Feedback Turing Computability, and Turing Computability as Feedback

Nathanael L. Ackerman
Dept. of Mathematics
Harvard University
Cambridge, MA 02138
Email: nate@math.harvard.edu

Cameron E. Freer
Computer Science and AI Lab
Massachusetts Institute of Technology
Cambridge, MA 02139
Email: freer@mit.edu

Robert S. Lubarsky
Dept. of Mathematical Sciences
Florida Atlantic University
Boca Raton, FL 33431
Telephone: +1 (561) 297–3340
Email: Lubarsky.Robert@comcast.net

***Abstract*—The notion of a feedback query is a natural generalization of choosing for an oracle the set of indices of halting computations. Notice that, in that setting, the computations being run are different from the computations in the oracle: the former can query an oracle, whereas the latter cannot. A feedback computation is one that can query an oracle, which itself contains the halting information about all feedback computations. Although this is self-referential, sense can be made of at least some such computations. This threatens, though, to obliterate the distinction between con- and divergence: before running a computation, a machine can ask the oracle whether that computation converges, and then run it if and only if the oracle says "yes." This would quickly lead to a diagonalization paradox, except that a new distinction is introduced, this time between freezing and non-freezing computations. The freezing computations are even more extreme than the divergent ones, in that they prevent the dovetailing on all computations into a single run.**

In this paper, we study feedback around Turing computability. In one direction, we examine feedback Turing machines, and show that they provide exactly hyperarithmetic computability. In the other direction, Turing computability is itself feedback primitive recursion (at least, one version thereof).

We also examine parallel feedback. Several different notions of parallelism in this context are identified. We show that parallel feedback Turing machines are strictly stronger than sequential feedback TMs, while in contrast parallel feedback p.r. is the same as sequential feedback p.r.

***Index Terms*—Computability theory, Turing machines, hyperarithmetic computability, least fixed points.**

AMS 2010 MSC: 03D10, 03D30, 03D60, 03D70

I. Introduction

In abstract computability theory, the most important type of non-computable operation is querying whether a computation halts. When one adds this ability, one obtains a new type of computation: computation relative to a halting oracle. This querying process can then be iterated, creating a transfinite hierarchy of computability strength, with each level of the hierarchy able to ask halting questions about lower levels.

This hierarchical view of computability strength invites the question, "What happens when one allows a computation to ask halting questions about computations of the same type as itself?" It was by considering this question in the specific context of Infinite Time Turing Machines (ITTMs, introduced in [3]) that the third author developed the notion of a Feedback Infinite Time Turing Machine (FITTM) in [10]. These FITTMs could be thought of as the most conservative notion of computation extending ITTMs that allows the computation to ask halting questions about other computations of the same type. While the FITTMs in [10] are presented as generalizations of ITTMs, the notion of an abstract feedback computation works for (almost) any model of abstract computation that allows for oracles. The fact that the development of feedback needs little prior theory, combined with its wide applicability, makes it a fundamental notion of computability theory.

In this paper, we will consider the notion of a *feedback Turing machine*, i.e., we will apply this notion of feedback to ordinary Turing machines. This will allow us to define the notion of *feedback Turing reducibility* in direct analogy with ordinary Turing reducibility. Specifically, a set of natural numbers X is feedback Turing reducible to Y when there is a feedback Turing machine with oracle Y that converges on all inputs and calculates the characteristic function of X. We will also show that feedback Turing reducibility is in fact the same thing as hyperarithmetical reducibility.

After that, we will show that those sets corresponding to the halting inputs of some feedback machine with oracle X are exactly the $\Pi^1_1(X)$ sets. While feedback Turing machines are able to ask about the halting of other feedback machines, there is still the issue of when such a machine becomes engaged in an infinite sequence of nested queries. When this happens, a feedback machine is said to *freeze*. In consequence, the collection of codes for feedback machines that freeze (relative to an oracle) is Turing equivalent to the hyperarithmetical jump (of the oracle). This provides another sense in which the hyperarithmetical jump is the natural analogue of the Turing jump for hyperarithmetical reducibility.

Then we turn from the question "what is the feedback version of Turing computability" to its converse, "what is Turing computability the feedback version of." As it turns out, for any natural and sufficiently strong properly sub-recursive collection, its feedback version yields exactly the (partial) computable functions. We detail here the case of the primitive recursive class. Actually, we distinguish two different ways one might model feedback here, and show that while one way does indeed produce all the computable functions, the other produces instead those sub-functions of the primitive recursive functions with computably enumerable domain.

The section thereafter addresses parallel feedback. One aspect of feedback is that some oracle calls about whether a computation halts do not end with an answer, but rather freeze the computation, so that no further progress is possible. You could address this by allowing for a separate kind of oracle call, which asks whether a computation freezes. This notion, which we discuss as "iterated feedback" in the last section, is perfectly good, but very powerful, and there is a weaker way to address that issue. One could present to the oracle an indexed family of computations, and ask whether one of them does not freeze. This goes beyond simple feedback, for while one could attempt to ask about the individual members of the family one at a time, if the first freezes then the computation at hand freezes once that first one is asked about. In contrast, the model here seems like running an entire family in parallel, and returning an answer as soon as one of them is seen not to freeze, hence the name. We discuss several variants of parallel feedback Turing computability, and isolate the one that seems to be of greatest interest. We show that it gets strictly more than sequential feedback Turing computability, and that in contrast parallel feedback p.r. is essentially the same as sequential feedback p.r.

Finally, we end this paper with a discussion of several possible research directions in feedback computability.

A. Related Work

The notion of a feedback machine was first introduced for the case of ITTMs by Lubarsky [10]. The feedback Turing machines in this paper can be viewed as the result of uniformly restricting all running times of an FITTM to be finite, where divergence occurs whenever a program would have run for an infinite amount of time.

In fact, even feedback ITTMs have a history, in that they were inspired by the analysis of the μ-calculus in [9]. First defined by Scott and DeBakker (unpublished), the μ-calculus is the fragment of second-order logic which, in addition to the first-order part, contains least and greatest fixed-point operators. (For background on the μ-calculus, see [1].) The source of its great expressive power is that the definition of a greatest fixed point can have as a parameter a least fixed point, which itself can depend on the parameter that same greatest fixed point – in other words, feedback.

Interestingly, there are other ways in which ITTMs relate to the current work. Whereas ITTMs allow a Turing machine ordinal time, Koepke [6] considers ordinal Turing machines, which have ordinal time *and space*, meaning tapes of length ORD. Later, with Seyfferth [8], he considers α-machines, which are OTMs with time and space restricted to a fixed ordinal α; they show that, for α admissible, their machines produce exactly the classical notions of α-recursive, α-r.e., and α-reducible [12]. Admissibility and hyperarithmetic theory are closely linked, so this is relevant; still, they are not the same. For an α-machine, the choice of α is fixed, even if α is no longer admissible relative to a given input, whereas the ordinals needed for hyperarithmetic computations are dependent on the input and cofinal through the countable ordinals.

Another kind of infinitary machine, also developed by Koepke [7], are (unresetting) infinite time register machines: registers can contain arbitrary natural numbers, and the machine runs for ordinal-many steps. He shows that the ITRM-computable reals are exactly the hyperarithmetic reals, and asks what fine structure theory might arise from allowing the registers to contain ordinals; one might also think to limit also the amount of time they are allowed to run. Given the result cited, we expect the answers would speak to our work; so far as we are aware, though, this has not yet been investigated.

A different way to represent the hyperarithmetic sets comes from the theory of computability on higher types. Let ${}^2E$ be the type-2 functional such that, for $f : \mathbb{N} \to \mathbb{N}$,

$${}^2E(f) := \begin{cases} 0, & \text{if } (\exists n \in \mathbb{N}) f(n) = 0, \text{and} \\ 1, & \text{otherwise.} \end{cases}$$

Kleene [5] (see also E-recursion in [12]) showed that a set of natural numbers is hyperarithmetic iff it is computable from ${}^2E$ (in Kleene's sense of recursion in higher types).

II. Feedback Turing Machines

A. Feedback Turing Machines

In the remainder of the paper, we will speak of (Turing) machines having the ability to make two types of queries. First, they can query an *oracle* $X : \omega \to 2$, and second, they can query a partial function $\alpha : A \to \{\uparrow, \downarrow\}$, called the *halting function*, where $A \subseteq \omega$. (By identifying ω with $\omega \times \omega$, such an A can sometimes be considered as a set of pairs.) The notation $\{e\}^X_\alpha(n)$ denotes the eth machine with oracle X and halting function α on input n. When a Turing machine queries the oracle, it is said to make an **oracle query**, whereas when a Turing machine queries the halting function it has made a **halting query**. A halting query behaves just like an oracle query, so long as the number n asked about is in the domain of α. If not, the computation *freezes*: since $\alpha(n)$ cannot return an answer, there is no next step, but the machine is not in a halting state, so it is not said to halt either.

Our first result states that for any oracle X, there is a smallest collection H of codes of machines for which the distinction between convergence and divergence is unambiguous.

Lemma 1: For any $X : \omega \to 2$ there is a smallest collection $H_X \subseteq \omega \times \omega$ such that there is a function $h_X : H_X \to \{\uparrow, \downarrow\}$ satisfying the following:

($\downarrow$) If $\{e\}^X_{h_X}(n)$ makes no halting queries outside of H_X and converges after a finite number of steps then $(e, n) \in H_X$ and $h_X(e, n) = \downarrow$, and conversely.

