An Introduction to Feedback Turing Computability

Nathanael L. Ackerman

Dept. of Mathematics, Harvard University, Cambridge, MA 02138, USA

nate@math.harvard.edu

Cameron E. Freer

Computer Science and AI Lab, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

freer@mit.edu

Robert S. Lubarsky

Dept. of Mathematical Sciences, Florida Atlantic University, Boca Raton, FL 33431, USA

Robert.Lubarsky@alum.mit.edu

Abstract

Feedback computability is computation with an oracle that contains the correct convergence/divergence information for all computations calling that same oracle. Here we study feedback Turing computability, as well as feedback for some smaller classes of computation. We also examine some versions of parallelization of these notions.

Keywords: computability theory, Turing machines, hyperarithmetic computability, least fixed points, parallel computation, feedback, determinism, non-determinism, reflection, gap-reflection, admissibility
2010 MSC: 03D10, 03D30, 03D60, 03D70, 03E10, 03E75

1. Introduction

Suppose you have some notion of computation that allows for oracle calls. What would be the most canonical choice of oracle? Over the years, we have learned that would be the halting problem, the set of indices of convergent computations. With that choice of oracle, note that the computations in the oracle are different from those calling the oracle: the computations that are being run can query the oracle, and the ones in the oracle cannot. What if they were the same? What if the oracle contained all the correct con- and divergence information about computations that call that same oracle?

This is feedback. One can consider the procedure which takes as input an oracle, and returns the set of halting computations relative to that oracle. A

model for feedback is exactly a fixed point of this procedure. As a topic, then, feedback can be understood as centering on the study of such fixed points.

The first time feedback was pursued seriously seems to have been by the third author in [8]; perhaps it is odd that the computability to which feedback was applied there is a somewhat exotic form of computation, infinite time Turing machines (first introduced in [4]). The feedback version of the most common notion of computability, Turing computability, was done only afterwards, by the current authors in [1], where feedback primitive recursion was analyzed also. It was during the preparation of that work that we discovered that the idea of feedback had already been articulated clearly by Rogers ([13], pp. 406-407), even if not under that name. Those few pages also anticipate some of the results of [1], albeit without proof. Oddly enough, even though that was likely the leading recursion theory text for decades, no one ever picked up on those ideas, not even to supply the missing proofs of Rogers' assertions.

Since then, progress has been made with feedback in the study of parallelism. While parallel feedback ITTMs were identified in [8], nothing much was done with them there. In contrast, several versions of parallel feedback Turing computability were identified in [1], and one was shown to be strictly stronger than its sequential analogue. Further progress was made in [10], where one kind of parallelism was completely analyzed, among other results.

The feedback fixed points studied so far have all been least fixed points. Philip Welch has observed that their study could well be couched differently, in more traditional terms. For instance, the least feedback fixed point for Turing computability is Kleene's $\mathcal{O}$, and in fact that analysis ends up looking like one of Kleene's own [6], recursion relative to the jump operator J (both being based on the well-foundedness, or not, of various trees). Similarly, the third author had conjectured that feedback ITTMs have the exact computational strength of Σ^0_3 Determinacy [9], and this was ultimately proven by Welch [20], couched not in terms of feedback ITTMs, but rather via ITTM computation relative to the ITTM-jump operator. All that notwithstanding, we feel that feedback stands on its own. For one, even when the analyses end up being similar, the concept of feedback is a different idea from that of recursion in J. If that argument leaves you cold, because your philosophy is that the only ultimate justification of a new idea is new results, then we have something for you too. The parallelism discussed in [10] and here, although given by a least fixed point semantics, has apparently not been studied elsewhere. Also, there are various natural non-fixed point semantics currently under investigation.

The current paper is a review and extension of [1] and [10]. Following this introduction, it starts with the connections between feedback Turing computability and hyperarithmeticity. Then feedback is applied to sub-recursive classes, such as primitive recursion. Finally, feedback parallelism is discussed.

2. Feedback Turing Machines

2.1. Feedback Turing Machines

We would like to consider feedback as applied not just to regular Turing machines but also to their relativization to an arbitrary oracle. So 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 2.1. *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 [2, 7, 14].) Let h_X be its least fixed point, with domain H_X. These are as desired. □

Definition 2.2. *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.

2.2. 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 sub-machine 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:

$$\{n\mid \sigma^\frown n\in T\}\leq\omega. \tag{$*$}$$

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 2.3. *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 [14] or [3].)

Proposition 2.4. *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 2.4 is the best possible, by the following two propositions.

Proposition 2.5. *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 2.6. *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., [14]), 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 2.7. *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.*

2.3. 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 2.8. *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 [14].

Definition 2.9. *Suppose $X, Y : \omega \to 2$. Then X is* **hyperarithmetically reducible** *to Y, written $X \leq_H Y$, when $X \in A(Y)$.*

Definition 2.10. *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 2.11. *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 2.12. *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 2.13. *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 2.14. *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 [14] II.5.6).

Lemma 2.15. *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 2.1. *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 2.4 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 2.15 we know for some $m \in \mathcal{O}^Y$, $X \leq_T Y^{[m]_Y}$. By Lemma 2.13, $Y^{[m]_Y} \leq_F Y$. Hence $X \leq_F Y$. □

We then have the following as an easy corollary of Theorem 2.1.

Corollary 2.2. *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 $A(Y)$).*

3. Freezing Jumps and Feedback Semicomputable Sets

3.1. 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 3.1. *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 3.2. *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 [8].

Lemma 3.3. *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 $a \in \mathcal{O}^X$,*

$$\langle b \rangle^X(e,n,a) \simeq \langle e \rangle^X(n) \Leftrightarrow ht(T^X(e,n)) \leq [a]_X,$$

and

- *for $a \in \mathcal{O}^X$,*

$$\langle b \rangle^X(e, n, a) = \ddagger \Leftrightarrow ht(T^X(e, n)) > [a]_X,$$

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. $\square$

Theorem 3.1. *For any $X : \omega \to 2$, we have $X^{(f)} \equiv_T \mathcal{O}^X$.*

Proof: It follows directly from Lemma 2.13 that $\mathcal{O}^X \leq_T X^{(f)}$.

To show that $X^{(f)} \leq_F \mathcal{O}^X$, notice that by Lemma 3.3 and Proposition 2.4,

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

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

3.2. Feedback Semicomputability

Definition 3.4. *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 3.5. *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 2.3 and Proposition 2.4, $b \in B$ 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 3.6. *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 3.7. *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. □

4. 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 [19].) This naturally leads to a nested semantics, as is most easily seen in the definition of Kleene's T predicate ([19], 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 4.1. *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. [19], 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 4.2. *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 [2, 7, 14].) □

Definition 4.3. *The* **feedback primitive recursive functions** *are the h-p.r. functions, with the h from the preceding theorem.*

Theorem 4.4. *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 4.5. *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 4.6. *The* **convergence feedback primitive recursive functions** *are the h-p.r. functions, with the h from the preceding theorem.*

Lemma 4.7. *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 4.8. *Every convergence feedback p.r. functions is a sub-function of a primitive recursive function.*

Proof: Letting $\mathbf{0}$ 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 $\mathbf{0}$ is p.r. □

Theorem 4.9. *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

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

□

5. Feedback on Other Sub-Recursive Classes

The choice of primitive recursion in the previous section was really just based on the convenience of having a particular, concrete notion of computation to work with that was also well-known. Here we show that a similar result is obtained when starting from any of a number of families of computable functions.

