Tel Aviv University
The Raymond and Beverly Sackler
Faculty of Exact Sciences

Comparing Computational Power

Thesis submitted in partial fulfillment of the requirements for the M.Sc. degree in Tel-Aviv University, School of Computer Science

by

Udi Boker

Prepared under the supervision of Prof. Nachum Dershowitz

February, 2004

Acknowledgments

I deeply thank my supervisor, prof. Nachum Dershowitz, for his help, thought, patience, and most of all for his exceptional attitude, which makes the research an enlightening and enjoyable experience for me.

Abstract

It is standard to establish that one computational model, e.g. recursive functions, is strictly stronger—from the computational point of view—than another model, primitive recursive functions, for example, by showing that the functions computed by the second model are a proper subset of those computed by the first. Another standard method to show that a model, such as the untyped lambda calculus, is at least as powerful—computationally speaking—as another, the partial recursive functions, say, is to demonstrate that the former can simulate all partial functions of the latter under some suitable encoding of the domain.

The problem we raise in this thesis is the incompatibility of these two methods of comparing computational power. We provide examples of cases in which one of two models is at least as powerful as the other by the second method, but strictly weaker by the first. We also show that any alternative definition of 'suitable encoding', under which simulation is compatible with the first method of comparison, is too restrictive.

On the positive side, we prove that (1) Turing machines, the recursive functions, and the partial recursive functions are robust, in the sense that no encoding can make them equivalent to a hyper-recursive model, and that (2) the recursive functions are strictly more powerful than the primitive recursive functions, even according to the second method.

Contents

1 Introduction

1.1 History

The 1930s saw a blossoming of work on the nature of computation, including the development of several utterly different and unrelated models, each purported to be universal:

- *Partial recursive functions*, based on an inductive mechanism for the definition of functions. Proposed by Gödel and Kleene.
- *General recursive functions*, defined through an equational mechanism. Proposed by Gödel, Kleene, and Herbrand.
- *Lambda calculus*, based on a particularly constrained type of inductive definitions. Proposed by Church.
- *Turing machines*, based on a mechanistic model of problem solving by mathematicians. Proposed by Turing.

A few other universal models were suggested in the 1940s, 1950s, and 1960s:

- *Post systems*, based on deductive mechanisms. Proposed by Post in 1943.
- *Markov algorithm*, a model very similar to today's formal grammars. Proposed by Markov in 1954.
- *Universal register machines*, explicitly intended to reflect the structure of modern computers. Proposed by Shepherdson and Sturgis in 1963. Nowadays, many variants of that model have been devised and go by the generic name of *register-addressable machines*, or *random access machines (RAM)*

The remarkable result about these varied models is that all of them define exactly the same class of computable functions: whatever one model can compute, all the others can too! [8, pp. 6–7]

This equivalence among computational models, and the underlying formal notion of power comparison, is the main objective of the thesis.

1.2 Motivation

Our overall goal is to formalize the comparison of computational models, that is, to determine when one set of partial functions is computationally more powerful than another set. We seek a general definition of relative

power, one that does not depend on the notion of computability. It should allow one to compare arbitrary models over arbitrary domains in some quasi-ordering that captures the intuitive concept of computational strength. Such a formalization should also allow one to prove statements like "analogue machines are strictly more powerful than digital devices," even though the two models operate over domains of different cardinalities. We are only interested here in the extensional quality of a model, not complexity-based comparison or step-by-step simulation.

1.3 Standard Comparison Definitions

There are several methods by which models have been compared:

I. Normally, one would say that a model A is at least as powerful as B if for all programs q of B, there exists (not necessarily effectively) a program p of A such that p computes the same (partial) function as q. If A allows *more* functions than B, then it is standard to claim that A is *strictly* stronger. For example, primitive recursion (Prim) is weaker than general recursion (Rec) (e.g. [12, p. 92]), and inductive Turing machines are more powerful than Turing machines [1, p. 86].

II. The above definition does not work, however, when models use different data structures (representations). Instead, A is deemed at least as powerful as B if A can *simulate* every function computable by B. Specifically, the simulation is obtained by requiring an encoding ρ from the domain of B to that of A and a decoding τ in the reverse direction, such that for every function g computed by B $g = \tau \circ f \circ \rho$ for some function f computed by A, in which case A is said to be at least as powerful as B. Compare [4, p. 21]. Without additional restrictions, however, this definition allows for unreasonable instances: with $\rho(x) = \langle x, x\rangle$, $\tau(\langle x, g\rangle) = g(x)$, and $f'(\langle x, x\rangle) = \langle x, f\rangle$, for any f, the "dumb" model $\{f' : f \in M\}$ "simulates" *every* other model M.

Such problems are usually avoided by insisting that ρ be an injection and $\tau = \rho^{-1}$. See, for example, [9, p. 27], [3, p. 24], [6, p. 52], or [11, pp. 29–30]:

> Let W be some non empty set. A *coding* from $\mathbf{N}$ into W is an injection $c : \mathbf{N} \to W$. ... f incorporated in g relative to coding c, if $g(c(x)) = c(f(x))$ for all $x \in \mathbf{N}^n$. ... Computability relative to a coding is the basic concept in comparing the power of computation models.... The computational power

of the model is represented by the extension of the set of all functions computable according to the model. Thus, we can compare the power of computation models using the concept 'incorporation relative to some suitable coding'.