($\uparrow$) If $\{e\}^X_{h_X}(n)$ makes no halting queries outside of H_X and does not converge (i.e., runs forever) then $(e, n) \in H_X$ and $h_X(e, n) = \uparrow$, and conversely.

Furthermore, this h_X is unique.

Proof: For any halting function α, let

$$\begin{aligned} \Gamma^\downarrow(\alpha) &= \{(e, n) : \{e\}^X_\alpha(n) \text{ converges}\}, \\ \Gamma^\uparrow(\alpha) &= \{(e, n) : \{e\}^X_\alpha(n) \text{ diverges}\}, \\ h^{-1}_\alpha(\uparrow) &= \Gamma^\uparrow(\alpha), \text{ and} \\ h^{-1}_\alpha(\downarrow) &= \Gamma^\downarrow(\alpha). \end{aligned}$$

Then $h_{(\cdot)}$ is a monotone inductive operator. (For background on such, see [1], [9], [12].) Let h_X be its least fixed point, with domain H_X. These are as desired. ■

Definition 2: A **feedback Turing machine** (or **feedback machine** for short) is a machine of the form $\{e\}^X_{h_X}$ for some $e \in \omega$. The notation $\langle e \rangle^X(n)$ is shorthand for $\{e\}^X_{h_X}(n)$.

Then H_X is the collection of **non-freezing** computations and the notation $\langle e \rangle^X(n) \Downarrow$ means $(e, n) \in H_X$. If $(e, n) \notin H_X$ then $\langle e \rangle^X(n)$ is **freezing**, written $\langle e \rangle^X(n) \Uparrow$.

While not surprising, it bears mention that the h_X constructed in the preceding lemma as a fixed point of a certain operation is not the only such fixed point. By the Recursion Theorem, let e be a code of a machine which makes a halting query about itself; if it gets back $\downarrow$ it halts, and if it gets back $\uparrow$ it enters into a loop. Inductively, $e \notin H_X$. Also, the least fixed point (as in the previous lemma) starting with $h_X \cup \{\langle e, \uparrow \rangle\}$ contains both h_X and $\langle e, \uparrow \rangle$; similarly for $\langle e, \downarrow \rangle$. A similar construction shows that no such fixed point can have domain all of ω. Let e code a machine that queries $\alpha(e)$; if it gets back $\downarrow$ it loops, and if it gets back $\uparrow$ it halts. Such an e cannot be in the domain of any consistent halting function h.

B. The Tree of Sub-Computations

Just as a computation of a normal Turing machine converges if and only if there is a witness to this fact, a feedback machine is non-freezing if and only if there is a witness to that fact.

The idea is that, from the outside of our computation (freezing or not), one can imagine that any time a feedback machine makes a halting query, one creates a new feedback machine representing this query. One then runs this submachine to figure out what the result of the query should be, returning $\downarrow$ if the new machine converges and $\uparrow$ if it diverges (where each query, be it oracle or halting, is considered to take one time step). The tree of sub-computations is just a record of this process, organized naturally as a tree. We will see that a feedback machine is non-freezing if and only if its tree of sub-computations is well-founded. This tree is then a computational witness to the machine being non-freezing.

What follows is the definition of the tree $T = T^X(e, n)$ of sub-computations of the feedback machine $\langle e \rangle^X(n)$. It will be a subtree of $\omega^{<\omega}$; in fact, it will be a nice subtree, in that the set of successors of any node σ will be an initial segment of ω:

$$\{n \mid \sigma^\frown n \in T\} \leq \omega. \qquad (*)$$

Moreover, the nodes of the tree will be labeled with associated computations.

The root of the tree clearly is the empty sequence $\langle\rangle$, labeled with the main computation (e, n). We think of this root as being on top and the tree growing downwards. Because it will be important whether the tree is well-founded or not, the ordering of nodes $\leq$ is $\supseteq$: $\sigma \leq \tau$ iff $\sigma \supseteq \tau$ (that is, σ extends τ). So the tree is well-founded exactly when the relation $\leq$ is well-founded.

We can finally turn to the definition of T. This will be done inductively on the ordinals. At every ordinal, we describe how the computation proceeds. At some, not all, of these ordinal stages, nodes are inserted into T. This insertion is done depth-first. That is, nodes are included starting with the root and continuing along the left-most path. Once a terminal node is reached (if ever), we back up until we hit a branching node, and then continue down along the second-left-most path. Besides defining these ordinal steps, and the nodes and their labels, at every ordinal stage control is with one node.

At stage 0, control is with the root $\langle\rangle$, which is labeled with the index (e, n).

At a successor stage, if the computation at the node currently in control is in any state other than making a halting query, no new node is inserted into T, and the action of the computation is as with a regular Turing machine. If taking that action places that machine in a halting state, then, if there is a parent, the parent gets the answer "convergent" to its halting query, and control passes to the parent. If there is no parent, then the current node is the root, and the computation halts. If the additional step does not place the machine in a halting state, then control stays with the current node. If the current node makes a halting query, a new child is formed, after (to the right of) all of its siblings: in notation, if the current node is σ, then the new child is $\sigma^\frown k$, the lexicographically least direct extension of σ not yet used. Furthermore, this child is labeled with the index and parameter of the halting query; a new machine is established at that node, with program the given index and with the parameter written on the input tape; and control passes to that node.

At a limit stage, there are three possibilities. One is that on some final segment of the stages there were no halting queries made, and so control was always at one node. Then that computation is divergent. At that point, if there is a parent, then the parent gets the answer "divergent" to its halting call, and control is passed to the parent. If there is no parent, then the node in question is the root, and the entire computation is divergent.

A second possibility is that cofinally many halting queries were made, and there is a node ρ such that cofinally many of those queries were ρ's children. Note that such a node must be unique. Then ρ was active cofinally often, and as in the previous case ρ is seen to be divergent. So control passes to ρ's parent, if any, which also gets the answer that ρ is divergent; if ρ is the root, then the main computation is divergent.

The final possibility is that, among the cofinally many halting queries made, there is an infinite descending sequence, which is the right-most branch of the tree. This is then a freezing computation. The construction of the tree ends at this point.

Lemma 3: For each $e, n \in \omega$,

(1) $\langle e \rangle^X(n) \Downarrow$ if and only if $T^X(e, n)$ is well-founded, and
(2) if $\langle e \rangle^X(n) \Uparrow$ then $T^X(e, n)$ has a unique infinite descending chain, which is the right-most branch (using $\preceq_{T^X(e,n)}$ and $\leq_{T^X(e,n)}$) through the tree.

Proof: Since h_X and H_X are least fixed points, (1) follows by induction on their construction. For (2), the only way that $\langle e \rangle^X(n)$ can freeze is if it makes a freezing halting query. Once it does so, its computation does not continue, so the first such query is (e,n)'s right-most child. That means that everything to the left is well-founded (by part (1)). Continuing inductively, each freezing query itself has a freezing query for a child. ■

Now, $\langle e\rangle^X(n)\Downarrow$ if and only if there is a (unique) computational witness to this fact, but we can in fact get a bound on how complicated this witness can be. In the following, let $A(X){:=}L_{\omega_1^X}(X)$ be the smallest admissible set containing the real X. (For background on admissibility, see [12] or [2].)

Proposition 4: If $\langle e\rangle^X(n)\Downarrow$ then $T^X(e,n)\in A(X)$.

Proof: Inductively on the height of the tree $T^X(e,n)$. Notice that if the node $\langle k\rangle$ in $T^X(e,n)$ is labeled (e_k,n_k), then $T^X(e,n)$ restricted to the part beneath $\langle k\rangle$ is almost identical to the tree $T^X(e_k,n_k)$ (the only difference being the presence of the k at the beginning of every node). So each such $T^X(e_k,n_k)$ has smaller rank than $T^X(e,n)$, and hence is in $A(X)$. If there are only finitely many such k's, then it is a simple enough matter to string the $T^X(e_k,n_k)$'s together to build $T^X(e,n)$. If there are infinitely many such, then the admissibility of $A(X)$ will have to be used. It is a simple enough matter to give a Δ_1 definition of the function from k to $T^X(e_k,n_k)$, and that function suffices to build $T^X(e,n)$. ■

Note that if $(\forall n\in\omega)\langle e\rangle^X(n)\Downarrow$ then the sequence $\langle T^X(e,n):n\in\omega\rangle$ is in $A(X)$, as well as a sequence witnesses that the trees in the sequence do indeed satisfy the Δ_1 definitions of the $T^X(e,n)$'s.

Proposition 4 is the best possible, by the following two propositions.

Proposition 5: There is a $\mathbf{wf}\in\omega$ (independent of X) such that if $T\subseteq\omega^\omega$ is a well-founded tree satisfying $(*)$, and is computable in X, via index n say, then $T^X(\mathbf{wf},n)=T$. Moreover, if T is not well-founded, then $T^X(\mathbf{wf},n)$ will be the sub-tree of T consisting of those nodes lexicographically less than (i.e., to the left of) some node on T's left-most path.

