean circuit $\lambda_{e,x}$ (encoded by $\gamma_e(x)$) be labeled by the atoms $\{(z_1, r_1), \dots, (z_k, r_k)\}$. Let $X_e = \{z_1, \dots, z_k\}$. Note that the queries in this reduction are of the form: "Is $H(z_i) \leq r_i$?". If $H^*(z_i) \leq r_i$ then we already know $H(z_i) \leq r_i$, so we can replace that input to the circuit with the value $TRUE$ and simplify the circuit accordingly. Renumber the z's, rename k to again be the number of questions, and rename X_e to be the set of all z's being asked about. When we are done we have $\{(z_1, r_1), \dots, (z_k, r_k)\}$ and we know that $(\forall z_i \in X_e)[H^*(z_i) > r_i]$.

We make one more change to X_e. If there exists an element z_i such that $z_i \in X_e$ and $z_i \in X_{e'}$ for some $e' < e$, then changing H^* on the value z_i during the game $\mathcal{G}_{e,x}$ could affect the game associated with the requirement $R_{e'}$, which would upset our priority ordering. Hence we will take

$$X_e = X_e - \bigcup_{e'<e} X_{e'}.$$

This will ensure that R_e cannot injure any $R_{e'}$ with $e' < e$.

Let $H^*_{e,x}$ be the function H^* when the game $\mathcal{G}_{e,x}$ is first constructed. Let $\epsilon = 2^{-e-i_e-5}$. (How i_e is determined will be explained later). The game $\mathcal{G}_{e,x}$ is played on a labeled DAG. The label of each node of the DAG has the following two parts:

1. A function h that maps X_e to $\mathbb{N}$. The function h provides conjectured values for H restricted to X_e. The function h will be consistent with $H^*_{e,x}$ in that $(\forall i)[h(z_i) \leq H^*_{e,x}(z_i)]$.

2. A truth value VAL, which is the value of $\lambda_{e,x}$ assuming that $(\forall z \in X_e)[H(z) = h(z)]$. Note that this will be either YES or NO indicating that either, under assumption $(\forall z \in X_e)[H(z) = h(z)]$, $\lambda_{e,x}$ thinks $x \in A$ or thinks $x \notin A$.

There is a separate node in the DAG for every possible such function h.

We now describe the start node and how to determine the edges of the DAG.

1. There is a node (h, VAL) where $h = H^*_{e,x}$ restricted to X_e. This is the start node and has indegree 0.

2. There is an edge from (h, VAL) to (h', VAL') if for all $z_i \in X_e$, $h(z_i) \geq h'(z_i)$ (so it is possible that H^* could at some point evolve from $H^*_{e,x}$ to h, and then at a later point evolve from h to h'.)

The game $\mathcal{G}_{e,x}$ is played between two players who we call FIRST and SECOND which indicates who goes first. Each player has a score, which originally is zero, and represents how much the player has been penalized so far in the game. (In other words a high score is bad).

1. The game starts with a token placed on the start node.
2. FIRST picks a value $V \in \{YES, NO\}$.
3. FIRST goes first after which the players alternate moves.
4. On a given turn a player can either leave the token where it is or move the token to a new node in the DAG. Suppose a player moves the token from a node t to a node t', where h is the function labeling t and h' is the function labeling t'. In this case we add $\sum_{z_i \in X_e} 2^{-h'(z_i)} - 2^{-h(z_i)}$ to the player's score. A player can legally move the token from node t to t' if
   (a) There is an edge from t to t' in the game DAG.
   (b) The score of the player after making the move does not exceed ϵ.
5. FIRST wins if the token ends up on a node with $\mathit{VAL} = V$. SECOND wins otherwise.

By an easy strategy stealing argument FIRST has a winning strategy.

During the actual construction the games will be played between us (the construction) trying to make the computation go one way, and K (which we do not control) trying to make it go (perhaps) another way. We will be FIRST and hence have a winning strategy. We will effect our moves by enumerating elements into $ov(F)$, which changes F^* and hence H^*. (To move the token to a node labeled with the function h, we modify H^* so that h equals H^* restricted to the set X_e) The K moves will occur when a new element is enumerated into $ov(K)$ at the beginning of each stage, which changes K^* and hence H^*. (In this case K is moving the token to the node in the game DAG labeled by the new H^*).