To show that two models are of equivalent power, one would then need to find two injections, each showing that every function computed by one can be simulated by the other. For example, Turing machines (TM), the untyped lambda calculus (Λ), and the partial recursive functions (PR) were all shown to be of equal computational power by Church [2], Kleene [7] and Turing [13].

III. When two models operate on different domains, it is common to fix an injective encoding between the domains (and to claim, subjectively, that the encoding is "natural") and show that one model is strictly stronger, since it simulates a proper superset of the other model. It is in this sense that one commonly asserts that analogue computation is strictly more powerful than digital (e.g. [10]).

1.4 Our Main Results

It turns out, however, that these various methods of comparing computational power are incompatible. In Section 3, we provide examples of cases in which a model A is strictly more powerful than B by the first method, whereas B is at least as powerful as A by the second, and one in which A is both stronger and weaker than B by Method III. On the other hand, we prove in Section 6 that the recursive functions are strictly more powerful than the primitive recursive functions, even according to Method II (Theorem 32).

In Section 5, we present the notion of a "robust" model, which is a model that is not equivalent to any proper superset of itself under any injective encoding, and provide several results regarding robustness. In particular, we show in Section 6 that both the recursive and partial recursive functions are robust, in the sense that no encoding can make them equivalent to a hyper-recursive model (Theorems 34 and 35). Accordingly, we provide a criterion with which to verify that a model M, operating over a denumerable domain, is hyper-recursive (Corollary 37).

In Section 7, we show that Turing machines are robust (Theorem 39) and isomorphic to the recursive functions (Corollary 40).

In Section 8, we extend the robustness results to inductive domains.

In Sections 9 and 10 we show that any alternative restriction on the simulation encoding of Method II, under which Method I and II are compatible, is too restrictive.

2 Definitions

We consider only deterministic computational models; hence, we deal with partial functions. To simplify the development, we will assume for now that the domain and range of functions are identical, except that the range is extended with $\perp$, representing "undefined."

Two partial functions (f and g) over the same domain (D) are deemed *(semantically* or *extensionally) equal* (and denoted simply $f = g$) if they are defined for exactly the same elements of the domain ($f(x) = \perp$ iff $g(x) = \perp$ for all $x \in D$) and have the same value whenever they are both defined ($f(x) = g(x)$ if $f(x) \neq \perp$, for all $x \in D$).

Definition 1 (Model of Computation). *Let D be an arbitrary domain. A* model of computation over D *is any set of functions $f : D \to D \cup \{\perp\}$.*

Since models are sets: When $A \subsetneq B$, for models A and B over the same domain, we say that A is a *submodel* of B and, likewise, that B is a *supermodel* of A.

Whenever we speak of $A \subseteq B$, we mean to also imply that the two models operate over the same domain.

We shall write dom A for the domain over which model A operates. To deal with models operating over different domains it is incumbent to map the domain of one model to that of the other. Let $\rho : \text{dom } B \to \text{dom } A$ be such an encoding. Then $\rho \circ M = \{\rho \circ f : f \in M\}$ and $M \circ \rho = \{f \circ \rho : f \in M\}$, for any relation ρ and set of functions M. Additionally, we insist that $\rho(y) = \perp$ iff $y = \perp$.

Definition 2 (Simulation). *Model A* simulates *model B* via *encoding ρ* : dom $B \to$ dom A*, denoted $A \succsim_\rho B$, if $\rho \circ B \subseteq A \circ \rho$.*

This is the notion of "implemented" used in [6] and of "incorporated" used in [11]. To quote:

> Given sets A, B a *coding* from A to B is a one-to-one total function $c : A \to B$. A partial function $g : B \to B_\perp$ *implements* partial function $f : A \to A_\perp$ relative to coding c if for all $a \in A$
>
> 1. $f(a) \neq \perp$ implies $g(c(a)) = c(f(a))$
> 2. $f(a) = \perp$ implies $g(c(a)) = \perp$
>
> [6, p. 52].

"Equivalently, f incorporated in g relative to coding c, if $g(c(x)) = c(f(x))$ for all $x \in \mathbf{N}^n$" [11, p. 29].

As a degenerate case, with the identity encoding ι ($\lambda n.n$), we have $A \succsim_\iota B$ iff $A \supseteq B$. Method I of comparison (see the introduction) uses this simple relation.

Method II is embodied in the following:

Definition 3 (Computational Power).

1. *Model A is* (computationally) at least as powerful *as model B, denoted $A \succsim B$, if there is an injection ρ such that $A \succsim_\rho B$.*
2. *Model A is* (computationally) more powerful *than B, denoted $A \succ B$, if $A \succsim B$ but $B \not\succsim A$.*
3. *Models A and B are* computationally equivalent *if $A \succsim B \succsim A$, in which case we write $A \sim B$.*

This equivalence is the same as in [11, p. 30]:

1. $F(SAL(X_1))$ is *incorporated* in $F(SAL(X_2))$ relative to coding c, if for every function f in $F(SAL(X_1))$ there is a function g in $F(SAL(X_2))$ which incorporates f relative to c.
2. The computation models $SAL(X_1)$ and $SAL(X_2)$ are *equivalent* if there exist codings c_1 and c_2 such that
   (a) $F(SAL(X_1))$ incorporates $F(SAL(X_2))$ via coding c_1
   (b) $F(SAL(X_2))$ incorporates $F(SAL(X_1))$ via coding c_2

where $F(SAL(X_i))$ denotes the set of functions computed by $SAL(X_i)$ [11, p. 27].