Proof: Let $\langle\mathbf{wf}\rangle^X(n)$ be the program that runs as follows. Query in order whether each of $\langle 0\rangle,\langle 1\rangle,\langle 2\rangle,\ldots$ is in T. Whenever it is seen that $\langle k\rangle$ is in T, a halting query is made about $\langle\mathbf{wf}\rangle^X(n_k)$, where n_k is a code for T restricted to $\langle k\rangle$ (i.e., σ is in the restricted tree iff $k^\frown\sigma\in T$). Then the generation of the tree of sub-computations for $\langle\mathbf{wf}\rangle^X(n)$ is the depth-first search of T, from left to right, until the first infinite path is traced. ■

Proposition 6: The ordinal heights of the well-founded trees $T^X(e,n)$ are those ordinals less than ω_1^X.

Proof: By the previous proposition, it suffices to show there are computable (in X) trees of such height. This is fairly standard admissibility theory, but just to be self-contained we sketch an argument here.

This uses ordinal notations, also quite standard (see, e.g., [12]), described in the next sub-section for the convenience of the reader; we promise the argument won't be circular. The ordinal notations quite naturally present the ordinals as trees. That is, we define a function T such that, when n is a notation for the ordinal α, then $T(n)$ is a computable tree of height α. For $n=0$, the tree $T(0)$ consists of the empty sequence. For $n=2^e$, the tree $T(n)$ consists of the empty sequence and $T(e)$ appended at $\langle 0\rangle$. For $n=3\cdot 5^e$, the tree $T(n)$ consists of the empty sequence, and, for each k, $T(\{e\}(k))$ appended to $\langle k\rangle$. ■

Regarding freezing computations, $\langle e\rangle^X(n)$ is freezing exactly when during its run it makes a halting query outside of H_X, or, in other words, it makes a halting query about a freezing index. Because the computation cannot continue after that, this halting query is the right-most node on level 1 of $T^X(e,n)$. Similarly, that node makes a freezing halting query, and so on. So for a freezing computation, $T^X(e,n)$ has a unique infinite path, which is its right-most path. If $T^X(e,n)$ is truncated at any node along this path (i.e., the sub-tree beneath that node is eliminated), what's left is well-founded. This truncated tree can be built just the way trees for non-freezing computations can be built, with the use of the finite parameter of the path leading down to the truncation node, and so is a member of $A(X)$, hence with height less than ω_1^X. In fact, there are computations such that the heights of these well-founded truncated sub-trees are cofinal in ω_1^X, as follows.

Example 7: Let $A^*(X)$ be a non-standard admissible set with ordinal standard part ω_1^X. Let (e,n) be a non-freezing computation in the sense of $A^*(X)$ with $T^X(e,n)$ of height some non-standard ordinal. When run in the standard universe, (e,n) is as desired.

C. Feedback Reducibility

Having described the notion of feedback machines, we may now define feedback reducibility. Just as one set X is Turing reducible to Y when there is a Turing machine that with oracle Y computes the characteristic function of X, the set X is feedback reducible to Y when there is a feedback machine that with oracle Y computes the characteristic function of X. We make this precise.

Definition 8: Suppose $X,Y:\omega\to 2$. Then X is **feedback reducible** to Y, or **feedback computable** from Y, written $X\leq_F Y$, when there is an $e\in\omega$ such that

- for all $n\in\omega$, $\langle e\rangle^Y(n)\Downarrow$,
- for all $n\in\omega$, $\langle e\rangle^Y(n)\downarrow$, and
- for all $n\in\omega$, $\langle e\rangle^Y(n)=X(n)$.

It is easy to see that $\leq_F$ is a preorder.

It turns out that feedback reducibility is intimately connected with hyperarithmetical reducibility. Therefore we present the central definitions and results of that theory. For a more thorough treatment, with background and further citations, we refer the reader to [12].

Definition 9: Suppose $X,Y:\omega\to 2$. Then X is **hyperarithmetically reducible** to Y, written $X\leq_H Y$, when $X\in A(Y)$.

Definition 10: The **ordinal notations** (relative to X), which are an assignment of ordinals to certain integers, and are written $[n]_X=\alpha$, are as follows.

- $[0]_X=0$.

- If $[n]_X = \gamma$ then $[2^n]_X = \gamma + 1$.
- If e is such that $\{e\}^X$ is a total function, and for each n, $\{e\}^X(n)$ is an ordinal notation, with $[\{e\}^X(n)]_X = \lambda_n$, then

$$[3 \cdot 5^e]_X = \sup\{\lambda_n : n \in \omega\}.$$

Then $\mathcal{O}^X$, the **hyperarithmetical jump** or **hyperjump** of X, is the domain of this mapping.

For $n \in \mathcal{O}^X$, the **iterated Turing jump** $X^{[n]_X}$ is defined inductively:

- $X^{[0]_X} = X$.
- If $n = 2^{n'}$ then $X^{[n]_X}$ is the Turing jump of $X^{[n']_X}$, i.e., the set of those e such that $\{e\}^{X^{[n']_X}}(e) \downarrow$.
- If $n = 3 \cdot 5^e$ then $X^{[n]_X} = \{\langle m, i\rangle : i \in X^{[\{e\}^X(m)]_X}\}$.

Lemma 11: If $[n]_X = [k]_X$ then $X^{[n]_X}$ and $X^{[k]_X}$ have the same Turing degree.

Hence we can define the **αth Turing jump** of X, X^α, as the Turing degree of $X^{[n]_X}$, for any n such that $\alpha = [n]_X$. For which α is there such an n?

Lemma 12: The ordinals for which there is an ordinal notation are exactly the ordinals less that ω_1^X. Furthermore, for any real $R \subseteq \omega$, there is an $n \in \mathcal{O}^X$ such that $R \leq_T X^{[n]_X}$ iff $R \in A(X)$.

Returning to the actual subject at hand, we show that various of the objects above are feedback computable.

Lemma 13: There is a code **hj** (for "hyperarithmetical jump") where for any $X : \omega \to 2$ the feedback machine $\langle \mathbf{hj}\rangle^X(m, n)$ does the following:

- If $m \notin \mathcal{O}^X$ it freezes.
- If $m \in \mathcal{O}^X$ then $\langle \mathbf{hj}\rangle^X(m, n) = X^{[m]_X}(n)$ for all $n \in \omega$.

Proof: We start by defining several auxiliary feedback machines. First is $\langle r\rangle^X(e_0, e_1)$, which is intended to help with the case of $n = 3 \cdot 5^e$ in the definition of $\mathcal{O}^X$. Namely, r assumes that e_0 is telling the truth about membership in $\mathcal{O}^X$, and uses that to decide whether $3 \cdot 5^{e_1} \in \mathcal{O}^X$. Formally, let $\langle s\rangle^X(e_1, n)$ be the feedback machine that makes a halting query of $\{e_1\}^X(n)$. If it gets back the answer "diverges," then it freezes; if it gets back the answer "converges," then it halts. Then let $\langle t\rangle^X(e_1)$ (t for "total") go through each $n \in \omega$ in turn, and make a halting query of $\langle s\rangle^X(e_1, n)$. Notice that $\langle t\rangle^X(e_1)$ cannot halt. Rather, $\langle t\rangle^X(e_1)$ diverges iff $\{e_1\}^X$ is total, and freezes otherwise. Finally, let $\langle r\rangle^X(e_0, e_1)$ begin by making a halting query of $\langle t\rangle^X(e_1)$. If it gets the answer "diverges," it then runs through each $n \in \omega$, and makes a halting query of $\langle e_0\rangle^X(\{e_1\}^X(n))$. So $\langle r\rangle^X(e_0, e_1)$ freezes if $\{e_1\}^X$ is not total, or if $\langle e_0\rangle^X$ is not total on $\{e_1\}^X$'s range; else it diverges.

We now leverage r to build a machine which returns 1 on any input from $\mathcal{O}^X$ and freezes otherwise. Let $\langle h\rangle^X(m)$ be the feedback machine that does the following:

- If $m = 0$, it halts and returns 1.
- If $m = 2^{m'}$, it makes a halting query about $\langle h\rangle^X(m')$ and then returns 1.
- If $m = 3 \cdot 5^e$, it makes a halting query about $\langle r\rangle^X(h, e)$ and then returns 1.
- If m has another value, it freezes.

A straightforward induction shows that $\langle h\rangle^X(m) \Downarrow$ if and only if $m \in \mathcal{O}^X$, and in this case $\langle h\rangle^X(m) = 1$.

Next is the case of computing the Turing jump of a computable set. Let $\langle c^*\rangle^X(m, n)$ be the feedback machine that runs $\{n\}(n)$ as an oracle (not feedback) Turing computation, and anytime n makes an oracle call for a (i.e., queries whether a is in the oracle), it runs $\langle m\rangle^X(a)$ and uses the result as the response to the query. So if $\langle m\rangle^X$ is the characteristic function of some set Y, then $\langle c^*\rangle^X(m, n)$, as a function of n, converges on the Turing jump of Y and diverges on the complement. Then let $\langle c\rangle^X(m, n)$ be the code that asks a halting query of $\langle c^*\rangle^X(m, n)$, and returns 1 if it halts and 0 otherwise. So then c is the code that (as a function of n) computes the characteristic function of the Turing jump of Y.