Definition 5.1. *An* **indexed collection** *is a pair* (e, E) *such that the following hold.*

- $E \subseteq \omega$*, which can be thought of as codes for Turing machines in the collection.*
- $e \in E$ *is such that* $(\forall n)\{e\}(n)\downarrow$ *and* $E = \{n : \{e\}(n) = 0\}$*. Hence* e *is a code for a Turing machine which picks out elements of* E*.*

In other words, an indexed collection is a computable collection of codes for Turing machines along with a witness to its computability which is itself one of the Turing machines in the collection.

The constructions below make use of indexed collections where all elements compute total functions, moreover with respect to any oracle. We use two oracles, the first of which allows for an arbitrary relativization, and the second is taking the role of a potential feedback oracle.

Definition 5.2. *An indexed collection* (e, E) *is* **absolutely halting** *if for all* $f \in E$*,*

$$(\forall X, Y : \omega \to 2)\,(\forall n \in \omega)\,\{f\}^{X,Y}(n)\downarrow .$$

So an indexed collection is absolutely halting if every element of E, when considered as code for a Turing machine with two oracle tapes, always halts no matter the oracle or the input. In particular, a standard diagonalization argument shows that there can be no universal function for an absolutely halting indexed collection.

An easy example of such a collection is given by programming languages that are restricted to be total but not necessarily Turing complete.

Example 5.1. *As is well known (e.g., by considering the programming language* `LOOP` *[15]), the primitive recursive functions are exactly those which can be computed with bounded loops (i.e., loops whose length is specified in advance). Let* E *be the collection of natural numbers that code programs which only use these operations. It is easy to see that there is a primitive recursive program* e *which checks its input to see if it represents code for such a program. Hence* (e, E)*, is an absolutely halting indexed collection.*

Another example of such collections comes when the resources of computation are restricted, but in a way that is not so severe that a computation cannot check to confirm when a code satisfies the restrictions.

Example 5.2. *Suppose X is any class of computable functions from ω to ω satisfying the following.*

- *X is closed under composition, pairing, and projections.*
- *There is a subset $E_X \subseteq \omega$ such that for all $\alpha \in X$ there is an $f_\alpha \in E_X$ such that $\{f_\alpha\}$ computes α.*
- *The characteristic function of E_X is in X.*
- *If $\alpha \in X$, then for all $a, b \in \omega$ the map $n \to \alpha(n) + an + b$ is in X as well.*

Then the collection of functions which run in time at most X is an absolutely halting indexed collection. In particular, the collection of machines which run in linear time, polynomial time, or exponential time are all absolutely halting indexed collections.

Given an absolutely halting family, and any fixed oracle X, one can define the least fixed point for Y, and the trees of sub-computations, exactly as above for the full collection of Turing computations. Intuitively, an (e, E)-feedback computation freezes if it makes a feedback call on code, which makes a feedback call on code, etc., in a way that never terminates. In this situation, there ceases to be a unique way to determine the value of the computation.

The one difference is that instead of the feedback oracle returning just $\downarrow$ and $\uparrow$, it is supposed to return the value of the called computation, just as in the previous section for primitive recursion. In the context of feedback Turing machines, there is no computational difference between simply being told that code for a feedback machine halts and being given the halting output, since the Turing machine could, when the code it queried halts, simply simulate that code. However, in the context of (e, E)-feedback machines, not only is there no assumption of a universal element in the indexed collection (which would make simulation of code possible), but also there is an assumption that all code is absolutely halting, i.e., will halt no matter what answers are given to the feedback queries (so long as they don't freeze). Hence in this case, simply asking halting queries provides no new information (as the answer will always be "yes"), and further, there may be no way to simulate the query to discover the output at halting. Because of this, it is important in this implementation to not just answer whether the queried code halts, but also to give the resulting output.

Example 5.3. *Let f be code that does the following in order.*

- *Ask a feedback query on $(f, 0)$.*
- *If the feedback query returns 0, then return 1. If the feedback query returns 1, then return 0.*

Note that, when considered as a Turing machine, the code f halts on all oracles. However, for any indexed collection (e, E) with $f \in E$, the code f must freeze as an (e, E)-feedback machine.

The following is then easy to check by induction.

Lemma 5.3. *Suppose e is absolutely halting and f, g are well-founded computational trees for (e, n). Then*

- *$f = g$, and*
- *f is finitely branching.*

This then motivates the following definition.

Definition 5.4. *The feedback machine e is said to* **converge** *on input n, written $\langle e \rangle(n)\Downarrow$, if there is a well-founded computation tree for (e, n). If the output of computational tree is m, then one writes $\langle e \rangle^X(n) = m$. Otherwise, e is said to freeze on n, written $\langle e \rangle^X(n)\Uparrow$.*

We are now in a position to prove our goal theorem.

Theorem 5.5. *Suppose (e, E) is an absolutely halting indexed collection containing the linear-time functions. Then the functions computable from a feedback machine from this collection are exactly the Turing computable functions.*

Proof: For the left-to-right direction, we will define a Turing machine m such that, for all $f \in E$,

(a) $\{m\}^X(f, n)\downarrow$ if and only if $\langle f \rangle^X(n)\Downarrow$, and

(b) if $\{m\}^X(f, n)\downarrow$, then $\{m\}^X(f, n) = \langle f \rangle^X(n)$.

Define m recursively, as follows. First, have m mimic $\langle f \rangle^X(n)$ until a feedback query is made, say for (g, r). Then call $\{m\}^X(g, r)$ and continue to mimic $\langle f \rangle^X(n)$ with the call to the feedback query returning $\{m\}^X(g, r)$. When $\{m\}^X(f, n)$ mimics $\langle f \rangle^X(n)$ halting, then return the value it halted at.

Part (a): First note that if $\{m\}^X(f, n)$ converges, then $\{m\}^X(f, n)$ traces out a computational tree for (f, n) and so $\langle f \rangle^X(n)\Downarrow$. But by Lemma 5.3, (f, n) has a finite computational witness if and only if all of its computational witnesses are finite, if and only if $\langle f \rangle^X(n)\Downarrow$. Therefore, as $\{m\}^X(f, n)$ is tracing out the computational witness, $\{m\}^X(f, n)$ halts whenever $\langle f \rangle^X(n)\Downarrow$.

Part (b): This is immediate from the definition of the output of a computational.

(Note the use made here of the fact that f is absolutely halting, i.e., that no matter the result of oracle or feedback queries, the computations of f always halts (because $f \in E$). Without this hypothesis, one cannot guarantee that the computational witnesses are finitely branching, and hence the Turing machine which mimics it might run forever despite it not freezing.)

In particular, this implies that every total function which can be computed by an (e, E)-feedback machine is in fact computable, that every set which is the

domain of a partial (e, E)-feedback machine is computably enumerable, and that every set which is the range of a total (e, E)-feedback machine is computably enumerable.

In the other direction, for every Turing machine $\{f\}^X$ we will describe a feedback machine f^+ such that

(a) $\{f\}^X(n)\downarrow \Leftrightarrow \langle f^+\rangle^X(\langle n, \langle\rangle\rangle)\Downarrow$, and

(b) if $\{f\}^X(n)\downarrow$ then $\{f\}^X(n) = \langle f^+\rangle^X(\langle n, \langle\rangle\rangle)$.

Let f^+ work as follows. It decomposes its input into a pair $\langle n, x\rangle$. It views x as the state information of a Turing machine. If x is in a halting state, f^+ returns the contents of the output tape. Else f^+ computes one step of the computation of f with input n and state x, to produce a new state x^+. Then f^+ makes a feedback oracle call on $\langle f^+, \langle n, x^+\rangle\rangle$, and returns whatever answer it gets from the oracle.