If $\rho \circ B \subsetneq A \circ \rho$, for some injection ρ, then Method III would assert that A is strictly more powerful. But be warned: This does *not* imply that $A \succ B$ by Method II!

Proposition 4. *The computational power relation $\succsim$ between models is a quasi-order. Computational equivalence $\sim$ is an equivalence relation.*

Proof. Transitivity of $\succsim$ is because the composition of injections is an injection, that is, $A \succsim_\rho B \succsim_\tau C$ implies that $A \succsim_{\rho\circ\tau} C$. Reflexivity is because the identity is an injection, that is, $A \succsim_\iota A$. □

Example 5. *Turing machines (*TM*) simulate the recursive functions (*Rec*) via a unary representation of the natural numbers. See, for example, [5, p. 147]:*

> *In order to use Turing machines to evaluate number-theoretic functions, it is necessary to choose a convention for representing natural numbers. We adopt the unary notation, in which the number n represented by a block of $(n+1)$ $1's$. A number-theoretic function is then said to be Turing computable if there exists a Turing machine that converts the unary representations of the arguments of the function into the unary representation of value of the function.*

And the corresponding theorem in [5, p. 240]: "Every μ-recursive function f is Turing computable."

Example 6. *The (untyped) λ-calculus (Λ) is equivalent to the partial recursive functions (*PR*) via Church numerals, on the one hand, and via Gödelization, on the other.*

Since the domain encoding ρ implies, by the simulation definition, a function mapping, we can extend ρ to functions and models, as follows:

Definition 7 (Function Mappings). *An encoding $\rho : \mathrm{dom}\ B \to \mathrm{dom}\ A$ induces the following mappings:*

$$\begin{aligned} \rho(g) &= \rho \circ g \circ \rho^{-1} \\ \rho\langle f \rangle &= \rho^{-1} \circ f \circ \rho \end{aligned}$$

from B to A and from A to B, respectively. These extend to sets of functions in the usual manner:

$$\begin{aligned} \rho(M) &= \{\rho(g) : g \in M\} \\ \rho\langle M \rangle &= \{\rho\langle f \rangle : f \in M\}\ . \end{aligned}$$

Note that any function f, such that $f \restriction_{\mathrm{rng}\,\rho} = \rho(g) \restriction_{\mathrm{rng}\,\rho}$, simulates g via ρ, while $\rho\langle f \rangle$ is the only function simulated by f. The model $\rho(M)$ is minimal (with respect to the restriction of the domain to rng ρ) among those that simulate M via ρ, and $\rho\langle M \rangle$ is the maximal model simulated by M (see Lemma 14 below).

Definition 8 (Strong Equivalence). *Model A is* strongly equivalent *to model B, denoted $A \simeq B$, if there are bijections π and τ such that $A \succsim_\pi B \succsim_\tau A$.*

Definition 9 (Isomorphism). *Model A is* isomorphic *to model B, denoted $A \equiv B$, if there is a bijection π such that $A \succsim_\pi B \succsim_{\pi^{-1}} A$.*

Example 10. *Lisp with only pure lists as data is isomorphic to the partial recursive functions via the Gödel pairing function:* $\pi(\mathbf{nil}) = 0$*;* $\pi(\mathbf{cons}(x, y)) = 2^{\pi(x)}(2\pi(y) + 1)$.

When π is recursive, one may speak of *recursively isomorphism*: function f is recursively isomorphic to g if there is a recursive permutation π, such that $f = \pi^{-1} \circ g \circ \pi$ [9, pp. 52–53]. Moreover: "A property of k-ary relations on $\mathbf{N}$ is *recursively invariant* if, whenever a relation R possesses the property, so does $g(R)$ for all $g \in \mathcal{G}^*$" [9, p. 52], where $\mathcal{G}^*$ are the recursive permutations of $\mathbf{N}$. Thus, one may claim: "[Recursion] theory essentially studies ... those properties of sets and functions which remain invariant under recursive permutations. For example, recursiveness, re-ness, m-completeness are such invariants" [12, p. 333].

In our notation, we would say that property P of model M is *recursively invariant* if $P(\pi(M))$ for all permutations $\pi \in \mathrm{Rec}$.

3 Equivalent Submodels

Unfortunately, the above standard definition of "simulates" (Method II) allows for the possibility that a model is equivalent to its supermodel.

Example 11. *The set of "even" recursive functions (R_2) is of equivalent power to the set of all recursive functions. Define:*

$$R_2 \quad = \quad \left\{ \lambda n. \left\{ \begin{array}{ll} 2f(n/2) & n \text{ is even} \\ n & \text{otherwise} \end{array} \right\} : f \in \mathrm{Rec} \right\}$$

We have that $R_2 \succsim_{\lambda n.2n} \mathrm{Rec} \succsim_{\iota} R_2$.

