A Minimal Degree Below $0'$

Duarte Maia

July 1, 2024

1 Introduction

This document is a proof that there is a minimal degree below $0'$. It is my version of Sacks' proof as seen in [2]. I don't think it adds anything new, but I don't want a repeat of that time where I thought I'd figured out the proof of the effective saturated model theorem and months later found out that I had not. [Writer note: In the process of writing this document, I found out that my perception of the proof had a large mistake... Twice! So this document has already paid its dividends. If you find a mistake, please do not hesitate to contact me as well, and make it a third time...]

2 Spector's Degree

First, to ensure that we're on the same page, we prove Spector's weaker result:

Theorem 1. There exists a minimal degree below $0''$.

Proof: We build a decreasing sequence of perfect computable binary trees T_0, T_1, etc. More precisely, we will produce a $0''$-computable algorithm for the indices of these trees. The final set A shall be the unique set which is a path in all of these trees. (It will be reasonably obvious that some of the steps will fix an increasing initial segment, so A exists.)

First, we let T_0 be the full tree $2^{<\omega}$. Then, to construct the next tree in the sequence, we alternate between the following two tasks, for $e \in \mathbb{N}$:

(Making A noncomputable) Look at the first branching point of T_n; say it happens at depth k. Then, use a $0'$-oracle to check if $\{e\}(k)$ halts. If it does not, set $T_{n+1} = T_n$. If it does, set T_{n+1} as the branch of T_n which does the opposite of what $\{e\}(k)$ does. This ensures that our final set, which will be a path in T_{n+1}, *cannot* be equal to $\{e\}$.

(Making A minimal) We introduce a certain dichotomy to the tree, wherein either the final path A doesn't affect $\{e\}^A$ at all, or it affects it so drastically that from $\{e\}^A$ we can recover A again. This ensures that any set computable from A is either computable or computes A itself.

Let's be more precise about "A affects $\{e\}^A$ so drastically that the latter computes the former". The technical name for the property we are trying to encapsulate is: "T_{n+1} is an e-splitting tree". Before defining this notion, we introduce some helpful notation.

Definition 2. If T is a computable binary tree and $\sigma \in T$, we define for $e, x \in \mathbb{N}$ the (computable) expression $\{e\}^{\sigma *}(x)$ as follows: Dovetail over all $\tau \in T$ extending σ, checking if $\{e\}^{\tau}(x)$ halts. For the first found instance in which it does halt, output the resulting value.

The output may be undefined if no such extension exists. On the other hand, the output may or may not depend on the chosen way to dovetail. We say that $\{e\}^{\sigma *}(x)$ is *well and uniquely defined* if

for *every* $\tau \in T$ extending σ for which $\{e\}^\tau(x)$ halts, the output is the same.[1] Finally, we say that $\{e\}^{\sigma *}(x)$ is *strongly well and uniquely defined* if there is a τ extending σ in T such that $\{e\}^\tau(x)$ halts, and moreover there are no decision nodes in $[\sigma, \tau[$.[2]

Remark 3. The significance of strongly well and uniquely definedness is that it is something that may be computably confirmed.

Definition 4. Let T be a perfect computable binary tree. We say that T is e-splitting if, for every branching point $\sigma \in T$, there is an $x \in \mathbb{N}$ such that $\{e\}^{\sigma 0 *}(x)$ and $\{e\}^{\sigma 1 *}(x)$ are both strongly well and uniquely defined, but both take different values.[3]

The significance of these definitions is that, if T is e-splitting and A is a path on T, we can compute A from $\{e\}^A$ by the following procedure.

Algorithm 1 Computing A from $\{e\}^A$ when T is e-splitting.

Follow along the tree until you hit a decision point σ. Let σ_L be the first branching node extending $\sigma 0$, and σ_R the first branching node extending $\sigma 1$. Then, compute $a_L = \{e\}^{\sigma_L}(x)$ and $a_R = \{e\}^{\sigma_R}(x)$ for every value of $x \in \mathbb{N}$ (this computation will need to be dovetailed) until you find some value for which a_L and a_R are both defined and distinct; the assumption that T is e-splitting guarantees that this x exists. In this case, you look at which of these two agrees with $\{e\}^A(x)$, and continue following along the respective path until the next branching point. The algorithm continues in the same manner whenever you find a new branching.