The key is that the players' scores measure how much the sum $\sum_{x \in \{0,1\}^*} 2^{-H^*(x)}$ has gone up, which we bound by not allowing a player's score to exceed ϵ. (Of course K is oblivious to the rules of the game and will at times cheat – we take this into account as part of our analysis.) One final note: it is possible that K will simply stop playing a game in the middle and never make another move. This will not matter to us in the construction; what is important is that we have a winning strategy and if K does move we always have a winning response.

2.3 Formal Construction

Stage 0:

- Let F^* initially be defined as $F^*(x) = 2|x| + 2$.
- Let K be the standard prefix complexity function K, and K^* initially be the empty function.
- At all times throughout the construction, we have that $H^*(x) = \min(K^*(x) + 4, F^*(x) + 2)$. (In the case where $K^*(x)$ is undefined, let $H^*(x) = F^*(x) + 2$). We will denote by H_s the function H^* as it is at the beginning of stage s.

- For all e, set $i_e = 0$. In the future i_e will be the number of times R_e has been injured by the requirements $R_{e'}$, $1 \le e' \le e-1$.

- For all e, $j_{e,0} = 0$. In the future $j_{e,s}$ will be the number of times R_e has been injured by K by stage s. Unless otherwise mentioned we will assume that $j_{e,s+1} = j_{e,s}$. The fact that $j_{e,s}$ is the number of times R_e has been injured by K at stage s will not be used in the proof. We mention only as a tool of exposition.

Stage s (for $s \ge 1$):

Let (x', y') be the sth element in the fixed enumeration of $ov(K)$. Update K^* by setting $K^*(x') = \min(K^*(x'), y')$. (This automatically updates H^* as well)

(**) For $1 \le e \le s$ we consider requirement R_e.

Case 1: R_e is not active. We set the witness to be $(e, i_{e,s}, j_{e,s})$ which we denote (e,i,j). If $\gamma_{e,s}(e,i,j)\uparrow$ then go to (**) and process the next e. If $\gamma_{e,s}(e,i,j)\downarrow$ then *Activate* R_e and do the following:

1. Let $x = (e,i,j)$ for notation purposes only. (If other requirements are using a variable named x they are not affected by this.)

2. Let $\gamma_{e,x}(e,i,j) = \lambda_{e,x}$.

3. Set up the game $\mathcal{G}_{e,x}$ associated with $\lambda_{e,x}$ as described in the last section.

4. Determine FIRST's winning strategy and store it.

5. If FIRST's winning strategy involves trying to get to a node with $VAL = YES$ then do not put x into A (no other requirement has an interest in x so we know that x will never be put into A). If FIRST's winning strategy involves trying to get to a node with $VAL = NO$ then *do* put x into A (no other requirement has an interest in x hence this action will not injure any other requirement).

Case 2: R_e is active. Hence there is a witness $x = (e,i,j)$ and a game $\mathcal{G}_{e,x}$ in progress for which FIRST (i.e., the construction) has a winning strategy. There are a few sub-cases to consider.

1. K has not changed on X_e at all since the last stage. We do nothing.

2. K on X_e has changed so much that the move K plays causes his score to exceed ϵ (i.e. K "cheats"). Deactivate the requirement and set $j_{e,s+1} = j_{e,s} + 1$ (This means that the next attempt to satisfy R_e will use witness $(e, i, j+1)$. Note that i does not change.)

3. It is our turn in $\mathcal{G}_{e,x}$, either because the token is on the start node of the DAG or because K on X_e has changed in a way that his score does not exceed ϵ, so he has played a legal move.

   In this case we play the move dictated by our winning strategy (which we have stored). This may be to do nothing or it may involve moving the token to a new node, in which case we change H^* accordingly by enumerating elements into $ov(F)$.

If either case (2) or (3) occurs, we say that "R_e is acting" in which case for all $s \ge e' \ge e+1$, we deactivate $R_{e'}$ and set $i_{e',s+1}$ to $i_{e',s} + 1$ (This means that the next attempt to satisfy requirement $R_{e'}$ will use witness $(e', i+1, j)$.)

If R_e acts then proceed to the next stage. Otherwise return to (**) and process the next e.

END OF CONSTRUCTION

Claim 2: For all e, each R_e acts at most finitely often and is satisfied.
Proof of Claim 2:

We prove this by induction on e. Assume that the claim is true for all $e' < e$. We show that the claim holds for e. By the inductive hypothesis there exists a stage s' such that, for all $s \geq s'$, for all $e' < e$, $R_{e'}$ does not act at stage s.