It is then immediate that f^+ has the desired properties. □

6. Parallelism

A variant of feedback, as identified in [8], 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. Indeed, most of the notions of computability studied by mathematical logicians have universal machines, which may well explain why they have never, as far as we know, studied parallelism, in contrast with complexity theorists and computer scientists, where parallelism is of great importance (for instance in P vs. NP or in quantum or distributed computing). With feedback, the trick of simulating parallelism by sequential computation does not work. 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 [8], it was left open there whether or not they yield more than sequential FITTMs. That unhappy fact notwithstanding, we identify below several versions of parallelism and determine whether in fact they do yield more than their sequential counterparts, in most cases analyzing their computational strengths completely. Since there is this variety in the kinds of parallelism, including some that we do not

study here, it is not yet clear which are most fruitful. By analyzing some of them, we hope to bring this issue along.

So, what should be the oracle's response to the question whether, for some n, $\langle e\rangle^X(n)$ does not freeze? One possibility is a straightforward yes-or-no answer. We do not want to go there just now: as a simple example of this, one could simulate non-parallel oracle calls, by asking the oracle about $\langle e\rangle^X(n)$ where in the program $\langle e\rangle$ the input n is never examined, and thereby answer the question whether a feedback computation freezes or not. This is of interest, but would take us too far afield for now. A related option is just to answer "yes" when that is the truth and to freeze otherwise. This is useless, since one could not thereby compute anything new: if a computation does not freeze, then one knows without needing to make an oracle call that all of the oracle calls would have to answer "yes" (lest the computation freeze).

Hence we will have our parallel machines return at least some witnessing input n to this non-freezing, some n such that $\langle e\rangle^X(n)$ does not freeze. Which n? The most natural choice 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. This is the topic of the next section, where it is shown that it provides no increase in computational power over sequential feedback.

In the section after that, the option considered is to punt on the determinism. Allow the oracle to return non-deterministically any n such that $\langle e\rangle^X(n)$ does not freeze. There is a choice to be made here. 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, leaving the other for future investigation.

Since our efforts with non-deterministic parallelism end up being only partially successful, we then return to determinism, by distinguishing a canonical choice of n, albeit slyer than the one above.

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.

In the following, we recycle notation, by using $\langle e\rangle(n)$ ambiguously to refer to any notion of feedback Turing computation, whether sequential or parallel, of any of the stripes listed above, the choice of which we hope is clear from the context.

7. Absolutely Deterministic Parallelism

The idea here is that the oracle is supposed to return the "least" n leading to non-freezing, by some measure. The measure to be used is primarily that of ordinal height of a computation. That is, n minimizes the height of the tree of sub-computations.

The tree $D_\alpha^{(e,n)}$ (D for determinism) is defined inductively on α, simultaneously for all e, n, as is whether $\mathrm{rank}(e,n) = \alpha$. Assume this is known for all $\beta < \alpha$. Start the run of $\langle e\rangle(n)$, which is considered as taking place at the root of $D_\alpha^{(e,n)}$. Suppose at some stage of that computation, an oracle call e' is made. Then a child of the root is established, to the right of any previous children, for the outcome of this oracle call. Suppose there is an n' such that $\mathrm{rank}(e',n') < \alpha$. Then let n' be chosen to minimize this rank; if there is more than one such, then among those pick the least in the natural ordering of ω. The tree $D^{(e',n')} = D^{(e',n')}_{\mathrm{rank}(e',n')}$ is placed at the child, and the value $\langle e'\rangle(n')$ is returned to the main computation, which then continues. If there is no such n', then the computation pauses, and the construction of $D_\alpha^{(e,n)}$ is finished.

If no oracle calls pause, then by this stage α the computation $\langle e\rangle(n)$ is seen to be non-freezing; $D^{(e,n)}$ can be taken to be $D_\alpha^{(e,n)}$ and is the tree of sub-computations; $\mathrm{rank}(e,n) \leq \alpha$; and the value of $\langle e\rangle(n)$ is the content on the output tape if the main computation ever entered into a halting state, else $\uparrow$ if it did not.

It is not hard to show that the rank of a computation is the ordinal height of its tree of sub-computations. For a freezing computation, i.e. one that remains paused however big α is taken to be, we do not (yet) have a good notion of a tree of sub-computations. For the eternally paused node, which is trying to run, say, e' in parallel, it's paused because for each n' the trees $D_\beta^{(e',n')}$ remain paused, say at $e''_{n'}$. This could be viewed as countable branching from e', but of course this branching is different from that in $D^{(e,n)}$: in the latter tree, the branching shows the sequential computation, and the unsuccessful parallel runs are suppressed; from e', the branching represents all the parallel attempts. Of course, from $e''_{n'}$, the same story continues.

The problem with this notion is that it doesn't get us anything new.

Theorem 7.1. *If $\langle e\rangle(n)$ does not freeze, then $D^{(e,n)} \in L_{\omega_1^{CK}}$.*

Proof: By induction on the ordinal height of $D^{(e,n)}$. Consider the subtrees $D^{(e',n')}$ that occur on the top level (i.e. children of the root) of $D^{(e,n)}$. Induc-

tively, they are all in $L_{\omega_1^{CK}}$. If there are only finitely many of them, then the ordinal α by which they all appear is easily less that ω_1^{CK}, since the latter is a limit ordinal. $D^{(e,n)}$ is then easily definable over L_α. If there are infinitely many, then the admissibility of ω_1^{CK} must be used to get α to be strictly less than ω_1^{CK}. The set of such $D^{(e',n')}$'s is the range of a Σ_1 definable function f with domain ω, since the run of $\langle e\rangle(n)$ is simply defined, and (mod the oracle calls) continues for ω-many steps; $f(k)$ is then the sub-tree $D^{(e',n')}$ for the k^{th} oracle call. □

8. Non-deterministic Parallelism

Since choosing one canonical output to a parallel call didn't work out so well, let's go to other extreme and allow all possible answers. As with sequential feedback machines, we will formalize this in two ultimately equivalent ways, first as a positive inductive operator, of which we take the least fixed point, and then via a canonical tree, albeit one different from the tree of sub-computations.

8.1. Semantics

In the following definition, think of $H(e,n)$ as our current belief of the set of possible outputs for $\langle e\rangle_H^X(n)$.

Definition 8.1. *Given $X:\omega\to 2$ and $H:\omega\times\omega\to\mathcal{P}(\omega\cup\{\uparrow\})$, a* **legal run** *of $\langle e\rangle_H^X(n)$ is (some standard encoding of) a run of a Turing machine calculation of $\{e\}_H^X(n)$, 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, the set of legal outputs is absolute. To see this, we will need to view the computation via an associated tree, the **tree of runs**. The tree of runs is not to be confused with the tree of sub-computations, so central in developing feedback. The tree of sub-computations summarized the sequential running of an algorithm, which can be viewed as traversing that tree, depth-first, from left to right. In contrast, the tree of runs captures the non-determinism. The splitting at a node is the many parallel runs of an oracle call. A single run of the algorithm is a path through the tree. There is no room in this tree for the sub-computations: if a node in the tree of runs represents $\langle e\rangle(n)=k$, the witness to that last computation is not contained in the tree, but rather must be found in the tree of runs for $\langle e\rangle(n)$.

The tree of sub-computations is hidden in the step from a node with end $\hat{e}$ to its children, or to its lack of children, which can be determined only by building $\hat{e}$'s own tree of runs.