Now, let us construct T_{n+1}. First, we ask the following question regarding T_n: Is there a node $\sigma \in T_n$ such that $\{e\}^{\tau *}$ represents the same (partial computable) function for every $\tau \in T_n$ extending σ? (That is, is $\{e\}^{\sigma *}(x)$ uniquely defined for all $x \in \mathbb{N}$, even if not necessarily well-defined?) If there is, we set T_{n+1} to be the subtree of T_n starting at σ. For the case that there is no such node, we introduce the following construction.

Definition 5. Let T be a perfect computable binary tree, and $e \in \mathbb{N}$ such that there is no node of T for which $\{e\}^{\sigma *}$ is uniquely defined. We define the *e-splitting computable subtree of T*, denoted $\mathrm{SplSbt}(T, e)$, as the tree T' given by the following algorithm.

At each stage, T' is given by a partially-defined tree, with a finite collection of nodes and lack thereof fully commited to, but at each point we will have some "leaves" below which we haven't committed to anything. Let σ be such a leaf. Since $\{e\}^{\sigma *}$ isn't uniquely defined (relative to T), we can (computably) find $x \in \mathbb{N}$ and two paths τ, τ' in T extending σ, such that $\{e\}^\tau(x)$ and $\{e\}^{\tau'}(x)$ are both well-defined and yield distinct values. Once they've been found, we commit to these two paths being the only extensions of σ in T'.

The algorithm proceeds recursively. At each stage, since τ and τ' must be incompatible extensions of σ, the tree is gaining at least one layer, so it is guaranteed that every node will be eventually be committed to being in or out of the tree.

Lemma 6. In the conditions of Definition 5, $\mathrm{SplSbt}(T, e)$ is necessarily an e-splitting tree.

We are now ready to finish constructing T_{n+1}: In the event that $\{e\}^{\tau *}$ is not uniquely defined for any $\tau \in T_n$, we set $T_{n+1} = \mathrm{SplSbt}(T_n, e)$.

[1] Of course, when we defined $\{e\}^{\sigma *}(x)$, it is presumed that we fixed some particular way to dovetail, so the expression "well and uniquely defined" doesn't immediately seem to make that much sense. The significance of the notion is that it is precisely the circumstances in which the value of the expression $\{e\}^{\sigma *}(x)$ is preserved when replacing T by a (computable) subtree.

[2] In english: If we start from σ and go down until we find a branching point, β, it must be the case that β equals τ or an extension of it.

[3] Our definition of e-splitting is slightly different (in a sense, weaker) from the one in the literature.

The only thing that remains to be addressed is verifying that the resulting degree is below $0''$. This consists of verifying that the question "is $\{e\}^{\tau*}$ uniquely defined for any $\tau \in T$?" can be answered by a $0''$-oracle. To verify that, we begin by noting that asking this question for *one particular* $\tau \in T$ is $0'$-computable, as we can consider a program which dovetails over all children of τ and every $x \in \mathbb{N}$, and halts if it finds an x and two values of τ for which this halts to different results; then we ask $0'$ if this program halts, which it does iff $\{e\}^{\tau*}$ is not uniquely defined. Then, we consider a $0'$-program which loops over all values of τ until it finds one for which $\{e\}^{\tau*}$ is uniquely defined, in which case it halts; $0''$ can tell whether this latter program halts, or equivalently if T contains a node τ such that $\{e\}^{\tau*}$ is uniquely defined. The proof is now complete. ■

3 Sacks' Degree

To verify that we can construct a minimal degree below $0'$, we will need to adapt the tools constructed above. The main issue lies in the fact that $0'$ cannot tell whether $\{e\}^{\tau*}$ is uniquely defined for some $\tau \in T$; it can confirm this fact once it finds such a value of τ, but if it hasn't found one yet, it can't rule out the possibility that there is a value of τ far, far away from where we are currently looking which does that.