Finally, define $\langle \mathbf{hj}\rangle^X(m, n)$ as follows. We will have use of the notation $\langle \mathbf{hj}^*_m\rangle^X(n) \simeq \langle \mathbf{hj}\rangle^X(m, n)$, which is well-defined by the Recursion Theorem. First call $\langle h\rangle^X(m)$. If this doesn't freeze, then $m \in \mathcal{O}^X$, and we have the following three cases:

- If $m = 0$, make an oracle query of $X(n)$ and return the result.
- If $m = 2^{m'}$, run $\langle c\rangle^X(\mathbf{hj}^*_{m'}, n)$ and return the result.
- If $m = 3 \cdot 5^e$, then for $n = \langle q, i\rangle$, first run $\{e\}^X(q)$ (which must converge as $m \in \mathcal{O}^X$)) with output a; then run $\langle \mathbf{hj}\rangle^X(a, i)$ and return the result.

A straightforward induction on the definition of $X^{[m]_X}$ shows that

- $\langle \mathbf{hj}\rangle^X(m, n) \Downarrow$ if and only if $m \in \mathcal{O}^X$, and
- for $m \in \mathcal{O}^X$, if $n \in X^{[m]_X}$ then $\langle \mathbf{hj}\rangle^X(m, n) = 1$, and otherwise $\langle \mathbf{hj}\rangle^X(m, n) = 0$.

■

The following lemma can be thought of as a stage comparison test.

Lemma 14: There is a code $\mathbf{hj}_\leq$ where for any $X \to 2$ the feedback machine $\langle \mathbf{hj}_\leq\rangle^X(m_0, m_1)$ does the following:

- If $m_1 \notin \mathcal{O}^X$, then $\langle \mathbf{hj}_\leq\rangle^X(m_0, m_1) \Uparrow$.
- Otherwise,
  - if $m_0 \in \mathcal{O}^X$ and $[m_0]_X \leq [m_1]_X$, then $\langle \mathbf{hj}_\leq\rangle^X(m_0, m_1) = 1$, and
  - if $m_0 \notin \mathcal{O}^X$ or $[m_0]_X > [m_1]_X$, then $\langle \mathbf{hj}_\leq\rangle^X(m_0, m_1) = 0$.

Proof: Define $\langle \mathbf{hj}_\leq\rangle^X(m_0, m_1)$ to be the code that does the following in order:

(0) Call $\langle \mathbf{hj}\rangle^X(m_1, 0)$.
(1) If $m_0 = 0$, return 1.
(2) If $m_0 = 2^{m_0'}$ and $m_1 = 2^{m_1'}$, then call $\langle \mathbf{hj}_\leq\rangle^X(m_0', m_1')$ and return the result.
(3) If $m_0 = 2^{m_0'}$ and $m_1 = 3 \cdot 5^{e_1}$, make a halting query on the following code:
  - At stage n compute $\{e_1\}^X(n)$, call the output a_n.
  - If $\langle \mathbf{hj}_\leq\rangle^X(m_0, a_n) = 1$ then return 1
  - Otherwise move to stage $n + 1$.

  If the code halts, then return 1; if not, return 0.
(4) If $m_0 = 3 \cdot 5^{e_0}$, make a halting query on the following code:

  - At stage n if $\{e_0\}^X(n) \uparrow$ then return 0; otherwise let $a_n = \{e_0\}^X(n)$.
  - If $\langle \mathbf{hj}_{\leq} \rangle^X(a_n, m_1) = 0$, then return 0.
  - Otherwise, move to stage $n+1$.

  If the code halts then return 0; if not return 1.

(5) Else return 0.

Clause (0) in the above ensures that $\langle \mathbf{hj}_{\leq} \rangle^X(m_0, m_1)$ will freeze if m_1 is not in $\mathcal{O}^X$.

Clause (1) ensures that if $[m_0]_X = 0$ then $\langle \mathbf{hj}_{\leq} \rangle^X(m_0, m_1) = 1$.

Clause (2) ensures that if $[m_0]_X = [m_0']_X + 1$ and $[m_1]_X = [m_1']_X + 1$ then $\langle \mathbf{hj}_{\leq} \rangle^X(m_0, m_1) = \langle \mathbf{hj}_{\leq} \rangle^X(m_0', m_1')$.

Clause (3) ensures that if $[m_0]_X = [m_0']_X + 1$ and

$$[m_1]_X = \lim_{n \to \infty} [\{e_1\}^X(n)]_X,$$

then $\langle \mathbf{hj}_{\leq} \rangle^X(m_0, m_1) = 1$ if and only if there is some n with

$$\langle \mathbf{hj}_{\leq} \rangle^X(m_0, \{e_1\}^X(n)) = 1.$$

That is, if $[m_0]_X$ is a successor ordinal and $[m_1]_X$ is a limit ordinal, then $[m_0]_X \leq [m_1]_X$ if and only if $[m_0]_X$ is less than or equal to one of the elements whose limit is $[m_1]_X$.

Clause (4) ensures that if

$$[m_0]_X = \lim_{n \to \infty} [\{e_0\}^X(n)]_X,$$

then $\langle \mathbf{hj}_{\leq} \rangle^X(m_0, m_1) = 1$ if and only if for every $n \in \omega$, both $\{e_0\}^X(n) \downarrow$ and $\langle \mathbf{hj}_{\leq} \rangle^X(\{e_0\}^X(n), m_1) = 1$ hold. That is, if $[m_0]_X$ is a limit ordinal, then $[m_0]_X \leq [m_1]_X$ if and only if every element of the sequence whose limit is $[m_0]_X$ is also less than or equal to $[m_1]_X$.

An easy induction then shows that

- If $m_0, m_1 \in \mathcal{O}^X$, then $\langle \mathbf{hj} \rangle^X(m_0, m_1) = 1$ if $[m_0]_X \leq [m_1]_X$, and $\langle \mathbf{hj} \rangle^X(m_0, m_1) = 0$ if $[m_0]_X > [m_1]_X$.
- If $m_1 \in \mathcal{O}^X$ and $m_0 \notin \mathcal{O}^X$, then $\langle \mathbf{hj} \rangle^X(m_0, m_1) = 0$.
- If $m_1 \notin \mathcal{O}^X$, then $\langle \mathbf{hj} \rangle^X(m_0, m_1) \Uparrow$.

■

The next lemma is well-known (e.g., see [12] II.5.6).

Lemma 15: For reals $X, Y : \omega \to 2$, there is an $n \in \mathcal{O}^Y$ such that $X \leq_T Y^{[n]_Y}$ (where $\leq_T$ is Turing reducibility) iff $X \in A(Y)$.

The following, which is the main result of this paper, shows that feedback reducibility and hyperarithmetical reducibility are in fact the same thing. This therefore tells us that feedback machines give us a model for hyperarithmetical reducibility.

Theorem 16: For any $X, Y : \omega \to 2$, we have $X \leq_F Y$ if and only if $X \leq_H Y$.

Proof: Suppose $X \leq_F Y$, and let $e \in \omega$ be such that $X(n) = \langle e \rangle^Y(n)$ for all $n \in \omega$. By the observation immediately after Proposition 4 we have $\langle T^Y(e,n) : n \in \omega \rangle \in A(Y)$. The output $\langle e \rangle^Y(n)$ can be computed from $T^Y(e,n)$, using the admissibility of $A(Y)$. (In a little detail, inductively on the tree $T^Y(e,n)$, the computation at a node can be run in ω-many steps, using the results from the children.) Hence $X \in A(Y)$, and $X \leq_H Y$.

Now suppose $X \leq_H Y$. By Lemma 15 we know for some $m \in \mathcal{O}^Y$, $X \leq_T Y^{[m]_Y}$. By Lemma 13, $Y^{[m]_Y} \leq_F Y$. Hence $X \leq_F Y$. ■

We then have the following as an easy corollary of Theorem 16.

Corollary 17:
For any $X, Y : \omega \to 2$ we have $X \leq_F Y$ if and only if X is $A(Y)$-recursive (i.e., Δ_1-definable over $A(Y)$).

III. Freezing Jumps and Feedback Semicomputable Sets

A. Freezing Jump

Just as how there is a natural Turing jump (the set of halting computations), there is a natural feedback jump: the set of non-freezing computations.

Definition 18: Fix a computable bijection $p : \omega \to \omega \times \omega$. Define the **feedback jump** of $X : \omega \to 2$ to be the function $X^{(f)} : \omega \to 2$ such that $X^{(f)}(a) = 1$ if and only if $p(a) = (e, n)$ and $\langle e \rangle^X(n) \Downarrow$.

The following lemma is then immediate.

Lemma 19: If $X \leq_F Y$ then $X^{(f)} \leq_F Y^{(f)}$. Furthermore, for any $X : \omega \to 2$, we have $X <_F X^{(f)}$.

Proof: The proofs are identical to their Turing jump counterparts. ■

We will show that the feedback jump is Turing equivalent to the hyperjump. First, though, we need the notion of a bounded computation. This is analogous to the notion of an Iterated Infinite Time Turing machine from [10].

Lemma 20: There is a feedback machine $\langle b \rangle^X(e, n, a)$ such that

- $\langle b \rangle^X(e, n, a) \Downarrow$ if and only if $a \in \mathcal{O}^X$,
- for $a \in \mathcal{O}^X$,
$$\langle b \rangle^X(e, n, a) \simeq \langle e \rangle^X(n) \Leftrightarrow \text{ht}(T^X(e, n)) \leq [a]_X,$$
and
- for $a \in \mathcal{O}^X$,
$$\langle b \rangle^X(e, n, a) = \ddagger \Leftrightarrow \text{ht}(T^X(e, n)) > [a]_X,$$
where ht returns the height of a tree.