Definition 8.2. *The* **tree of runs** *is built from the root (thought of as being on the top) downwards, or, equivalently, as the computation proceeds, starting from the beginning, step 0. Each node has a start, meant to be the state of the computation when that node becomes active, and an end, meant as the state of the computation when the node becomes inactive. The start of the root is the program (e, n) being run. What the end of the root, or any other node for that matter, is, depends. If continuing the computation from the start of the node leads to an oracle call, say $\hat{e}$, then the end of the node is this $\hat{e}$; as need be, we may assume that the state of the computation at that point is also recorded in the node. If no such oracle call exists, then there are two possibilities. One is that after finitely many steps from the start of the node the computation has entered into a halting state. Then the end of the node is this halting state, and the content of the output tape is an output of the main computation. The other possibility is that the computation from the node's start never enters into a halting state, and so it diverges. Then the end of the node is this divergence, symbolically $\uparrow$, which is an output of the main computation.*

Nodes that end in a halting state or with divergence have no children. A node that ends with $\hat{e}$ may have children. For any natural number $\hat{n}$, and any $k \in H(\hat{e}, \hat{n})$, there is a child with start $(\hat{e}, \hat{n}, k)$, and which continues the computation of its parent with that start as the answer to the oracle call. Implicitly, and now explicitly, if there are no such $\hat{n}$ and k, then that node has no children, and the computation freezes there.

The **tree outputs** *consist of the following: for each terminal node in the tree of runs ending in a halting state, the contents of the output tape; for each terminal node ending in $\uparrow$, $\uparrow$ itself; and if the tree of runs is ill-founded, $\uparrow$.*

Lemma 8.3. *The legal outputs of $\langle e \rangle_H^X(n)$ are the same as the tree outputs, and are absolute among all standard models of ZF.*

Proof: A legal run is exactly a maximal path through the tree of runs, which is itself absolute. 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 either a finite path ending in $\uparrow$ or 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 8.4. *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 $\langle e \rangle_H^X(n)$.*

Lemma 8.5. *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 8.6. *$\langle e\rangle^X(n)$ refers to $\langle e\rangle^X_H(n)$, with H as from the lemma above.*

Because the semantics is given by a least fixed point, ordinal heights can be associated with these computations (when non-freezing). Ultimately, we will define the height of an output. But we must be careful here: because of the nondeterminism, there could be wildly different ways to arrive at the same output. The simple solution to that would be to define the height of an output as the least ordinal among all the ordinals given by the different ways to get to that output. To do this right, one must define the height of a run of a computation, or, actually, the height of a *hereditary run.*

A **hereditary run** of a non-freezing computation is a run of that computation, along with an assignment, to each oracle call in the run (i.e. node in the run with end $\hat{e}$), with answer $(\hat{e}, \hat{n}, k)$ (i.e. the child in this run of that aforementioned node has start $(\hat{e}, \hat{n}, k)$), a hereditary run of $(\hat{e}, \hat{n})$ with output k.

The **height of a hereditary run** is defined inductively as the least ordinal greater than the heights of all of the sub-runs, meaning the hereditary runs assigned to oracle calls along the way.

The **height of a computation** $\langle e\rangle^X(n) = k$ is the smallest height of any hereditary run of such a computation. We will want to show that this is absolute among all transitive models.

Define $T^{(e,n)}_\alpha$, the sub-tree of the tree of runs of (e, n) which contains only those children of rank less than α, inductively on α.

For $\alpha = 0$, this tree contains only the root; if $\langle e\rangle^X(n)$ makes an oracle call then $T^{(e,n)}_0$ does not witness any output, else it witnesses either some finite k or $\uparrow$ as an output.

More generally, if $\beta < \alpha$, then $T^{(e,n)}_\beta \subseteq T^{(e,n)}_\alpha$. Furthermore, if a node in $T^{(e,n)}_\alpha$ ends with an oracle call $\hat{e}$, and there are $\beta < \alpha, \hat{n}$, and k (including $\uparrow$) such that $T^{(\hat{e},\hat{n})}_\beta$ witnesses that k is an output, then the child with start $(\hat{e}, \hat{n}, k)$ is in $T^{(e,n)}_\alpha$.

The outputs witnessed by $T^{(e,n)}_\alpha$ are the outputs of any terminal node (i.e. k if a node ends in a halting state with output k, or $\uparrow$ if a node ends with $\uparrow$), and also $\uparrow$ if $T^{(e,n)}_\alpha$ is ill-founded.

Notice that the height of $\langle e\rangle^X(n) = k$ is at most α iff $T^{(e,n)}_\alpha$ witnesses k as an output.

Proposition 8.7. *The height of $\langle e\rangle(n) = k$ is absolute among all transitive models.*

Proof: Inductively on α, the trees $T_\alpha^{(e,n)}$ and the outputs they witness are absolute. The outputs witnessed by terminal nodes are clearly absolute, individual nodes being finite, and for divergence, well-foundedness is absolute for well-founded models. □

8.2. Functions and Ordinal Notations

Ultimately we would like to characterize just what is parallel feedback computable. In the context of multi-valued functions, what this means should be clarified.

Definition 8.8. *A function f is* **parallel feedback computable (pfc)** *(from X) if there is an index e such that $\langle e\rangle^X(\cdot)$ is single valued and $\langle e\rangle^X(n) = f(n)$. A set is pfc if its characteristic function is. Notice that in both cases $\langle e\rangle^X(n)$ could still have freezing legal runs.*

We would like to know what functions are pfc, and what relations are pfc.

While it should be no surprise that functions offer some benefits over relations, let's bring out a particular way that happens. Consider the index e which on any n returns both 0 and 1. (In more detail, let p be the parity function: $\{p\}(n)$ is 0 when n is even, 1 when odd. Let $\langle e\rangle^X(n)$ make a parallel call to p and return its output.) Notice that the characteristic function of any set at all is given by some run of e. So if you're non-deterministically searching for, say, the truth set of some L_α, there may well be a pfc function that gives you what you want, but you can't distinguish that from this e. And it does you no good to pick one non-deterministically, because if you pick e, when you go to use it again later, you might get different answers.

Since we expect that the analysis of this will involve computing initial segments of L, we might have need of notation for ordinals, which can be defined à la Kleene's $\mathcal{O}$. In honor of this history, and since the current subject is parallelism, we will call it $\mathcal{P}$. Because of the non-determinism present, there are several options for how this can be defined (in the limit case).