One of the repercussions of this phenomenon is that, if we are thinking of adapting the above procedure (which we are), we may accidentally consider $\mathrm{SplSbt}(T, e)$ for an invalid index e, in the sense that, for some $\tau \in T$, $\{e\}^{\tau*}$ is uniquely defined. It is worthwhile to think about what will happen in this case. We get a sort of partially-computable tree, where from some nodes downward it may be that the answer to "is τ in T?" does not have neither a positive nor a negative answer. This is roughly what Odifreddi [1] defines as a *partial recursive tree* in page 494, and what Soare [2] defines as an *f-tree* in page 104. This object is nuancedly different from either a c.e. or a co-c.e. tree, in that there is some extra bookkeeping information that is present in neither of those. We will now provide a definition which is admittedly much more cumbersome than the definition in the literature, but which I feel place some emphasis in useful places which the usual definition does not.[4]

Definition 7. An *f-tree* is a partial computable function $T\colon 2^{<\omega} \rightharpoonup 2$ satisfying the following properties:

- If T is well-defined on σ, it is also well-defined on any prefix of σ,
- If $T(\sigma) = 1$, then T also yields 1 on any prefix of σ, ("tree-like")
- If $T(\sigma) = 0$, then T is also well-defined on any extension of σ,
- If $T(\sigma) = 1$ and T is well-defined on either $\sigma 0$ or $\sigma 1$, it is well-defined on both and at least one of them yields 1.

[4] It should be noted that our definition is also not exactly equivalent.

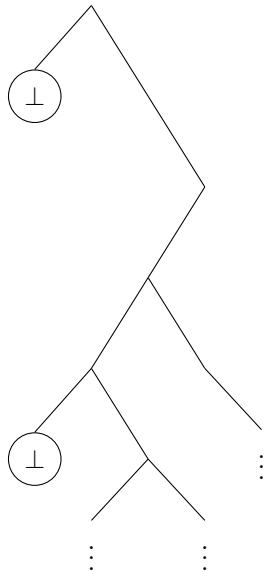

Figure 1: A diagram representing an f-tree.

The mental image that I have of this concept is of a binary tree with no dead ends but which may have some "defective fruits"; nodes below which there is no tree or lack thereof. These are what may happen in $\mathrm{SplSbt}(T, e)$ when T is not e-splitting.

Let us now begin discussing our approach. Our goal is to attempt to reproduce Sacks' construction to the best of our ability, given that we only have access to a $0'$-oracle. This means that we won't know at a given stage n (in which we are considering the index e, say) whether T_n is e-splitting or not, and so whether we should set $T_{n+1} = \mathrm{SplSbt}(T_n, e)$. Generically, we will be optimistic and do so unless given proof that this is a bad idea, namely if we find that T_{n+1} is only partially-defined; say that $T_{n+1}(\sigma)\uparrow$. In this case, morally, we change tacks: We know that $\{e\}^A$ is computable for any $A \in T_n$ extending σ, so we set $T_{n+1} = T_n \cap \sigma^{\downarrow}$, erase all the work we've done since stage n, and proceed with the algorithm. This begins to look like a finite injury procedure, where each requirement is injured at most once, but there is a big issue which throws a wrench in our plans.

The main problem is that, in order to make our final set A be $0'$-computable, it is not sufficient to have a set defined in the limit; we need to *commit* to elements being in or out of the set. In the procedure outlined in the previous paragraph, we are never actually committed: It might be that we defined a "bad" T_3, and with it we thought that our set started as 0101, but the ill-definedness is only visible very far down. So, we only find it by the time that we're around T_{10^8}, and we find that after all our set should have started as 0100 instead. And even now, we're not quite sure, because maybe T_2 will also result in an injury and we'll have to throw this out as well!

So, in summary, we need to add a commitment element to our algorithm. This isn't too hard; let's say that at each step to one new element and never change our mind. But then, a new complexity arises. Suppose that, in the example above, we have *already* committed to our initial segment being 0101, and we find that T_3 is undefined at 01000000. By our previous logic, we should undo our commitment so that $\{3\}^A$ is computable. But we can't. Now the question is: do we care?