Furthermore, method III leads to situations where $A \succ B \succ A$ for models A, B.

Example 12. *Define the "odd" recursive functions (R_1):*

$$R_1 \quad = \quad \left\{ \lambda n. \left\{ \begin{array}{ll} 2f\lfloor n/2 \rfloor + 1 & n \text{ is odd} \\ n & \text{otherwise} \end{array} \right\} : f \in \mathrm{Rec} \right\}$$

We have that, $R_1 \succsim \mathrm{Rec} \supsetneq R_2 \succsim \mathrm{Rec} \supsetneq R_1$, *thus* $R_1 \succ R_2 \succ R_1$.

The mappings used in the above examples are the standard natural way of mapping the domains, thus method III of the introduction is ill-defined.

It turns out that the equivalence of a model and its supermodel is possible even when the encoding ρ is a bijection and the model is closed under functional composition.

Example 13. *Let K be a set of "basic functions" over* $\mathbf{N}$, *containing all the constant functions κ_k ($\lambda n.k$), plus the identity, ι. We present two models, A and B, that both contain the basic functions and are closed under functional composition, such that the smaller one (B) simulates every function of the infinitely larger one (A).*

Imagine the natural numbers arranged in a triangular array:

0	0							
1	1	2	3					
2	4	5	6	7	8			
3	9	10	11	12	13	14	15	
4	16	...						
⋮			⋱					
	0	**1**	**2**	**3**	**4**	**5**	**6**	...

Now, define the following computable functions:

$$\begin{aligned} f_{i,j}(n) &= \left(\lfloor\sqrt{n}\rfloor + i\right)^2 + j \bmod \left(2\lfloor\sqrt{n}\rfloor + 2i + 1\right) \\ g_i(n) &= f_{i,0}(n) = \left(\lfloor\sqrt{n}\rfloor + i\right)^2. \end{aligned}$$

If n is located on row m, then $f_{i,j}(n)$ is the number in row $n+i$ and column j, while $g_i(n)$ is the first number in row $n+i$.

Consider the following sets of functions:

$$\begin{aligned} F &= \{f_{i,j} : i, j > 0\} \\ G &= \{g_i : i > 0\}. \end{aligned}$$

Note that F and G are disjoint, since for every $i, j > 0$ and $n > j^2$, $f_{i-1,j}(n) < g_i(n) < f_{i,j}(n)$.

Define:

$$\begin{aligned} B &= K \cup F \\ A &= K \cup F \cup G. \end{aligned}$$

Thus, A has functions to jump anywhere in subsequent rows, while $B \subsetneq A$ is missing infinitely many functions g_i for getting to the first position of subsequent rows. Since, for $i + k > 0$,

$$f_{i,j} \circ f_{k,\ell} = f_{i+k,j}\ ,$$

it follows that both F and G are closed under composition, as is their union $F \cup G$, from which it follows that A and B are also closed.

There exists a (computable) permutation π of the naturals $\mathbf{N}$, such that $B \succsim_\pi A$:

$$\begin{aligned} \pi(n) &= f_{0,n-\lfloor\sqrt{n}\rfloor^2+1} \\ &= \lfloor\sqrt{n}\rfloor^2 + \left(n - \lfloor\sqrt{n}\rfloor^2 + 1\right) \bmod \left(2\lfloor\sqrt{n}\rfloor + 1\right), \end{aligned}$$

mapping numbers to their successor $n+1$, but wrapping around before each square. That is, π has the following cycles (of unbounded length):

$$\pi = \{(0),\ (1\,2\,3),\ (4\,5\,\ldots\,8),\ \ldots\}.$$

It remains to show that for all $f \in A = K \cup F \cup G$, we have $\pi(f) \in B = K \cup F$:

$$\pi(\iota) = \pi \circ \iota \circ \pi^{-1} = \iota \in K \subseteq B$$

$$\pi(\kappa_k) = \pi \circ \kappa_k \circ \pi^{-1} = \pi \circ \kappa_k = \kappa_{\pi(k)} \in K \subseteq B$$

$$\pi(f_{i,j}) = \pi \circ\ f_{i,j} \circ \pi^{-1} = \pi \circ\ f_{i,j} = f_{i,j+1} \in F \subseteq B,\ \textit{for } i > 0,\ j \geq 0$$

4 Basic Lemmas

One can categorize the maximal model that can be simulated, as follows:

Lemma 14. *For all models A and B, $A \succsim_\rho B$ iff $B \subseteq \rho\langle A\rangle$.*

Proof. We have $B \subseteq \rho\langle A\rangle$ iff for every $g \in B$ there is $f \in A$, such that $g = \rho^{-1} \circ f \circ \rho$. This is the same as requiring that for every $g \in B$ there is an $f \in A$, such that $\rho \circ g = \rho \circ \rho^{-1} \circ f \circ \rho = f \circ \rho$, that is, $A \succsim_\rho B$. □