Definition 8.9. *Functional $\mathcal{P}$ ($f\mathcal{P}$) is defined inductively:*

- *$0 \in f\mathcal{P}$ and ord(0) = 0.*
- *If $a \in f\mathcal{P}$ then $2^a \in f\mathcal{P}$ and ord(2^a) = ord(a) + 1.*
- *If $\langle a\rangle(\cdot)$ is a function, and for all n we have $\langle a\rangle(n) \in f\mathcal{P}$, then $3\cdot 5^a \in f\mathcal{P}$ and ord($3\cdot 5^a$) = $sup_n\{\mathrm{ord}\langle a\rangle(n)\}$.*

Definition 8.10. *Strict $\mathcal{P}$ ($s\mathcal{P}$) is defined inductively:*

- *$0 \in s\mathcal{P}$ and ord(0) = 0.*

- *If $a \in s\mathcal{P}$ then $2^a \in s\mathcal{P}$ and ord(2^a) = ord(a) + 1.*
- *If $\langle a \rangle(\cdot)$ is a total relation, and for all n and any possible output k_n of $\langle a \rangle(n)$ we have $k_n \in s\mathcal{P}$, and moreover ord(k_n) is independent of the choice of k_n (for a fixed n), then $3 \cdot 5^a \in s\mathcal{P}$ and ord($3 \cdot 5^a$) = $sup_n\{\mathrm{ord}(k_n)\}$, where k_n is any output for $\langle a \rangle(n)$.*

Definition 8.11. *Loose $\mathcal{P}$ ($l\mathcal{P}$) is defined inductively:*

- *$0 \in l\mathcal{P}$ and ord(0) = 0.*
- *If $a \in l\mathcal{P}$ then $2^a \in l\mathcal{P}$ and ord(2^a) = ord(a) + 1.*
- *If $\langle a \rangle(\cdot)$ is a total relation, and for all n and any possible output k_n of $\langle a \rangle(n)$ we have $k_n \in l\mathcal{P}$, and $sup_n\{\mathrm{ord}(k_n)\}$ is independent of the choice of k_n's, then $3 \cdot 5^a \in l\mathcal{P}$ and ord($3 \cdot 5^a$) = $sup_n\{\mathrm{ord}(k_n)\}$, where k_n is any output for $\langle a \rangle(n)$.*

Clearly, $f\mathcal{P} \subseteq s\mathcal{P} \subseteq l\mathcal{P}$.

Proposition 8.12. *Every pfc well-ordering is isomorphic to one given by a functional ordinal notation.*

Proposition 8.13. *If Y is pfc then $\mathcal{O}^Y$ is pfc and ω_1^Y has a functional ordinal notation.*

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 $n \in \mathcal{O}^Y$.

To handle the case of those $n \notin \mathcal{O}^Y$, 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 $\mathcal{O}^Y$. When considering 0, g freezes, because 0 is in $\mathcal{O}^Y$. When considering 2^e, g moves on to e. When considering $3 \cdot 5^e$, first g checks whether $\{e\}^Y$ 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 $n \in \mathcal{O}^Y$, will always freeze, because the machine will eventually consider 0. When started on some $n \notin \mathcal{O}^Y$, 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 $n \notin \mathcal{O}^Y$, we know $n \neq 0$; if $n = 2^e$ then $e \notin \mathcal{O}^Y$; else if n is not of the form $3 \cdot 5^e$ then the computation halts; else there is some k such that $\{e\}^Y(k) \notin \mathcal{O}^Y$, so the computation can continue. Hence there is a legal output (either 0 or $\uparrow$).

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

From a computation of $\mathcal{O}^Y$, it is easy to get an ordinal notation for ω_1^Y. $\square$

Proposition 8.14. *If α has a loose ordinal notation then the Σ_1 truth set Tr_α of $L_{\omega_\alpha^{CK}}$ is pfc (where, as a function of α, ω_α^{CK} enumerates the closure of the set of admissible ordinals).*

Proof: Let $e \in \mathcal{P}$ be a fixed representation of α. By the recursion theorem, we can do this inductively on the ordinal height of $f <_{\mathcal{P}} e$.

If $f = 0$, then $Tr_f = \emptyset$.

If $f = 2^g$, then $Tr_f = \mathcal{O}^{Tr_g}$ from the previous proposition. (It is standard hyperarithmetic theory that $\mathcal{O}^X$ is Turing equivalent to the Σ_1 truth predicate of $L_{\omega_1^{CK}}$.)

If $f = 3 \cdot 5^g$, then the truth or falsity of any Σ_1 assertion ϕ in the limit structure can be determined as follows. Let n run through ω, and see whether ϕ is true according to each $Tr_{g(n)}$ in turn. If you ever find such an n making ϕ true, halt, else continue. Using feedback, ask whether that computation halts. If so, then ϕ is true in the limit structure; else ϕ is false there. □

Because of that last proposition, it seems that the loose notations are ultimately the best, since they seem to capture the flavor of this kind of computation.

Proposition 8.15. *The characteristic function of $T_\alpha^{(e,n)}$ (along with the start and end of each node) is computable from a loose ordinal notation for α, as are the outputs witnessed by $T_\alpha^{(e,n)}$.*

With a bit of work, this could be presented as a corollary of the previous proposition, since $T_\alpha^{(e,n)}$ and its outputs are definable over $L_{\omega_\alpha^{CK}}$.

Proof: By a simultaneous induction on ordinal notations.

The only notation for the ordinal 0 is 0. To compute $T_0^{(e,n)}$, one first asks the oracle whether computing $\langle e \rangle(n)$ will ever lead to an oracle call. If so, one runs $\langle e \rangle(n)$ until that call, which becomes the end of the root, and then stops. If not, one asks the oracle whether computing $\langle e \rangle(n)$ will ever halt. If so, one runs it until it halts; if not, then the output is $\uparrow$.

Consider the ordinal notation $a = 2^b$ for $\alpha = \beta + 1$. Of course, the root of $T_\alpha^{(e,n)}$ is computable, as above. For any node in $T_\alpha^{(e,n)}$, to see whether a child is in $T_\alpha^{(e,n)}$, we may assume the node ends with $\hat{e}$. A child starting with $(\hat{e}, \hat{n}, k)$ is in $T_\alpha^{(e,n)}$ iff $T_\beta^{(\hat{e},\hat{n})}$ witnesses that k is an output, which inductively is computable from b. The end of such a node is deterministic in the start. To compute whether k is witnessed to be an output, one can use the oracle to see whether the search through $T_\alpha^{(e,n)}$ for a terminal node with output k will halt. In addition, when $k = \uparrow$, check whether $T_\alpha^{(e,n)}$ is well-founded, which is computable in its hyperjump (cf. the penultimate proposition).

Now consider the ordinal notation $a = 3 \cdot 5^b$. We must decide membership in $T_\alpha^{(e,n)}$ of children of nodes ending in $\hat{e}$. For the child starting with $(\hat{e}, \hat{n}, k)$, use the oracle to see whether the search for an i such that, with $\beta_i = \text{ord}(b(i))$, the

tree $T_{\beta_i}^{(\hat{e},\hat{n})}$ witnesses that k is an output, halts. The determination of $b(i)$ is, of course, non-deterministic, as is the value β_i, but as β_i is guaranteed to be cofinal in α, this makes no difference. The computation of the outputs witnessed is as above. □

The hope is that the structure just identified will help in determining the pfc functions and relations, which we have not been able to do. Although the next section is dedicated to the study of a different kind of computation for its own sake, it also provides at least a coarse upper bound for those studied here.

9. Context-Dependent Determinism

In this section, we omit reference to the oracle X. It is easy to see that everything relativizes.

9.1. Semantics

The problem of the first alternative offered is that it's too restrictive, and so gives you nothing new. The problem with the second is that it's too liberal, allowing for multi-valuedness, and so we couldn't analyze it. This time we're going for something in the middle. Any oracle call will return at most one value, but possibly a different value every time it's called.

The semantics begins just as in the non-deterministic case. Trees $C_\alpha^{(e,n)}$ (C for context) are defined inductively on α. The new intuition here is that these trees are built until an output is seen, and that first output is taken as the value of $\langle e\rangle(n)$. More precisely, $C_\alpha^{(e,n)}$ yields an output if it contains a halting node (with some integer output k) or a diverging node (with output $\uparrow$), or is ill-founded (with output $\uparrow$). Let α be the least ordinal such that $C_\alpha^{(e,n)}$ yields an output. If it yields more than one output, pick the left-most one. That is, starting at the root, traverse the tree downwards. Every non-terminal node ends with an oracle call $\hat{e}$. The child to be followed has start $(\hat{e}, \hat{n}, k)$, where $\hat{n}$ is the least natural number such that the tree beneath that node yields an outcome (and k is the value of $\langle\hat{e}\rangle(\hat{n})$).