(Implicitly, $\ddagger$ is a new symbol. Formally, identify ω with $\omega \cup \{\ddagger\}$.)

Proof: Notice that $T^X(e, n)$ is never empty, always having at least the root $\langle\rangle$. If that is all of $T^X(e, n)$, then we say the height is 0.

Let $\langle b \rangle^X(e, n, a)$ be the code that does the following:

- Run $\langle \mathbf{hj} \rangle^X(a, 0)$. Note that this will freeze if and only if $a \notin \mathcal{O}^X$. If it does not freeze, proceed as follows.
- Run $\langle e \rangle^X(n)$, except that any time a halting query for (e^*, n^*) is requested, do the following instead:
  - If $a = 0$ then stop and output $\ddagger$.
  - If $a = 2^{a'}$ then ask a halting query of $\langle b \rangle^X(e^*, n^*, a')$.
    * If the result is that it diverges then accept $\uparrow$ as a response to the halting query about (e^*, n^*) and continue with the simulation of $\langle e \rangle^X(n)$.

    * Otherwise, $\langle b\rangle^X(e^*,n^*,a')$ converges, say to c. If $c = \ddagger$ then stop and output $\ddagger$; otherwise accept $\downarrow$ as a response to the halting query about (e^*,n^*) and continue with the simulation of $\langle e\rangle^X(n)$.
  - If $a = 3\cdot 5^{e_a}$ then ask a halting query on the following code:
    * At stage m let $a_m = \{e_a\}^X(m)$.
    * If $\langle b\rangle^X(e^*,n^*,a_m)\uparrow$ then converge to 0.
    * If $\langle b\rangle^X(e^*,n^*,a_m)\downarrow$ and $\langle b\rangle^X(e^*,n^*,a_m)\neq\ddagger$ then converge to 0.
    * If $\langle b\rangle^X(e^*,n^*,a_m)=\ddagger$ then advance to stage $m+1$.

    If this code converges, then run the halting query for $\langle e^*\rangle^X(n^*)$ and pass the result back to $\langle e\rangle^X(n)$. Otherwise stop and output $\ddagger$.

The intuitive idea of the above code is that at each stage of the computation, we keep track of a bound on the size of the computational witness. Then, whenever a halting query is made of a pair (e^*,n^*), instead of asking it about (e^*,n^*), we instead ask it about $\langle b\rangle^X(e^*,n^*,a^*)$, where a^* is some smaller bound on the computational witness. Then if we can find such an a^*, we simply proceed as normal. But if we can't, then we return $\ddagger$, which signifies that our computational witness is too large. ■

Theorem 21: For any $X:\omega\to 2$, we have $X^{(f)}\equiv_T \mathcal{O}^X$.

Proof: It follows directly from Lemma 13 that $\mathcal{O}^X \leq_T X^{(f)}$.

To show that $X^{(f)}\leq_F \mathcal{O}^X$, notice that by Lemma 20 and Proposition 4,

$$\langle e\rangle^X(n)\Downarrow \quad \text{if and only if} \quad (\exists a\in\mathcal{O}^X)\langle b\rangle^X(e,n,a)\neq\ddagger.$$

The latter is Σ_1-definable over $A(X)$. So $X^{(f)}$ is Σ_1-definable over $A(X)$ and hence Turing reducible to $\mathcal{O}^X$. ■

B. Feedback Semicomputability

Definition 22: A set $B\subseteq\omega$ is **feedback semicomputable** (in X) when there is an $e\in\omega$ such that $b\in B\Leftrightarrow\langle e\rangle^X(b)\Downarrow$.

In particular, it is easy to see that for any X, $\{(e,n):\langle e\rangle^X(n)\Downarrow\}$ is feedback semicomputable.

Feedback semicomputable sets look like the analogues of computably enumerable sets, and this is confirmed by the following.

Proposition 23: A set $B\subseteq\omega$ is feedback semicomputable in X if and only if it is Σ_1-definable over $A(X)$, i.e., in $\Pi^1_1(X)$.

Proof: Suppose $b\in B\Leftrightarrow\langle e\rangle^X(b)\Downarrow$. By Lemma 3 and Proposition 4, $b\in B$ if and only if there is a function in $A(X)$ witnessing the well-foundedness of $T^X(e,b)$, which provides the desired Σ_1 definition.

Now suppose $b\in B$ is Σ_1-definable over $A(X)$. Thus $B\in\Pi^1_1(X)$. Hence there is a relation $R_B\subseteq\omega^{<\omega}\times\omega$ computable in X satisfying

$$b\in B\Leftrightarrow \text{“}R_B(\cdot,b)\text{ is well-founded”}.$$

Let $\langle e^*\rangle^X(b,\sigma,i)$ (where $\sigma\in\omega^{<\omega}$ and $i\in\omega$) be the following program:

- Make a halting query about the program which searches for the least $j\geq i$ such that $R_B(\sigma^\frown j,b)$ holds.
- If the answer comes back “divergent” then stop.
- If the answer comes back “convergent” then find the least such j. Make a halting query about $\langle e^*\rangle^X(b,\sigma^\frown j,0)$. Then run $\langle e^*\rangle^X(b,\sigma,j+1)$.

Let $\langle e\rangle^X(b)$ be $\langle e^*\rangle^X(b,\emptyset,0)$. By construction, $T^X(e,b)$ is $R_B(\cdot,b)$ (with a few extra nodes thrown in, by the first step, which do not affect well-foundedness). So $\langle e\rangle^X(b)\Downarrow$ if and only if $R_B(\cdot,b)$ is well-founded, which holds if and only if $b\in B$. ■

Corollary 24: A set is feedback computable in X if and only if both it and its complement are feedback semicomputable.

It is worth mentioning that the argument that a set is computable if and only if it and its complement are computably enumerable does not lift to this context. Is there a way to uniformly transform a pair (e_0,e_1) into a code e where e_0 (resp. e_1) witnesses the feedback semicomputability of B (resp. B's complement), and $e(B)$ witnesses the feedback computability of B?

We next show that the range of any total feedback computable function is a feedback computable set. Hence, unlike with computably enumerable sets, not every feedback semicomputable set is the range of some total feedback computable function.

Lemma 25: If $(\forall n)\langle e\rangle^X(n)\Downarrow$ and $b\in B\Leftrightarrow(\exists n)\langle e\rangle^X(n)=b$, then B is feedback computable.

Proof: Let e^* be such that $\langle e^*\rangle^X(b)$ is the program that calls e on each $n\in\omega$ and halts if and only if b is ever returned. Note that $\langle e^*\rangle^X(b)$ never freezes, by our assumption that $(\forall n)\langle e\rangle^X(n)\Downarrow$. Now let f be such that the program $\langle f\rangle^X(b)$ returns 1 if $\langle e^*\rangle^X(b)$ halts and 0 otherwise. Then $\langle f\rangle^X$ is a total feedback computable function that computes the characteristic function of B. ■

IV. Turing Computability as Feedback

Our purposes here are twofold: substantive and methodological. For the former, we want to do the inverse of the previous sections. Whereas earlier we answered the question, what is feedback Turing computability, now we want to answer the question, what is Turing computability the feedback of. Regarding the latter, we would like to see how feedback can be done when the computation is not thought of as step-by-step. Until now, the notions of computation for which feedback has been worked out, namely Turing and infinite time Turing, are thought to run by one step proceeding after another. This leads to a very simple feedback model. When a halting query is made, a new entry is made into the tree of sub-computations, and whenever that query is answered, the answer is passed to the querying instance, and the computation resumes. This works fine, but there are other models of computation. Even within the Turing algorithms, there are different kinds of programming languages, which capture differing computational paradigms. The step-by-step model captures the run of an imperative program. Functional and declarative programming, in contrast, run differently. So it might turn out to be useful for the study of machine computability to see how feedback could be implemented in a different context, to say nothing of other more abstract uses in mathematics, which considers computations way beyond Turing procedures.

For these reasons, we consider feedback primitive recursion. We will show that it yields exactly the (partial) Turing computable functions. Furthermore, they are not presented in an imperative style, but rather functional, as are the recursive functions. (For the sake of definiteness, we use as our reference for the primitive recursive and the recursive functions [13].) This naturally leads to a nested semantics, as is most easily seen in the definition of Kleene's T predicate ([13], Theorem 7.2.12): a witness to a computation $\{e\}(x)$ exists only when the computation converges.

When turning to feedback p.r., one is quickly struck by two differences to feedback Turing. For one, all of the base functions are total. One might immediately ask, does this mean that all of the halting queries should come back with "yes"? That is not the case, as evidenced by the self-querying function. That is, there is still a function which asks the halting oracle whether it itself halts. This represents a freezing computation. What it does mean, as we shall see, is that there are no divergent computations, no computations that loop forever. Every computation either halts or gets stuck at some finite step, without being able to proceed. One could then ask whether anything has been gained by allowing for feedback. After all, if a halting query is freezing, then the computation itself freezes, and if the halting query is not freezing, then we already know the answer. There are two possible rejoinders. One is that there is information in the distinction between the freezing and the non-freezing computations, providing a certain kind of enumerability, akin to computable enumerability.