Lemma 15. *For all models A and B, $A \succsim B$ implies that $|A| \geq |B|$. That is, if $A \succsim B$ for models A and B then there exists an injection from the set of functions of B to the set of functions of A.*

Proof. Follows directly from the computational power definition (Definition 3). □

Lemma 16. *For all models A and B and bijections π, $A \subsetneq B$ implies that $\pi(A) \subsetneq \pi(B)$ and $\pi\langle A\rangle \subsetneq \pi\langle B\rangle$.*

Proof. Since π is a bijection, it follows that $\pi \circ f \circ \pi^{-1} \neq \pi \circ g \circ \pi^{-1}$, for every $f \neq g$. Thus, $\pi(M)$ is an injection (i.e. every function of M is simulated by exactly one function via π). Therefore, $\pi(B \setminus A) \cap \pi(A) = \emptyset$, and $\pi(A) \subsetneq \pi(B)$. By the same argument, $\pi\langle A\rangle \subsetneq \pi\langle B\rangle$. □

Lemma 17. *For all models A and B and bijections π, $A \succsim_\pi B$ iff $A \supseteq \pi(B)$.*

Proof. Clearly, $\pi\langle A\rangle = \pi^{-1}(A)$, for all bijections π. Thus, by Lemma 14, $A \succsim_\pi B$ iff $B \subseteq \pi\langle A\rangle = \pi^{-1}(A)$. Therefore, by Lemma 16, $A \succsim_\pi B$ iff $\pi(B) \subseteq \pi(\pi^{-1}(A))$. Hence, $A \succsim_\pi B$ iff $\pi(B) \subseteq A$ □

Corollary 18. *For all models A and injections ρ, $A \succsim \rho\langle A\rangle$.*

Corollary 19. *For all models A and bijections π, $A \simeq \pi(A)$.*

Lemma 20. *If $A \simeq B \subsetneq C$, for models A, B, C, then there is a model $D \supsetneq A$, such that $C \simeq D$.*

Proof. Suppose $B \succsim_\pi A$ for bijection π. Thus, by Lemma 14, $A \subseteq \pi\langle B\rangle$. Let $D = \pi\langle C\rangle$, for which we have $C \simeq D$. Since $B \subsetneq C$, it follows, by Lemma 16, that $A \subseteq \pi\langle B\rangle \subsetneq \pi\langle C\rangle = D$. □

5 Stability and Robustness

As shown in Section 3, a model can be of equivalent power to its supermodel, or even strongly equivalent to it. There are, however, models that are not susceptible to such an anomaly.

Definition 21 (Stable). *A model is* stable *if it is not strongly equivalent to any of its supermodels. That is, A is stable if $A \simeq B \supseteq A$ implies $A = B$ for all B.*

Definition 22 (Robust). *A model is* robust *if it is not of equivalent power to any of its supermodels. That is, A is robust if $A \sim B \supseteq A$ implies $A = B$ for all B.*

Theorem 23.

1. *Robustness implies stability. That is, if a model A is robust then it is also stable.*

2. *Stability doesn't imply robustness. That is, there is a stable model A that is not robust.*

Proof. The first statement is trivial. For the second, define $A = \{\lambda n.k : k \in \mathbf{N}, k > 0\}$ (all constant functions over $\mathbf{N}$, except for $\lambda n.0$), and $K = \{\lambda n.k : k \in \mathbf{N}\}$ (all constant functions over $\mathbf{N}$). Since $A \succsim_{\lambda n.n+1} K$, it follows that A is not robust. Since $\pi(A) = \pi^{-1} \circ A \circ \pi = \pi^{-1} \circ A = \{\lambda n.k : k \in \{\mathrm{rng}\ \pi \setminus \pi(0)\}\}$, for every permutation π, it follows that $\pi(A) \neq K$, and includes only constant functions. Thus, $\pi(A) \not\supseteq A$, and therefore A is stable. □

Theorem 24.

1. *Isomorphism of models implies their strong equivalence.*

2. *Strong equivalence of stable models implies their isomorphism.*

Proof. The first statement is trivial. For the second, assume $A \succsim_\pi B \succsim_\tau A$ for bijections π, τ. If $\pi\langle A\rangle \subsetneq B$, then, by Lemma 16, $\tau\langle\pi\langle A\rangle\rangle \subsetneq A$, which contradicts the stability of A. Thus $B = \pi\langle A\rangle$, and therefore, $A = \pi^{-1}\langle B\rangle$. □

Lemma 25. *A stable model is not strongly equivalent to any of its submodels. That is, for a stable model A, $A \simeq B \subseteq A$ implies $A = B$ for all models B.*

Proof. Suppose that A is stable, and assume, on the contrary, that $A \simeq B \subsetneq A$. It follows that $A \subseteq \pi\langle B\rangle$ for some permutation π. Thus, by Lemma 16, $A \subseteq \pi\langle B\rangle \subsetneq \pi\langle A\rangle$. Therefore, $A \simeq \pi\langle A\rangle \supsetneq A$, which contradicts the stability of A. □

Note that the above lemma cannot be extended to robustness, as shown in Example 11.