As an example of this semantics in practice, the proof above that $\mathcal{O}^Y$ is pfc from Y still works. The way that construction goes, given a non-well-founded order, and an n in the non-standard part, if k is in the standard part, it won't be chosen as a successor step after n, because that will definitely lead to a freezing state. Only a non-standard k (less than n in this ordering) will be chosen, and in fact that least such k in the natural ordering of ω will be.

9.2. Lemmas

Lemma 9.1. *There is a program which, on input e, diverges if e computes (the characteristic function of) the truth set of a model of some computable theory T, and freezes otherwise.*

We assume here some standard coding of syntax into arithmetic. The model can be taken to be a structure on, say, the odd integers, so that the even integers can be used for the symbols of the language, and formulas with parameters can be considered. Of course, this program can easily be converted into one that halts instead of diverges: ask the oracle about this program, and if the answer comes back "divergent," then halt. It will be easy to see that in some instances it can be recognized that e does not compute such a set, and our program could return that instead of diverging; but if, say, $\langle e \rangle(0)$ freezes, then any such program as ours would have to freeze, and there seemed to be no benefit in a program that sometimes recognizes when e is not as desired and sometimes freezes.

Proof: It is feedback Turing computable to dovetail the generation of T, the computations of $\langle e \rangle(n)$ for all n, and the check that that latter theory is complete, consistent, and contains T. If e computes such a model, this procedure will never end; if e finds some violation, the procedure can be taken to freeze. If some $\langle e \rangle(n)$ freezes, the procedure will necessarily freeze. □

We will be using this to see if e codes a model of $V = L_\alpha$. We do not sharply distinguish between the Σ_1 truth set of some L_α and the full truth set, since this computational paradigm can easily shuttle between them.

Lemma 9.2. *There is a program such that, if e computes a partial order on a subset of ω, on input e it will return 0 if e's order is well-founded and 1 if ill-founded.*

Proof: This is a lot like the proof of the computability of $\mathcal{O}$.

For pre-processing, check whether the domain of e is finite. If so, you have your answer. Else, continue.

First we check for well-foundedness. Go through the natural numbers, and for each such n, if n has no e-predecessors (determined by an oracle call), halt, else run this same procedure, via the fixed-point or recursion theorem, on the same order restricted to those elements e-less than n. In the tree of subcomputations, the children of a node given by n are exactly the e-predecessors of n. So this tree is well-founded iff $<_e$ is well-founded. So this procedure diverges iff $<_e$ is well-founded, else it freezes.

To check for ill-foundedness, run in parallel the following procedure on each $n \in \omega$. If n has no predecessor, freeze. Else, by the fixed point theorem, run this same procedure on the same order restricted to those elements e-less than n. In the tree of runs, the children of a node given by n are exactly the e-predecessors of n. So this tree is well-founded iff $<_e$ is well-founded. Since the terminal nodes all freeze, the only possible non-freezing semantics is an infinite descending path, which exists exactly when $<_e$ is ill-founded.

Now run both of those checks in parallel. Whichever one does not freeze is what tells you whether $<_e$ is well- or ill-founded. □

As usual, it is easy to see that what can be computed is exactly some initial segment of L. We will shortly see just what this initial segment is. Before that, we will prove some lemmas which handle some simpler cases, partly to get the reader (and author!) used to the kind of arguments employed, and partly so in the main theorem we can ignore some of the cases of weaker, messier ordinals, and focus on just the more strongly closed ones.

Lemma 9.3. *The supremum α of the computable ordinals is admissible.*

Proof: Suppose not. Let $f : \omega \to \alpha$ witness α's inadmissibility. For each n, using the previous lemmas, one can check whether $\langle e \rangle$ codes a model of "$V = L_\gamma$ is the least admissible set in which $f(n)$ is defined," and if so whether the model so coded is well-founded. On many inputs this will freeze, but since by hypothesis α is the least non-computable ordinal, there is at least one e_n on which this halts (possibly more, allowing for some flexibility in the coding). By making a parallel call of all natural numbers, one can produce such an e_n.

To see whether a Σ_1 formulas ϕ is in the Σ_1 truth set for L_α, consider the procedure which runs through each n, finds a truth set for $f(n)$ as above, and stops whenever ϕ shows up as true in one of those sets. Now ask the oracle whether that procedure halts. If so ϕ is true in L_α, else not. □

Lemma 9.4. *α is greater than the least recursively inaccessible.*

Proof: The following procedure will generate the Σ_1 truth set of the first recursively inaccessible.

Start with (a code for) the truth set of $L_{\omega_1^{CK}}$. We will describe a procedure which pieces larger and larger initial segments of L together, which diverges (continues indefinitely) as long as it's still working on the first inaccessible, and which freezes whenever it finds a contradiction in what it has done so far.

At any stage along the way, there will be a well-founded model of $V = L_\gamma$, as well as a finite set of Π_1 sentences the procedure is committed to making true. As soon as the model at hand falsifies one of those sentences, then the procedure freezes, because it sees that the jig is up.

Dovetail consideration of all countably many Σ_1 formulas $\phi(x, y, \vec{z})$ and all countably many sets A and tuples $\vec{b}$ that show up in the models produced in this construction. At stage n we are considering a certain ϕ, A, and $\vec{b}$, and will decide whether we think $\forall a \in A\ \exists y\ \phi(a, y, \vec{b})$ is true or false in the first recursively inaccessible. In parallel, choose either true or false. Moreover, if you choose true, then you must provide a well-founded model of $V = L_\gamma$ extending the previously chosen model by at least one admissible, in which the chosen formula with parameters is true, and which also models there is no recursively inaccessible. If you choose false, then you must also choose a specific $a \in A$, and include in the set of sentences "$\forall y\ \neg\phi(a, y, \vec{b})$".

Since this construction has no halting condition, the only way it can not freeze is if it diverges. It cannot diverge by always making the chosen formula false, if for no other reason than there are infinitely many total Σ_1 functions in the starting model, and they cannot consistently be made partial. So infinitely often the model under consideration will be extended by at least one admissible. Hence the limit model will be an initial segment of L which is a limit of admissibles. Let ϕ be Σ_1 and $A, \vec{b}$ be in the limit model. Suppose it's true in this model that $\forall a \in A\ \exists y\ \phi(a, y, \vec{b})$. When that formula came under consideration, it could not have been deemed false, because then we would have committed ourselves to a specific counter-example, and that counter-example would have been seen to be invalid at some point, leading to a freezing computation. So the formula was deemed to be true. Hence a model was picked in which the induced relation was total, thereby providing a bound on the range. Hence the limit model is admissible. Since it's a limit of models of "there is no recursively inaccessible," it is itself the least recursively inaccessible.

We have just argued that any divergent run of this program produces the least recursively inaccessible. Furthermore, there are divergent runs, by always choosing whatever is in fact true of that ordinal. □

9.3. Main Theorems

Definition 9.5. *Let Γ be a collection of formulas, X a class of ordinals, and ν^{+X} the least member of X greater than ν. We say that α is* **Γ-reflecting on** *X if, for all $\phi \in \Gamma$, if $L_{\alpha^{+X}} \models \phi(\alpha)$, then for some $\beta < \alpha$, $L_{\beta^{+X}} \models \phi(\beta)$.*

We are interested in the case $\Gamma = \Pi_1$ and $X =$ the collection of admissible ordinals. For this choice of X, we abbreviate ν^{+X} by ν^+, which is standard notation for the next admissible anyway. This is called Π_1 **gap-reflection on admissibles**. Let γ be the least such ordinal.