The other rejoinder also speaks to the other difference with feedback Turing computability. For Turing computability, an oracle is a set. For primitive recursion, the closest thing to an oracle is a function. That is, the primitive recursive functions are those generated by some base functions, closing under certain inductive schemes. If you want to include more, you would most naturally include another function f among the base functions, thereby getting the functions primitive recursive in f. So one is led to consider a halting oracle that returns not whether a computation halts (especially since we know it always does, when non-freezing), but rather the value of a computation.

With these considerations as motivation, we are now prepared to formalize the notions involved.

Definition 26: For a partial function f (from ω to ω), the f-primitive recursive functions are those in the smallest class containing f, the constant functions, projection, and successor, and closed under substitution and primitive recursion (cf. [13], Section 7.1). Implicitly, when defining $g(\overrightarrow{n})$, if any of the intermediate values are undefined, then so is $g(\overrightarrow{n})$.

We will need to use standard integer codings of the f-p.r. functions. Notice that these names can be defined independently of any choice of f. One can simply introduce a symbol for f, leaving it uninterpreted, and consider all the names so generated. If e is such a code, we will refer to e as an *oracle p.r. index*. We will use the standard notation from computability theory $\{e\}^f(\overrightarrow{n})$ for the application of the e^{th} oracle p.r. function, with oracle f, to a tuple of inputs. This makes sense even in a context with codes for non-p.r. functions, since it is easily computable whether e codes an oracle p.r. function (and if not $\{e\}^f(\overrightarrow{n})$ can be given any desired default value).

Theorem 27: There is a smallest set $H \subseteq \omega$, and unique function $h : H \to \omega$ such that, for all $\langle e, \overrightarrow{n}\rangle \in H$,

- if e is not an oracle p.r. index, or if arity(e) $\neq$ arity($\overrightarrow{n}$), then $h(e, \overrightarrow{n})$ = ERROR (some default value),
- else $h(e, \overrightarrow{n}) \simeq \{e\}^h(\overrightarrow{n})$.

Proof: H is the least fixed point of a positive inductive definition. (For more background, see [1], [9], [12].) ■

Definition 28: The **feedback primitive recursive functions** are the h-p.r. functions, with the h from the preceding theorem.

Theorem 29: The feedback primitive recursive functions are exactly the partial computable functions.

Proof: In one direction, we must show only that h is computable. The least fixed point construction of $h(e, \overrightarrow{n}) \simeq \{e\}^h(\overrightarrow{n})$ is naturally given by a finitely branching tree, with the finite branching corresponding to the substitution and primitive recursion calls. The construction of the tree is uniformly computable in e and $\overrightarrow{n}$. If the tree is infinite then it's ill-founded, and $h(e, \overrightarrow{n}) \simeq \{e\}^h(\overrightarrow{n})$ is undefined. If not, then the value is computable.

In the other direction, we will use the fact that if x is a tuple coding finitely many steps in a Turing computation of $\{e\}(\overrightarrow{n})$, then it is p.r. in that data either to extend x by one step of the computation, for which we use the notation x^+, or recognize that x ends in a halting state. Consider the feedback p.r. function $f(e, \overrightarrow{n}, x)$ which returns

- ERROR, if x is not an initial run of $\{e\}(\overrightarrow{n})$, else
- the output of the computation, if x ends in a halting state, else
- $h(f, \langle e, \overrightarrow{n}, x^+\rangle)$.

Then $\{e\}(\overrightarrow{n}) = f(e, \overrightarrow{n}, \langle\rangle)$. ■

There remains the question of what would happen if, instead of considering h to ω, returning the output of a computation, we took h to tell us merely that a computation converged. This is the goal of what follows.

Theorem 30: There is a smallest set $H \subseteq \omega$ such that, for h the unique function $h : H \to 1$, $H = \{\langle e, \overrightarrow{n}\rangle \mid e$ is an oracle p.r. index, arity(e) = arity($\overrightarrow{n}$), and $\{e\}^h(\overrightarrow{n})$ converges$\}$.

Proof: H is the least fixed point of a positive inductive definition. ■

Definition 31: The **convergence feedback primitive recursive functions** are the h-p.r. functions, with the h from the preceding theorem.

Lemma 32: Let f and g be partial functions, and e an oracle p.r. index. If $f \subseteq g$ then $\{e\}^f \subseteq \{e\}^g$.

Proof: An easy induction on the definition of $\{e\}$. ■

Corollary 33: Every convergence feedback p.r. functions is a sub-function of a primitive recursive function.

Proof: Letting **0** be the constant function with value 0, note that $h \subseteq \mathbf{0}$ (for h from the previous theorem). Then $\{e\}^h \subseteq \{e\}^{\mathbf{0}}$, and of course **0** is p.r. ■

Theorem 34: The convergence feedback primitive recursive functions are exactly the sub-functions of the p.r. functions with computably enumerable domains.

Proof: For e a feedback p.r. index, the computations of $\{e\}^h(\overrightarrow{n})$ for the various $\overrightarrow{n}$'s can be computably simulated and dovetailed, making the domain of $\{e\}^h$ enumerable.

In the other direction, given a c.e. set W, let w be a Turing index with domain W. Consider the convergence feedback p.r. function $f(w, \overrightarrow{n}, x)$ which returns

- ERROR, if x is not an initial run of $\{w\}(\overrightarrow{n})$, else
- 0, if x ends in a halting state, else
- $h(f, \langle w, \overrightarrow{n}, x^+\rangle)$,

where x^+ is as above. Then $\overrightarrow{n} \in W$ iff $f(w, \overrightarrow{n}, \langle\rangle) = 0$.

Finally, for e a p.r. index, let $g(\overrightarrow{n})$ be

$$\{e\}(\overrightarrow{n}) \cdot (1 + h(f, \langle w, \overrightarrow{n}, \langle\rangle\rangle)).$$

■

V. Parallel Feedback

A variant of feedback, as identified in [10], is **parallel feedback**. Imagine having an infinite, parametrized family of feedback machines, and asking, "does any of them not freeze?" It is not clear that this could be simulated by the sequential feedback machines from above. Perhaps this is surprising, because the situation is different for regular Turing machines. With them, you could use a universal machine to run all of the Turing machines together, by dovetailing. But with feedback, this is not possible. For sure, you could start to dovetail all of the feedback computations. But as soon as you make a freezing oracle call, the entire computation freezes. Similarly, for a parallel call, one might first think of going down the line until one finds a non-freezing machine, and certainly a feedback machine could ask the oracle "does the first in the family not freeze?" But if you're unlucky, and the first machine does freeze, then so does your computation right then and there; if a later one doesn't, you'll never get there to find out.

For parallel feedback infinite time Turing machines as explored in [10], it was left open there whether or not they yield more than sequential FITTMs. That unhappy fact notwithstanding, we show below that parallel feedback TMs get you more than the sequential version, but parallel feedback p.r. is the same as its sequential version.

A. Parallel Feedback Turing Machines

We should start by identifying the formalism of asking the oracle about a family of computations. To simplify notation, we will consider only unary functions; if we wanted to consider a binary function $\langle e\rangle^X(m, n)$, with, say, a fixed m as a parameter, then that value of m could be packed into e, or alternatively with m as a variable, then m and n could be packaged into an ordered pair. So then a question to the oracle will be an integer e, and will call for a positive response if $\langle e\rangle^X(n)$ does not freeze for some n, and will itself freeze otherwise.

At this point we must consider what the response would be if indeed, for some n, $\langle e\rangle^X(n)$ does not freeze. There are several options. One is a simple "yes." Another is a witnessing input n to this non-freezing. To go this route, n would have to be chosen in some canonical way, to keep the computation deterministic. The most natural way seems to be to minimize the ordinal height of the tree of sub-computations, in case of a tie returning the smallest integer (in the natural ordering of ω) of that bunch. In the end, that did not seem like a good way to go. As we will see, there's already a lot of power just restricting the trees to be of height 1, and the natural ordering of ω has little to do with computation: the simplest re-ordering of ω would produce wildly different results here, showing that notion of computation to be non-robust. A more natural way to go is to punt on the determinism. Allow the oracle to return any n such that $\langle e\rangle^X(n)$ does not freeze. There is a choice to be made in this definition. For some n's that might be returned, the current computation could freeze, and for others not. So it is possible that some runs of a computation freeze and others not. So when we say that an oracle call of e will return "any n such that $\langle e\rangle^X(n)$ does not freeze," does that mean that some run does not freeze, or that all runs do not freeze? Both notions seem interesting. We work here with the former.

Finally, we will have the oracle return not just some n with $\langle e\rangle^X(n)$ not always freezing, but also some possible output of $\langle e\rangle^X(n)$. To be sure, one could simulate the calculation of $\langle e\rangle^X(n)$ within the current computation instead, so nothing is gained or lost by doing this when the output is finite. The difference emerges when a computation diverges, meaning the output is $\uparrow$. For the part of the construction below that makes essential use of the parallelism, we will not need the output to be handed to us. The reason we're taking it anyway is to combine the parallel calls with the halting queries. That is, to simulate a halting query, one need only ask about a family of computations $\langle e\rangle^X(n)$ that do not depend on n. If a natural number is returned as an output, then the original computation converges; if $\uparrow$, then it diverges. A finer-grained analysis could have halting queries separate from parallel calls, the latter of which return only a non-freezing input.