Corollary 26. *If A is an unstable model, then there are (unstable) models B and C, such that $A \simeq B \subsetneq A \subsetneq C \simeq A$.*

Theorem 27. *There are models A and B, such that $A \succsim B$, while B is stable and A is not.*

Proof. Define $A = \{\lambda n.2k : k \in \mathbf{N}\}$ (all "even" constant functions over $\mathbf{N}$) and $B = \{\lambda n.k : k \in \mathbf{N}\}$ (all constant functions over $\mathbf{N}$). Since $\pi^{-1} \circ k \circ \pi$ is a constant function for every constant k, it follows that $\pi\langle B\rangle \subseteq B$, and thus B is stable. Clearly, $A \succsim_{\lambda n.2n} B$. Define the permutation

$$\pi(n) \quad = \quad \begin{cases} 1 & n = 0 \\ n+2 & n \text{ is odd} \\ n-2 & n \text{ is even} \end{cases}$$

Since $\pi(A) = A \cup \{\lambda n.1\} \notin A$, we have that A is not stable. □

Lemma 28. *If model A is robust and $A \succsim_\rho B \succsim_\pi A$, for model B, injection ρ and bijection π, then A and B are strongly equivalent models.*

Proof. Suppose A is robust, and $A \succsim_\rho B \succsim_\pi A$ for injection ρ and bijection π. It follows that $\pi\langle B\rangle = A' \supseteq A$. Thus, $A \succsim_\rho B \succsim_\pi A' \supseteq A$. Therefore, from the robustness of A, it follows that $A' = A$. Hence, $A' \succsim_{-\pi} B$, and A and B are strongly equivalent models. □

Theorem 29. *If A and B are strongly equivalent models, then A is stable iff B is, and A is robust iff B is.*

Proof. Suppose that A is robust and $A \simeq B \subsetneq C$. By Lemma 20, $C \simeq D \supsetneq A$ for some D. Were $B \succsim C$, then $A \simeq B \succsim C \simeq D$, contradicting the robustness of A. Hence, B is also robust. By the same argument, A is stable iff B is. □

Theorem 30. *If model A is robust and $A \simeq B \subsetneq C$, for models B, C, then $C \succ A$.*

Proof. If $A \simeq B \subsetneq C$, then $C \gtrsim B \gtrsim A$. And, by the previous theorem, if A is robust, then so is B; hence $B \not\gtrsim C$ and also $A \not\gtrsim C$. Hence, $C \succ A$. □

Corollary 31. *All finite models are robust. That is, if a model A has only a finite number of functions then A is robust.*

Proof. Follows from Lemma 15. □

6 The Recursive Functions

We turn now to specific computational models. We provide several results regarding the recursive, partial recursive, and primitive recursive functions. In addition, we provide a criterion for hyper-recursiveness.

Theorem 32. *The primitive recursive functions,* Prim*, are strictly weaker than the recursive functions.*

Proof. Clearly, Rec $\succsim_\iota$ Prim. So, assume, on the contrary, that Prim $\succsim_\rho$ Rec for some injection ρ. Let $S \in$ Rec be the successor function. There is, by assumption, a function $S' \in$ Prim such that $S' \circ \rho = \rho \circ S$. Since $\rho(0)$ is some constant and $\rho(S(n)) = S'(\rho(n))$, we have that ρ is derived by primitive-recursion from $S' \in$ Prim, thus $\rho \in$ Prim. Since ρ is a recursive injection, it follows that ρ^{-1} is partial recursive. Define the recursive function $h(n) = \rho(\min_i\{\rho(i) > ack(n,n)\})$, where ack is Ackermann's function. Since $\lambda n.ack(n,n)$ grows faster than any primitive recursive function and $h(n) > ack(n,n)$, it follows that $h \notin$ Prim. Since rng $h \subseteq$ rng ρ, it follows that $t = \rho^{-1} \circ h \in$ Rec. Thus, there is a function $t' \in$ Prim, such that $t' \circ \rho = \rho \circ t = \rho \circ \rho^{-1} \circ h = h$. We have arrived at a contradiction: on the one hand, $t' \circ \rho \in$ Prim, while, on the other hand, $t' \circ \rho = h \notin$ Prim. □

Definition 33 (Hyper-recursive). *Model M is* hyper-recursive *if there is an injection ρ, such that $\rho\langle M\rangle \supsetneq$* Rec*.*

Theorem 34. *The recursive functions* Rec *are robust. That is, they cannot simulate any hyper-recursive model.*

Proof. Assume Rec $\succsim_\rho M \supseteq$ Rec for some injection ρ, and let $S \in M$ be the successor function. Analogously to the proof of Theorem 32, $\rho \in$ Rec and ρ^{-1} is partial recursive. For every $f \in M$, there is an $f' \in$ Rec, such that $f = \rho^{-1} \circ f' \circ \rho$; thus f is partial recursive. Actually, f is total, since rng $(f' \circ \rho) =$ rng $(\rho \circ f) \subseteq$ rng ρ. Therefore, $M =$ Rec. □

By the same token:

Theorem 35. *The partial recursive functions* PR *are robust.*

Corollary 36. *The general recursive functions* (Rec) *and partial recursive functions* (PR) *are not strongly equivalent to any of their submodels or supermodels.*

Proof. Non-equivalence to supermodels is just Theorems 34 and 35. Non-equivalence to submodels follows from Lemma 25. □

As a corollary of Theorems 30 and 34, we obtain a criterion for hyper-recursiveness:

Corollary 37. *A model M, operating over a denumerable domain, is hyper-recursive if there is any bijection under which a proper subset of M simulates* Rec.