Let $\mathcal{G}_{e,x}$ be the game associated with R_e at stage s. If $\mathcal{G}_{e,x}$ is never destroyed in a later stage, then (by construction) R_e will be satisfied (since for $H = \lim_{s\to\infty} H_s$, our winning strategy ensures that $\gamma_{e,x}[H]$ evaluates to YES if and only if x is not in A).

Suppose that $\mathcal{G}_{e,x}$ is destroyed at some point. Then, since by the inductive hypothesis R_e cannot be injured by higher priority requirements, by the rules of the construction it must be that the "player" K cheats on the game $\mathcal{G}_{e,x}$. In doing this, K is adding at least $\epsilon = 2^{-e-i_e-5}$ to $\sum_{x\in X_e} 2^{-K^*(x)}$ and hence to $\sum_{x\in\{0,1\}^*} 2^{-K^*(x)}$.

Once K cheats and destroys the game $\mathcal{G}_{e,x}$, a new witness x' is found and a new game $\mathcal{G}_{e,x'}$ is started during the next stage. Once again if this game is never destroyed then R_e will be satisfied. If this game is also later destroyed, this means that another $\epsilon = 2^{-e-i_e-5}$ is added to $\sum_{x\in X} 2^{-K^*(x)}$. The crucial observation is that since i_e did not change, this is the same ϵ as before.

This process keeps repeating. If the games associated with R_e continue to be destroyed indefinitely, then $\sum_{x\in\{0,1\}^*} 2^{-K(x)} \geq \epsilon + \epsilon + \cdots$ so it diverges. This contradicts K being a prefix free entropy function.

Hence eventually there is some game $\mathcal{G}_{e,x''}$ that is played throughout all the rest of the stages. Since the game DAG for $\mathcal{G}_{e,x''}$ is finite, this means that eventually R_e stops acting and is satisfied.
End of Proof of Claim 2

Claim 3: $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{4}$.
Proof of Claim 3:

We have that $H = \lim_{s\to\infty} H_s$, and thus

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} = \sum_{x\in\{0,1\}^*} 2^{-H_1(x)} + \sum_{s\geq 1} \sum_{x\in\{0,1\}^*} (2^{H_{s+1}(x)} - 2^{H_s(x)}).$$

That is, we can bound the sum by bounding H_1 and by bounding the changes that occur to H over the lifetime of the construction (some of which are made by K, and some by F).

Originally $H^*(x) = F^*(x) + 2 = 2|x| + 4$ for all x, so $\sum_{x\in\{0,1\}^*} 2^{-H_1(x)} = \frac{1}{16}$.

The total contribution that K can make to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ is bounded by $\sum_{x\in\{0,1\}^*} 2^{-K(x)+4}$. Since K is a prefix free entropy function, this contribution is at most 1/16.

Let us now consider the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ due to movements by F on games on which K eventually cheats. On each of these games F contributes less to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ than K, so from the above we can say that the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ while playing these games is at most 1/16.

Finally, let us consider the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ due to movements by F on games that are never destroyed, or are destroyed by higher priority requirements. Consider such games associated with a particular requirement R_e. During the first such game associated with R_e, $i_e = 0$, so F can change at most $\epsilon = 2^{-e-i_e-5} = 2^{-e-5}$. On the second such game associated with R_e, $i_e = 1$, so F can change at most $\epsilon = 2^{-e-6}$. Generalizing, we see that the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ on such games associated with R_e is

$$\sum_{i=5}^{\infty} 2^{-e-i} = 2^{-e}\sum_{i=5}^{\infty} 2^{-i} = 2^{-e-4}$$

Hence, the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ on games that are never destroyed, or are destroyed by higher priority requirements is at most $\sum_{e=1}^{\infty} 2^{-e-4} = \frac{1}{16}$.

Putting all this information together we have that

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{16} + \frac{1}{16} + \frac{1}{16} + \frac{1}{16} \leq \frac{1}{4}$$

End of Proof of Claim 3 ▮

References

[AFG10] E. Allender, L. Friedman, and W. Gasarch. Limits on the computational power of random strings. In preparation, see http://ftp.cs.rutgers.edu/pub/allender/limitsk.pdf, 2010.

[LV08] M. Li and P. Vitanyi. *Introduction to Kolmogorov Complexity and its Applications.* Springer, third edition, 2008.

[MP02] A. A. Muchnik and S. Positselsky. Kolmogorov entropy in the context of computability theory. *Theoretical Computer Science*, 271:15–35, 2002.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de