Yet another option is to have a possible response give only the output $\langle e\rangle^X(n)$ and not the input n, from which n is not obviously computable; this strikes us as less natural, and so we mention it only in passing, to be thorough. For some discussion on all of these options, see the sub-section on parallelism in the final section.

In the following, we recycle some of the terminology and notation from earlier. This includes a computation making a halting query, even though we now interpret this as being a parallel halting query. The same notation $\langle e\rangle^X(n)$ will be used for parallel feedback computation.

Definition 35: Given $X : \omega \to 2$ and $H : \omega \times \omega \to \mathcal{P}(\omega \cup \{\uparrow\})$, a **legal run** of $\langle e\rangle^X_H(n)$ is (some standard encoding of) a run of a Turing machine calculation of $\{e\}^X_H(n)$, in which, whenever a halting query f is made, the answer is of the form $\langle m,k\rangle$, where $k \in H(f,m)$. (Implicitly, if $H(f,m)$ is always empty, then the computation freezes at this point.) If the last state of a finite legal run is a halting state, then the content on the output tape is the output of that run. If the legal run is infinite, then $\uparrow$ is the output of that run. A **legal output** is the output of a legal run.

Notice that the set of legal runs is not absolute among models of ZF. Suppose, for instance, that $H(f,m) = \{0,1\}$. Consider a computation that just keeps making the halting query f. Then both $\langle m,0\rangle$ and $\langle m,1\rangle$ are always good answers, so the legal runs depend on the reals in the model. Nonetheless:

Lemma 36: Whether k is a legal output of $\langle e\rangle^X_H(n)$ is absolute among all standard models of ZF.

Proof: The collection of all legal runs can be organized via a natural tree. A node of this tree corresponds to a computational stage in a legal run. The root is the beginning of the computation, and we think of the tree as growing downwards. At a node, continue the computation until the next halting query. The children of the node are the possible answers (given H). This tree is absolute.

A legal run is a maximal path through the tree. Such a path is either finite or infinite. The finite paths are absolute, as they correspond to terminal nodes (leaves). A finite run ends either in a halting state, and so gives a finite output, or by asking a halting query with no answer, and so thereby freezes. Hence the integer legal outputs are absolute. An infinite legal run is given by an infinite descending path. While the set of such paths is not absolute, whether the tree is well-founded or not is absolute among all standard models, so whether $\uparrow$ is a legal output is absolute. ∎

Definition 37: For any $X : \omega \to 2$ and $H : \omega \times \omega \to \mathcal{P}(\omega \cup \{\uparrow\})$ let $H^+(e,n)$ be the set of legal outputs of $\langle e\rangle^X_H(n)$.

Lemma 38: There is a smallest function H such that $H = H^+$.

Proof: The operator that goes goes from H to H^+ is a positive inductive operator: as any $H(f,m)$ increases, so does the set of legal runs. So the least fixed point exists, and is the desired H. ∎

Definition 39: $\langle e\rangle^X(n)$ refers to $\langle e\rangle^X_H(n)$, with H as from the lemma above.

If a computation can have more than one output, what would it mean to compute a set?

Definition 40: A function $f : \omega \to \omega$ is **computable from** X **via** e if, for all $n \in \omega$, $f(n)$ is the only legal output of $\langle e\rangle^X(n)$. A set $A \subseteq \omega$ is **computable from** X **via** e if its characteristic function is so computable. Notice that in both cases $\langle e\rangle^X(n)$ could still have freezing legal runs.

Theorem 41: Parallel feedback can compute more than sequential feedback. In particular, $\mathcal{O}^X$ is parallel feedback computable from X.

Proof: From the subsection on feedback semicomputability, there is a (sequential, hence also parallel) machine f which does not freeze (and WLOG outputs 1) on input n iff $n \in \mathcal{O}^X$.

To handle the case of those $n \notin \mathcal{O}^X$, consider the following computation g. If at any time it considers a number not of the form 0, 2^e, or $3{\cdot}5^e$, then g outputs 0, because it is immediately clear that number is not in $\mathcal{O}^X$. When considering 0, g freezes, because 0 is in $\mathcal{O}^X$. When considering 2^e, g moves on the e. When considering $3 \cdot 5^e$, first g checks whether $\{e\}^X$ is total. If not, then the machine halts. Else it picks an n non-deterministically, and then continues the computation from that n. This g, when run on any $n \in \mathcal{O}^X$, will always freeze, because the machine will eventually consider 0. When started on some $n \notin \mathcal{O}^X$, it is possible that some legal runs will freeze too, depending upon the non-deterministic choice made. But there will be at least one legal run that does not freeze: since $n \notin \mathcal{O}^X$, we know $n \neq 0$; if $n = 2^e$ then $e \notin \mathcal{O}^X$; else if n is not of the form $3 \cdot 5^e$ then the computation halts; else there is some k such that $\{e\}^X(k) \notin \mathcal{O}^X$, so the computation can continue. Hence there is a legal output (either 0 or $\uparrow$).

Now consider the machine $\langle h\rangle^X(e,i)$, which, for i even, returns $\langle f\rangle^X(e)$, and, for i odd, returns $\langle g\rangle^X(e)$. Running $\langle h\rangle^X(e,\cdot)$ in parallel, an even i is returned iff $e \in \mathcal{O}^X$, and an odd i is returned iff $e \notin \mathcal{O}^X$. In the former case, output 1, in the latter output 0. ∎

B. Parallel Feedback Primitive Recursion

The essence of sequential feedback p.r. was a least (i.e., smallest domain) function h such that $h(e,\overrightarrow{n}) \simeq \{e\}^h(\overrightarrow{n})$, for e an oracle p.r. index. Using the same paradigm as in the previous sub-section, by the nature of non-determinism, there is more than one possible output. So we consider h as a *multi-valued function*, which we take to be a function from $\omega \times \omega$ to $\mathcal{P}(\omega)$.

In what follows, we make use of the fact that an oracle p.r. computation is most naturally expressed as a finite tree, in which the splitting corresponds to substitution and primitive recursion, and the terminal nodes to applications of the base functions.

Definition 42: For H a multi-valued function, a **legal computation of** $\{e\}^h(n)$ (from H) is an oracle p.r. computation in which any value taken for $h(f,m)$ is a member of $H(f,m)$. A **parallel computation of** $\{e\}^h(n)$ (from H) is an oracle p.r. computation in which any value taken for $h(f)$ is of the form $\langle m,k\rangle$, where $k \in H(f,m)$.

Proposition 43: There is a smallest multi-valued function h such that $h(e,n)$ is the collection of outputs of all possible parallel computations of $\{e\}^h(n)$ from h. (The notion of h being smaller than g is taken pointwise: for all e and n, $h(e,n) \subseteq g(e,n)$.)

Proof: Take h to be the least fixed point of the obvious positive inductive definition. ∎

Definition 44: The **parallel feedback p.r. functions** are those multi-valued functions f given by some oracle p.r. index e using parallel computations from the h of the previous proposition.

Theorem 45: The parallel feedback p.r. functions are those multi-valued functions f such that $f(n)$ is computably enumerable, uniformly in n.

Proof: Given an oracle p.r. index e, a computation of $\{e\}^h(n)$ can be run computably, uniformly in e and n. Any occurrence of $h(g)$ can be handled by dovetailing the computations of $\{g\}^h(m)$ for all m. As outputs are generated in the sub-computations, they are then used by the calling instances, making the set of outputs computably enumerable.

In the other direction, let $f(n)$ be a c.e. set, uniformly in n. We identify $f(n)$ with a Turing code for the corresponding c.e. set, so that f is taken to be a Turing code also. (In the end, the set in question is the range of $\{\{f\}(n)\}$.) Let $g(n, x)$ be the function that checks that x codes a computation of $\{f\}(n)$, along with a computation of $\{\{f\}(n)\}(i)$ for some i with output y. If any of those checks fail, $g(n, x)$ freezes (by, for example, calling h on itself: $h(\langle g, n, x\rangle, z)$, where z is some dummy variable). If they all pass, then $g(n, x)$ outputs y. Notice that those checks are p.r., so that g is feedback p.r. (sequential even). Let $\{e\}^h(n)$ be a call to $h(g(n, \cdot))$. Then e is as desired. ■

So, in contrast to the parallel Turing machines, there is really nothing new gained by parallelizing primitive recursion.

VI. Future Directions

We consider this work to be just a first (or second, or third) exploration into feedback for oracle computability. There are other possible applications than those considered here.

A. Iterated Feedback

In a way, feedback, by providing an oracle for divergence and convergence, replaces that distinction with one between freezing and non-freezing. It takes little imagination to ask what would happen with a feedback-style oracle that also answered freezing questions. That is, the feedback computation sketched here is a notion of computability that allows for oracles and maintains a distinction between freezing and non-freezing computations. Hence the considerations above apply, and allow for a discussion of oracles that say whether a computation freezes or not. We call this hyper-feedback. So: what can hyper-feedback Turing machines compute?