Here is what I think is the main idea in order to understand Sacks' construction: Let's suppose that we've committed to an initial segment σ of our set. Then, how much do we care about the well-definedness of some T_e? In fact, put in a stronger way: Let's say that A is our final oracle, and that we never found any ill-definedness in T_e (but suppose also that we didn't try very hard). Then, what does T_e need to satisfy in order for Algorithm 1 to work? The answer is the following definition:

Definition 8. Let T be an f-tree, e a natural number, and A a path in T. We say that T is *A-e-weakly splitting* if, for every σ in A which is a branching point of T, there is $x \in \mathbb{N}$ such that $\{e\}^{\sigma 0*}(x)$ and $\{e\}^{\sigma 1*}(x)$ are both strongly well-defined and take distinct values.

Lemma 9. If T is an A-e-weakly splitting f-tree and $\{e\}^A$ is total, then $\{e\}^A \geq_{\mathrm{T}} A$.

Proof: We present a slight modification to Algorithm 1. This modification is necessary because, since T is only a partial recursive function, it need not be the case that a search for a "next branching node below this one" will ever halt.

Algorithm 2 Computing A from $\{e\}^A$ when T is A-e-weakly splitting.

Follow the tree T until you hit a branching point σ. Then, dovetail over $x \in \mathbb{N}$ and $\tau \in T$ extending σ, computing $\{e\}^\tau(x)$ along the way. Since T is known to be A-e-weakly splitting, we know that we will eventually find τ_0 extending $\sigma 0$, τ_1 extending $\sigma 1$, and $x \in \mathbb{N}$, such that $\{e\}^{\tau_0}(x)\downarrow \neq \{e\}^{\tau_1}(x)\downarrow$, and such that there are no branching points between τ_i and σi. This guarantees that the decision you make at σ irrevocably affects the value of $\{e\}^A(x)$, which you will then compare with your known value for $\{e\}^A(x)$. This informs you of whether you should follow along $\sigma 0$ or $\sigma 1$.
This procedure will work at any branching point you find. Thus, this algorithm will have you following along the path A.

This completes the proof. ■

Lemma 10. Let T be an f-tree and $e \in \mathbb{N}$. Then, if A is a path in $T' = \mathrm{SplSbt}(T, e)$, it must be the case that T' is A-e-weakly splitting.

Proof: Easy to verify by construction. As a sketch, given a branching point σ, we should consider the prefix σ_0 of σ which added σ's children to the tree (it may be the case that $\sigma_0 = \sigma$). By investigating what the algorithm does at this step, we find that indeed there is some x such that $\{e\}^{\sigma 0*}(x)$ and $\{e\}^{\sigma 1*}(x)$ are strongly uniquely well-defined. ■

Remark 11. It is not necessarily the case that $\mathrm{SplSbt}(T, e)$ admits *any* path A, even if T does!

We now have all the necessary ingredients. We will $0'$-computably construct a decreasing sequence of trees $T_0, \ldots, T_n$ at the same time that we iteratively construct a path $A = \{\sigma_n\}_{n \in \mathbb{N}}$ through them. When we add a node to the path, we commit to never removing it, but we retain the right to modify our trees. Namely, we will modify our trees if we find that it is not possible to ensure that a certain T_e is made A-e-weakly splitting, in which case we will change our approach to $\{e\}^A$ (we'll make it computable instead) and throw away our work on $T_e, T_{e+1}, \ldots, T_n$.

Theorem 12 (Sacks). There is a minimal degree below $0'$.

Proof: Consider the following $0'$-algorithm.

Algorithm 3 A minimal degree below $0'$.

Let $\sigma \in 2^{<\omega}$ be a variable path which starts empty, and T_0 (an index for) the full binary tree. Over the course of the algorithm, T_0 will never change.
At a given stage, suppose that we have defined a decreasing sequence of f-trees $T_0, \ldots, T_n$ and that σ is a node in T_n.
Suppose also as an inductive hypothesis that each T_{k+1} is either $\mathrm{SplSbt}(T_k, \tau)$ for some prefix τ of σ, or equal to T_k itself.
First, we make a new commitment. Use our oracle to check whether σ has a child in T_n.