This justifies the use of Method III in the particular case of the recursive functions.

7 Turing Machines

In this section we show the robustness of Turing machines (TM) and their isomorphism to the recursive functions. We base the proof on known results, cited from [6].

Adopting the approach of [6], TM, as a machine, operates over an infinite tape of the symbols $\{0, 1, B\}$, while TM, as a computational model (set of partial functions), operates over $\{0,1\}^*$ [6, p. 116]:

> The tape must be represented finitely in order to define the transition rules. One way is to include all nonblank symbols, so a full tape is obtained by appending infinitely many blanks to each end of a finite tape representation. ... Input and output are strings in TM-data $= \{0,1\}^*$, are found on the first tape, and consist of all symbols to the right of the scanned symbol, extending up to but not including the first blank.

Theorem 38. *Turing machines,* TM*, and the recursive functions,* Rec*, are strongly equivalent.*

Proof. Since Rec is robust, it is sufficient, by Lemma 28, to show that Rec $\succsim$ TM $\succsim_\pi$ Rec, for some bijection π. Since it is well-known that Rec $\succsim$ TM via Gödelization (e.g. [6, pp. 208–109]), it remains to show that TM $\succsim_\pi$ Rec, for some bijection π. Define (as in [6, p. 131]) the bijection $\pi : \mathbf{N} \to \{0,1\}^*$, by

$$\pi(n) \quad = \quad \begin{cases} \epsilon & n = 0 \\ d \text{ s.t. } 1d \text{ is the shortest binary representation of } n+1 & \text{otherwise} \end{cases}$$

For example, $\pi(0, 1, 2, 3, 4, 5, 6, 7, \ldots)$ is $\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$.

TM $\succsim_\pi$ *RAM*(Random Access Machine), by [6, pp. 131–133]; $RAM \supseteq CM$ (Counter Machine), by [6, pp. 116–118]; and $CM \succsim_\iota$ Rec by [6, pp. 207–208]. We have that Rec $\succsim$ TM $\succsim_\pi RAM \succsim_\iota CM \succsim_\iota$ Rec, thus TM and Rec are strongly equivalent. □

Note that the exact definitions of RAM and CM are of no importance, as they are only intermediates for Rec $\succsim$ TM $\succsim_\pi RAM \succsim_\iota CM \succsim_\iota$ Rec.

Theorem 39. *Turing machines,* TM*, are robust.*

Proof. By Theorem 38, TM and Rec are strongly equivalent. Since Rec is robust, it follows, by Theorem 29, that TM is robust. □

Corollary 40. *Turing machines,* TM*, and the recursive functions,* Rec*, are isomorphic.*

Proof. Follows directly from Theorem 38 and Theorem 24. □

8 Inductive Domains

The robustness of the (general and partial) recursive functions is due to several properties, among which is the inclusion of a successor function (Theorem 34). The results herein can be extended to show that computational models operating over other inductively-defined domains are also robust.

We illustrate the above with the following example.

Example 41. *We define the schematic WHILE-like programming language,* PLP*, over* $\{a, b\}^*$*, and show its robustness.*

The definition of PLP:

- *An* atom *is an element of* $\{a, b\}^*$*.* FALSE *denotes the atom* ϵ*, and* TRUE *stands for any other atom.*
- *The input and output of a program in PLP is an atom.*
- *A program in PLP is a countable set of expressions.*
- *An* expression *is a variable, basic operation or a program call. PLP supports recursive program calls.*
- *A* variable *in the language can hold an atom, and is initialized to* ϵ*. There is a special variable,* output*, which is the output value of the program.*
- *Below is the list of basic operations. Capital letters stand for variables, and small letters for expressions. The value of a basic operation is TRUE, unless defined otherwise.*
  - set X := e*, assigning the value of the expression e to the variable X.*
  - appendA(X)*, returning aX.*
  - appendB(X)*, returning bX.*
  - equal(x,y)*, returning TRUE if the value of x and y is the same atom, and FALSE otherwise.*
  - while e $\{e_1, \ldots, e_n\}$*, executing the expressions* $e_1, \ldots, e_n$ *as long as the expression e is TRUE.*
  - P(e)*, executing the program P with the input given by the value of e. Returning the output of the program P.*

The robustness of PLP:
Let $F(PLP)$ denote the set of functions computed by the programs of PLP. Assume $F(PLP) \succsim_{\rho} M \supseteq F(PLP)$ for some injection ρ, and let S_a and S_b be the functions computed by the operations appendA and appendB, respectively. There are, by assumption, functions $S'_a \in F(PLP)$ and $S'_b \in F(PLP)$ such that $S'_a \circ \rho = \rho \circ S_a$ and $S'_b \circ \rho = \rho \circ S_b$. Thus, ρ can be programmed in PLP:

$$\rho(x) \quad = \quad \begin{cases} atom \;\; k & x = \epsilon \\ S'_a \circ \rho(y) & x = ay \\ S'_b \circ \rho(y) & x = by \end{cases}$$

Since PLP has the 'while', 'appendA' and 'appendB' operations, it follows that $F(PLP)$ is closed under the inverse of total functions. Thus, ρ^{-1} can also be programmed in PLP. For every $f \in M$, there is an $f' \in F(PLP)$, such that $f = \rho^{-1} \circ f' \circ \rho$; Since PLP supports program calls, it follows that $F(PLP)$ is closed under functional composition, thus f is also in $F(PLP)$. Therefore, $M = F(PLP)$.