It may seem like a strange notion. But this is not the first time it has come up. Extending work in [12], it was shown in [7] that such ordinals are exactly the Σ^1_1 reflecting ordinals. (In this context, the superscript 1 refers not to reals but to subsets of the structure over which the formula is being evaluated.) The reason this topic came up in the latter paper is that a particular case of its main theorem is that γ is the closure point of Σ_2-definable sets of integers in the μ-calculus. (The μ-calculus is first-order logic augmented with least and greatest fixed-point operators. In this context, Σ_2 refers to the complexity of the fixed points in the formula, namely, in normal form, a least fixed point in front, followed by a greatest fixed point, followed by a fixed-point-free matrix.) In [12] it was also shown that the least Σ^1_1 reflecting ordinal is also the closure point of Σ^1_1 monotone inductive definitions. (Here the superscript does refer to reals.) Furthermore, that is the same least ordinal over which winning strategies for all Σ^0_2 games are definable (Solovay, see [11] 7C.10 or [18]). As though that weren't enough, [**?**] shows the equivalence of closure under Σ^1_1 monotone inductive definitions with the Σ^1_1 Ramsey property. (For all Σ^1_1 partitions P of

ω there is an infinite set $H \subseteq \omega$ such that the infinite subsets of H are either all in P or all not in P.) With all of these applications, this definition counts as natural.

Theorem 9.6. *The ordinals so computable are exactly those less than γ.*

So there is an intimate connection between parallel feedback computability and all of the other notions listed above. This was not expected. In the simpler case of feedback Turing computability [1], it was really no surprise that it turned out to be the same as hyperarithmeticity, as both are essentially adjoining well-foundedness to computation. But we have no intuition, even after the fact, in support of the current result.

Proof: For one direction, we will argue that no computation $\langle e\rangle(n)$ can be witnessed to converge or diverge from stage γ onwards. Notice that for any $\gamma' > \gamma$, if $T^{(e,n)}_{\gamma'}$ is different from $T^{(e,n)}_{\gamma}$, that can only be because some other computation $\langle e'\rangle(n')$ was seen to converge or diverge at some stage at least γ and less than γ'. Tracing back the computation of $\langle e'\rangle(n')$, we are eventually led to a computation that was seen to converge or diverge at exactly stage γ. Since γ is a limit of admissibles, there are no new terminal nodes on any tree of runs at stage γ. Hence there is some computation $\langle e\rangle(n)$ such that $T^{(e,n)}_{\gamma}$ is ill-founded, but $T^{(e,n)}_{\beta}$ is well-founded for any $\beta < \gamma$. How could the ill-foundedness of $T^{(e,n)}_{\gamma}$ be most economically expressed? Since γ is the γ^{th} admissible ordinal, $T^{(e,n)}_{\gamma}$ is definable over L_γ. It is a basic result of admissibility theory that a tree in an admissible set is well-founded iff there is a rank function from the tree to the ordinals in that very same admissible set. So the ill-foundedness of such a tree is witnessed by the non-existence of such a function in any admissible set containing the tree. In the case at hand, that is a Π_1 statement in L_{γ^+} with parameter γ. By the choice of γ, this reflects down to some smaller β. So $T^{(e,n)}_{\beta}$, for some smaller β, was already seen to be ill-founded. So there can be no new computation values at stage γ, and hence not beyond either.

For the converse, let β be strictly less than γ; by lemmas 9.2 and 9.3, we can assume that β is a limit of admissibles. Assume inductively that for each $\alpha < \beta$ there is an e such that $\langle e\rangle(\cdot)$ is the characteristic function of the Σ_1 truth set of L_α. Let ϕ witness that β is not Π_1 gap-reflecting on admissibles: so ϕ is Π_1, and $L_{\beta^+} \models \phi(\beta)$, but if $\alpha < \beta$ then $L_{\alpha^+} \not\models \phi(\alpha)$. We must show that (the characteristic function of) the Σ_1 truth set of L_β is computable.

As in lemma 9.4, start with (a code for the Σ_1 truth set of) $L_{\omega_1^{CK}}$. At any stage along the way, there will be a well-founded model of $V = L_\alpha$, as well as two finite sets (both empty at the beginning) of sentences. The intent of this construction is that, if it continues for ω-many steps, the union of the L_α's so chosen will be L_β, all of the sentences in the first set will be true in L_β, and the second set will provide a term model of $V = L_{\beta^+}$.

The action at any stage is much as in the previous lemma. First, check for the consistency of a theory, to be described below. If an inconsistency is found, freeze. Else we are going to continue building the ultimate model. This

involves interleaving steps to make sure that the union of the chosen L_α's, L_δ, is admissible (and $\delta \leq \beta$), with steps to insure that $L_{\delta^+} \models \phi(\delta)$ (guaranteeing $\delta \geq \beta$). We assume a dovetailing, fixed at the beginning, of all (countably many) formulas ψ with parameters. For the formulas in the first set, the parameters are the sets in the L_α's chosen along the way. For the formulas in the second set, the parameters include, in addition to the members of the L_α's, also constants c_i for the term model, as well as a dedicated constant we will ambiguously call δ, since the ordinal δ is its intended interpretation.

At any even stage $2n$, consider the n^{th} formula of the form $\forall a \in A\ \exists y\ \psi(a, y, \vec{b})$, where ψ is Σ_1 and the parameters are from the L_α at hand. In parallel, choose it to be either true or false. Moreover, you must provide a well-founded model of $V = L_\alpha$, extending the previously chosen model by at least one admissible. Furthermore, if you had deemed the formula to be true, then it must hold in the chosen L_α; if false, then you must also choose a specific $a \in A$, and include in the first set of sentences "$\forall y\ \neg\psi(a, y, \vec{b})$". Notice that this step includes as a degenerate case those instances in which ψ does not depend on a, thereby forcing us to decide all Σ_1 and Π_1 formulas. Finally, it must be the case that $\alpha < \beta$, which can be verified computably, since it needs only a well founded model of $V = L_{\alpha^+}$ (which exists by the inductive hypothesis and the choice of β) which also satisfies "$\neg\phi(\alpha) \wedge \forall \nu < \alpha\ L_{\nu^+} \not\models \phi(\nu)$".

At an odd stage $2n + 1$, consider similar to the above the n^{th} formula of the form $\forall a \in A\ \exists y\ \psi(a, y, \vec{b})$, where ψ is Σ_1, only this time the parameters are for the second set (that means the parameters are from an already chosen L_α and the c_i's and δ). Include in the second set either "$\forall a \in A\ \exists y \in \tau\ \psi(a, y, \vec{b})$", for some term τ, or "$\tau \in A \wedge \forall y\ \neg\psi(\tau, y, \vec{b}),$" for some term τ. Of course, this step is meant to include all possible degenerate cases, such as Σ_1 assertions, even quantifier-free sentences. Also, if "$\tau < \delta$" for some term τ is ever included in the second set, then, extending L_α if need be, for some $\epsilon < \alpha$ the sentence "$\tau = \epsilon$" is included in the second set.

With regard to the theory referenced above but there left unspecified, at any stage along the way it will be "$V = L_{\delta^+}$ is admissible, and δ is admissible, and $\alpha < \delta$ (where L_α is the model we have at this stage), and everything in the first set is true in L_δ, and everything in the second set is true in V."