One possible answer is already afforded by the analysis of the μ-calculus from [9]. Namely, are the hyper-feedback definable sets exactly those definable in the μ-calculus over the natural numbers? The reason to think so is that the base computations for hyper-feedback, by our work here, are the hyperarithmetic sets, or, depending on just how you set the problem up, those sets definable from $\mathcal{O}$. It is well known that this corresponds to least points of positive inductive operators in arithmetic, which is the first step in the μ-calculus, one application of the least-fixed-point operator. What gives the μ-calculus its enormous strength is the feedback built into the language. So both hyper-feedback and the μ-calculus are extensions of least fixed points by feedback. Hence one might think they produce the same sets.

On the other hand, the μ-calculus provides a canonical ω-sequence through its limit, namely via those sets you get by restricting the number of alternations between least and greatest fixed point to a finite number. There seems to be no such ω-sequence in hyper-feedback. So the ultimate comparison between these two models is of independent interest.

B. Feedback ITTMs

As discussed in the introduction, feedback machines were first introduced in [10], perhaps surprisingly not for the most common kind of machine, Turing machines, but rather for infinite time Turing machines. To this day, the reals so computable have yet to be characterized. It is easy to see they are the reals in an initial segment of L, but its ordinal height has yet to be described. What makes this especially intriguing, beyond for its own sake, is that it seems to be related to determinacy. That is, since [10] it has been shown that the closure point of FITTM computations is at most the least ordinal γ such that L_γ models Σ^0_3-Determinacy [11]. We conjecture that there is actual equality here. What makes this plausible is the connection on both sides with Σ_2 reflection. Welch [14], [15] characterized the closure point of regular ITTMs via L_ζ, the least initial segment of L with a Σ_2 elementary extension L_Σ. So you would expect FITTMs to have a closure point at some sort of super-Σ_2 reflecting ordinal. In other work [16], he characterizes the least model of Σ^0_3-Determinacy via just such a kind of strong Σ_2 reflection. Hence the presumed connection between FITTMs and Σ^0_3-Determinacy.

C. Feedback for Other Computabilities

It could go without saying that feedback applies to any notion of computation that allows for oracles. If you consider the feedback version of any known theory of computability, the result is a notion of computability that is either already known, or not. In the former case, it is interesting because it's a new connection between already established theories of computation. An example of this is the main result here, that feedback Turing machines are exactly the hyperarithmetic algorithms. The latter case is interesting, because you then have a newly identified kind of computation to explore. An example of this is from the previous sub-section, feedback ITTMs. We would like to see what happens with other examples of feedback.

Some that we are currently considering are other sub-recursive collections, like p.r., that are sufficiently robust. We believe we can show, using only certain closure properties of the class, that their feedback versions are exactly the Turing computable functions. This might answer the question, why we considered the primitive recursive functions in particular. The answer would then be, no solid reason, just a convenient choice among the many options.

D. Alternative Semantics

All of the semantics we have considered so far have been the most conservative possible: the interpretations we took were always least fixed points, allowing something in only when necessary. Of course, one gets a perfectly coherent semantics by taking any fixed point. Is there anything to be gained by studying these alternative interpretations?

Once one hears least fixed point, one naturally thinks of greatest fixed point, and so could naively think that there is a second natural semantics. It's actually more complicated than that, though. The considerations here are not just about a set of non-freezing computations, but also the determination of whether such a computation is convergent or divergent. If one were to follow the gfp dictum "start with everything and whittle it down until you have a fixed point," we would have to start with a set containing both assertions "e converges" and "e diverges." Such an oracle we would call **inconsistent**. Among the consistent oracles, there is no natural largest one, since any function from ω to $\{\uparrow, \downarrow\}$ is maximal; so there is no unique greatest fixed point. One could of course consider inconsistent oracles, thereby allowing for a gfp, but we are unsure how they should be interpreted. If for instance an oracle says that e both converges and diverges, if e is called several times during a computation, would a legal run insist that the oracle give the same answer every time? So our questions are, how should inconsistent oracles be interpreted, and what, if anything, are they useful for?

E. Parallelism

It would not take a lot to realize that the section on parallelism is incomplete. Even at its very opening, many different options for its interpretation were presented, but only one was used. What do the others give you? More importantly, it was shown that parallel Turing feedback gets more than sequential Turing feedback, but we just haven't gotten around yet to thinking about exactly what the former does compute. So what does it? In particular, how does it compare with iterated feedback? Another question comes from considering the other case discussed, that parallel feedback primitive recursion does not get any more than sequential feedback primitive recursion. Is there some general way, applicable to many cases, to distinguish those versions of feedback for which the parallel variant is stronger than the sequential from those versions for which it is not?

F. Kolmogorov Complexity

One of the properties of Martin-Löf randomness which makes it stand out among the other notions, such as Schnorr randomness or Kurtz randomness, is that the same notion is obtained through several natural, but very different, definitions. Two of the most significant such definitions are via Martin-Löf tests and via Kolmogorov complexity.

As with many concepts in classical computability theory there is an analogue of a Martin-Löf test in higher computability theory which is obtained by simply replacing "computable" in the definition with "Δ^1_1" and "computably enumerable" with "Π^1_1". The resulting notion of a Martin-Löf test is called a Π^1_1-Martin-Löf test, and the corresponding notion of a Martin-Löf random real is called a Π^1_1-Martin-Löf random real.

Given that there is a natural meta-computable analogue of a Martin-Löf random real, it is natural to ask if there is also a meta-computable analogue of a Kolmogorov random real. Here, though, we run into a small problem.

The difficulty of generalizing Kolmogorov randomness arises in defining Kolmogorov complexity for finite strings. In particular, to have this notion make sense, we need a notion of "machine" which takes in finite elements and outputs finite elements. One solution to this problem was proposed in [4], which introduced the notion of a Π^1_1-machine and showed that the analogous notion of Kolmogorov randomness agrees with the notion of passing Π^1_1 Martin-Löf tests. Feedback machines provide another natural notion of a finite machine that performs a meta-computation. This leads to the following definition.

Definition 46: Fix a universal (parallel) feedback machine U. The (parallel) feedback complexity of a finite string X (relative to U), denoted $K_F(X)$ (or $K_{PF}(X)$), is the length of the shortest input Y such that $U(Y) = X$. A real $r \in 2^\omega$ is said to be (parallel) feedback Kolmogorov random if there is a constant c such that $(\forall n) K_F(r|n) \geq n - c$ (or $(\forall n) K_{PF}(r|n) \geq n - c$).

Because feedback computation captures the Π^1_1-sets, we expect that the notion of a feedback Kolmogorov random real should coincide with that of a Π^1_1-Martin-Löf random real. Furthermore, just as parallel feedback machines can characterize sets which ordinary feedback machines can't, we expect the collection of parallel feedback Kolmogorov random reals to be strictly contained in the collection of feedback Kolmogorov random reals. It is then an interesting problem to characterize this notion of randomness.

References

[1] A. Arnold and D. Niwinski, "Rudiments of μ-Calculus," **Studies in Logic and the Foundations of Mathematics**, v. 146, North Holland, 2001

[2] Jon Barwise, "Admissibile Sets and Structures," **Perspectives in Mathematical Logic**, Springer-Verlag, Berlin 1975

[3] Joel Hamkins and Andy Lewis, "Infinite time Turing machines," **The Journal of Symbolic Logic**, v. 65 (2000), pp. 567–604

[4] Greg Hjorth and André Nies, "Randomness via effective descriptive set theory," **Journal of the London Mathematical Society**, Second Series, v. 75 (2007), pp. 495–508

[5] Stephen Cole Kleene, "Recursive functionals and quantifiers of finite types. I," **Transactions of the American Mathematical Society**, v. 91 (1959), pp. 1–53

[6] Peter Koepke, "Turing computations on ordinals," **The Bulletin of Symbolic Logic**, v. 11 (2005), pp. 377–397

[7] Peter Koepke, "Infinite time register machines," in **Logical Approaches to Computational Barriers** (Arnold Beckmann et al., eds.), Lecture Notes in Computer Science 3988 (2006), pp. 257–266

[8] Peter Koepke and Benjamin Seyfferth, "Ordinal machines and admissible recursion theory," **Annals of Pure and Applied Logic**, v. 160 (2009), pp. 310–318

[9] Robert Lubarsky, "μ-definable sets of integers," **The Journal of Symbolic Logic**, v. 58 (1993), pp. 291–313

[10] Robert Lubarsky, "ITTMs with feedback", in **Ways of Proof Theory** (Ralf Schindler, ed.), Ontos, 2010, pp. 341–354

[11] Robert Lubarsky, "Feedback ITTMs and Σ^0_3-determinacy," slides, available at http://math.fau.edu/lubarsky/pubs.html

[12] Gerald Sacks, "Higher Recursion Theory", **Perspectives in Mathematical Logic**, Springer-Verlag, Berlin 1990, pp. xvi+344

[13] Dirk van Dalen, **Logic and Structure**, Springer, 2008

[14] Philip Welch, "Eventually infinite time Turing machine degrees: infinite time decidable reals," **The Journal of Symbolic Logic**, v. 65 (2000), pp. 1193–1203

[15] Philip Welch, "Characteristics of discrete transfinite Turing machine models: halting times, stabilization times, and normal form theorems," **Theoretical Computer Science**, v. 410 (2009), pp. 426–442

[16] Philip Welch, "Weak systems of determinacy and arithmetical quasi-inductive definitions," **The Journal of Symbolic Logic**, v. 76 (2011), pp. 418–436