- If so: Set σ_{new} to be this child.
- If not: Find the largest value of m such that T_m admits a child for σ (which exists because 0 is one such value) . Then, set σ_{new} to be this child, and forget about $T_{m+1}, \ldots, T_n$.

  For posteriority, let us say in this case that the index m is *marked*. However, forget any marks $\geq m$ that we have made until now.

Second, we ensure that our set is not computable. Let $T_0, \ldots, T_n$ be the current list, and σ the current node. Use our oracle to tell whether there is a branching point in T_n below σ. More precisely, attempt to find incompatible extensions τ_1, τ_2 of σ such that $T_n(\tau_1)\downarrow = T_n(\tau_2)\downarrow = 1$.

- If found: Let σ' be the node at which τ_1 and τ_2 diverge. Use our oracle to tell whether $x = \{e\}(\mathrm{length}(\sigma'))$ halts, where e is the least index not yet considered by this step. If it does, set $\sigma_{\mathrm{new}} = \sigma' y$, where $y = 1 - x$.
- If not: As in the first step, find the largest value of m such that T_m admits a branching point below σ (which exists because 0 is such a value), mark m and forget $T_{m+1}, \ldots, T_n$ as well as all marks past m, and repeat this step.

Third, we add a new tree to the list. Let $T_0, \ldots, T_n$ be the current list, and σ the current node. If n is unmarked, set $T_{n+1} = \mathrm{SplSbt}(T_n \cap \sigma^{\downarrow}, n)$, where $T_n \cap \sigma^{\downarrow}$ means the subtree of T_n consisting only of its nodes which are comparable with σ; otherwise, that is if m is marked, set $T_{n+1} = T_n$.
At each step of this procedure, σ is increasing in length. Our final set is the limit of the strings σ attained during this algorithm.

The above algorithm yields a well-defined $0'$-computable set A. It now remains to verify that it is indeed minimal. Note that the second step ensures that it is not computable. Thus, it remains to, given an arbitrary index e, verify that $\{e\}^A$ is either computable or computes A. It turns out that this is correlated with whether the index e has been marked by the algorithm or not.

First, a remark: As a consequence of the given algorithm, an index m will be marked a finite number of times. This is because, for the same node to be marked more than once, it must have been "wiped" inbetween consecutive times (otherwise, the conditions necessary to mark m would just as well have marked $m+1$ instead). Thus, we have in some sense finite injury: each index e is injured at most 2^e times. Thus, for every index e, there will eventually be a tree T_e, and indeed there will be a *limit* tree which will never change, namely after all indices prior to e have been maked as many times as they ever will. In the sequence, T_e should be taken to mean this limit tree.

Now, to investigate the degree of $\{e\}^A$, let us divide into two cases:

Case 1. (e is unmarked in the limit) In this case, A is a path through $T_{e+1} = \mathrm{SplSbt}(T_e \cap \sigma^{\downarrow}, e)$ for some σ, and so by Lemmas 9 and 10 we have $\{e\}^A \geq_{\mathrm{T}} A$.

Case 2. (e is marked in the limit) In this case, we have that there is an initial segment σ of A such that $T' = \mathrm{SplSbt}(T_e \cap \sigma^{\downarrow}, e)$ was eventually marked by the algorithm. This means that there is some initial segment of A, say τ, such that either τ has no children in T', or such that there are no incompatible extensions $\tau_1$and τ_2 of τ in T'. The latter case subsumes the former. Note that by definition of SplSbt,

this can only be the case if $\{e\}^{\tau*}$ is uniquely defined on $T_e \cap \sigma^{\downarrow}$. As a consequence, if $\{e\}^A$ is total, it *must* coincide with $\{e\}^{\tau*}$, which is a computable function.

This completes the proof. ■

Remark 13. Since the second step always performs a commitment, the first step in our algorithm is actually superfluous. Removing it provides an algorithm similar to Soare's [2].

References

[1] P. Odifreddi. *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*. ISSN. Elsevier Science, 1992.

[2] Robert I. Soare. *Recursively enumerable sets and degrees*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1987. A study of computable functions and computably generated sets.