For this computation, the tree of runs has neither halting nor divergence nodes (since, whenever it does not freeze, it makes another oracle call). It is ill-founded, since there is a run of the computation which does not halt, namely one using the truth about L_β and L_{β^+} to make decisions along the way. We would like to show that along any infinite path in the tree of runs, the induced δ equals β.

Consider the term model induced by the second set. There is an isomorphism between the term δ and the union of the α's chosen along the way: on the one hand, the assertion "$\alpha < \delta$" was included in the theory along the way, and on the other, anything ever deemed less than δ was forced to be less than some α. So we can consider the term model as including some (standard) ordinal δ. Also, this δ is at most β, since each α is less than β. The next observation is

that this term model satisfies "$V = L_{\delta^+}$ is admissible," by the Henkinization (choice of explicit witnesses) performed on the second set. Of course, the term model might well be ill-founded. But its well-founded part has ordinal height the real δ^+. By the downward persistence of Π_1 sentences, since $\phi(\delta)$ holds in the term model, it holds in the actual L_{δ^+}. By the choice of ϕ, δ is at least as big as β.

We must turn this procedure into a way of getting the characteristic function for the truth set of L_β. For any Σ_1 sentence χ, run the procedure as above, with χ and $\neg\chi$ each separately, in parallel, included in the first set. The false option is inconsistent and so any such computation will freeze, so the answer you will get is the true option, along with the information that the procedure diverges. □

Corollary 9.7. *For $\beta < \gamma$, the order-types of the $\Sigma_1(L_\beta)$-definable well-orderings of ω are the ordinals less than β^+.*

This is a generalization of the earlier result that the order-types of the Π^1_1 well-orderings are cofinal in ω_2^{CK}. Sacks [14], giving this special case as an exercise (p. 51, 7.10), attributes it to Richard Platek, who never published a proof. Although Platek may have been the first to notice this (Sacks in personal correspondence dates it from the '60s), Tanaka [16] seems to have discovered it independently.

The corollary as stated is not the optimal result, since the conclusion holds for any β which is Σ_1 projectible, by arguments similar to Tanaka's. It's just that this more general result is no longer a corollary to the theorem.

Proof: For simplicity, assume that β is a limit of admissibles. The construction of the theorem is of an ill-founded tree T_β, Σ_1 definable over L_β, such that any infinite path yields a term model of $V = L$ with ordinal standard part β^+. If the well-founded nodes all had rank less than some $\beta' < \beta^+$, then they could all be distinguished from the non-well-founded nodes definably over $L_{\beta'}$. So an infinite path, and hence such a term model, is also definable over $L_{\beta'}$. It is then easy (which we can here take to mean "definable over $L_{\beta'}$") to read off all the reals in this model. This includes reals with L-rank cofinal in β^+. This is a contradiction. Hence, for any $\beta' < \beta$, there is a node in T_β with that rank. The nodes of T_β are labeled with pairs (e, n). They also have associated with them two finite sets of formulas. The formulas are just finite pieces of syntax, except for the parameters from L_β's. But L_β is the Σ_1 Skolem hull of ω, which provides an integer name for each of its members (for instance, a Σ_1 formula that it uniquely satisfies). So each formula can be coded by a natural number. All told, each node can be represented by a natural number. This produces an ordering of a subset of ω with rank β'. To get this to be a well-ordering, it suffices to take the Kleene-Brouwer ordering of that tree. □

Happily, the work done also enables us to determine at least an upper bound for the non-deterministic computations.

Theorem 9.8. *Any relation computable via a non-deterministic parallel feedback Turing machine, as in the previous section, is* $\Sigma_1(L_\gamma)$.

Proof: By much the same argument as before. The only possible values come from halting nodes, divergent nodes, and the ill-foundedness of trees. A node is seen to halt at a successor stage, and γ is not a successor ordinal. A node is seen to diverge at a stage of the form $\alpha + \omega$, and γ is not of that form. As for the last possibility, the tree $T_\gamma^{(e,n)}$ is Δ_1 definable in L_{γ^+} with parameter γ. If it's not well-founded, that fact is Π_1 expressible in L_{γ^+}. By the choice of γ, a smaller $T_\alpha^{(e,n)}$ was already ill-founded, so divergence was already a value for $\langle e \rangle(n)$. Hence there are no new possible values for any computation at or after stage γ. □

10. 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. If we allow for non-determinism, then there will be 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 10.1. *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 10.2. *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 10.3. *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 10.4. *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 non-deterministic parallel Turing machines, there is really nothing new gained by parallelizing primitive recursion.

11. Future Directions

We consider this work to be just an early exploration into feedback for oracle computability. There are other possible applications than those considered here.

11.1. 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 [7]. 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. Also, hyper-feedback yields the computability of the parallelism studied here, by dovetailing, and this parallelism gives the beginning of the μ-calculus hierarchy.

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.

11.2. 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 one of the main results 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 feedback ITTMs [9, 20]. We would like to see what happens with other examples of feedback.

11.3. 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?

11.4. Parallelism

The section above on parallelism is incomplete. Some forms of parallelism were mentioned by not analyzed. Non-deterministic parallelism was studied, but we were unable to determine just what it computes. Those items remain open. 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?

11.5. 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 [5], 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 11.1. *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] Nathanael Ackerman, Cameron Freer, and Robert Lubarsky, "Feedback Turing Computability, and Turing Computability as Feedback," **Proceedings of LICS 2015, Kyoto, Japan**; also available at http://math.fau.edu/lubarsky/pubs.html

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

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

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

[5] 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

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

[7] Robert Lubarsky, "μ-definable sets of integers," **The Journal of Symbolic Logic**, v. 58 (1) (1993), pp. 291–313

[8] Robert Lubarsky, "ITTMs with Feedback," in **Ways of Proof Theory** (Ralf Schindler, ed.), pp. 341-354. Ontos (2010); also available at http://math.fau.edu/lubarsky/pubs.html

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

[10] Robert Lubarsky, "Parallel Feedback Turing Computability," in **Proceedings of LFCS 2016, Lecture Notes in Computer Science 9537** (Sergei Artemov and Anil Nerode, eds.), pp. 236-250

[11] Yiannis Moschovakis, **Descriptive Set Theory**, First edition North Holland (1987); second edition AMS (2009)

[12] Wayne Richter and Peter Aczel, "Inductive Definitions and Reflecting Properties of Admissible Ordinals," in: **Generalized Recursion Theory** (Fenstad and Hinman, eds.), pp. 301-381. North-Holland (1974)

[13] Hartley Rogers **Theory of Recursive Functions and Effective Computability**, McGraw-Hill (1967)

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

[15] Uwe Schöning, **Theoretische Informatik Kurz Gefaßt**, Bibliographisches Institut, Mannheim, 1992

[16] Hisao Tanaka, "On Analytic Well-Orderings," **Journal of Symbolic Logic** 35 (2), pp. 198-204 (1970)

[17] Kazuyuki Tanaka, "The Galvin-Prikry Theorem and Set Existence Axioms," **Annals of Pure and Applied Logic** 42 (1), pp. 81-104 (1989)

[18] Kazuyuki Tanaka, "Weak Axioms of Determinacy and Subsystems of Analysis II (Σ^0_2 Games)," **Annals of Pure and Applied Logic** 52 (1-2), pp. 181-193 (1991)

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

[20] Philip Welch, "$G_{\delta\sigma}$-games and generalized computation," to appear