9 Neutrality

As shown in Section 3, a model can be strongly equivalent to its supermodel. That is, there are models that can simulate their supermodels via bijections. There are, however, families of encodings via which no model can simulate its supermodel. We call such a family *neutral*, and provide a sufficient and necessary condition for neutrality.

Definition 42 (Neutral Family). *A family of encodings F is* neutral *if for all models A and B and encodings $\rho \in F$, $A \succsim_\rho B \supseteq A$ implies $A = B$.*

Definition 43 (Bounded Cycle Permutations). *An encoding $\pi : D \to D$ is a* bounded cycle permutation *if there is a constant $k \in \mathbf{N}$, such that $\pi^k(x) = x$, for every $x \in D$.*

Definition 44 (Cycle). *A* cycle *of an element $x \in D$ with respect to an injection $\rho : D \to D$, denoted $cycle_\rho(x)$, is the set $\{\rho^i(x) : i \in \mathbf{Z}\}$.*

Theorem 45. *For every encoding $\rho : D \to D$, there are models A and B, such that $A \subsetneq B \precsim_\rho A$, iff ρ is an unbounded cycle permutation.*

Proof. Suppose that π is a permutation with cycles bounded by k. Assume $A \succsim_\pi B \supseteq A$. There is, by assumption, a function $f \in A$, for every function $g \in B$, such that $g = \pi^{-1} \circ f \circ \pi$. Since $f \in B$, there is, by induction, a function $f_k \in A$, such that $g = \pi^{-k} \circ f_k \circ \pi^k = f_k$. Therefore, $B = A$.

For the other direction, we must consider three cases: (i) non-surjective encodings; (ii) surjective encodings that are not injective; (iii) bijections with no bound on the length of their cycles. We prove each case by constructing a computational model that simulates a supermodel of itself via the given encoding.

Case i. Suppose that $\rho : D \to D$ is non-surjective, and let $k \in (D \backslash \text{rng } \rho)$. Define $M = \{\lambda x.\rho^i(k) : i \in \mathbf{N}\}$, and $M' = M \setminus \{\lambda x.k\}$. Since $k \notin rng\rho$, it follows that $\{\lambda x.k\} \notin M'$, thus $M' \subsetneq M$. Since $\rho^{-1} \circ \lambda x.\rho^{i+1}(k) \circ \rho = \lambda x.\rho^i(k)$, we have that $M' \succsim_\rho M$.

Case ii. Suppose that $\rho : D \to D$ is surjective and not injective, and let $k \in D$, such that $\rho(a) = \rho(b) = k$, for some $a \neq b$ in D. Since ρ is a function, it follows that at least one of $\{a, b\}$ is not in $\{\rho^i(k) : i \in \mathbf{N}\}$. We assume, without loss of generality, that $a \notin \{\rho^i(k) : i \in \mathbf{N}\}$. Define $M = \{\lambda x.\rho^i(a) : i \in \mathbf{N}\}$, and $M' = M \setminus \{\lambda x.a\}$. By the same argument of case (i), we have that $M' \subsetneq M$, and $M' \succsim_\rho M$.

Case iii. Suppose that $\rho : D \to D$ is an unbounded cycle permutation. Assume, by the axiom of choice, a function $c(x)$, such that $c(x) \in cycle_\rho(x)$,

for all $x \in D$, and $c(x) = c(y)$ if $cycle_\rho(x) = cycle_\rho(y)$, for all x and $y \in D$. Define $M = \{\lambda x.\rho^i(c(x)) : i \in \mathbf{N}\}$, and $M' = M \setminus \{\lambda x.c(x)\}$. Since the cycles of ρ are not bounded, it follows that $M' \subsetneq M$. By the same argument of case (i), we have that $M' \succsim_\rho M$. □

Corollary 46. *A family of encodings F is neutral iff all encodings ρ in F such that* dom $\rho =$ rng ρ *are bounded-cycle permutations.*

10 Unconditional Neutrality

Is neutrality an inherent property of a family of encodings, or is it context-dependent? Since the bounded-cycles property is not transitive (i.e. the composition of two bounded cycle permutations might yield an unbounded cycle permutation), the answer is, in general, that neutrality is context dependent. There are, however, families of encodings that are inherently neutral, named *unconditionally neutral*. We provide a sufficient and necessary condition for an unconditionally neutral family. As the families of encodings are aimed for a quasi-ordering comparison notion, we consider only families closed under composition that include the identity.

Definition 47 (Closure). *A* closure *of a family of encodings F, denoted $closure(F)$, is the family of encodings $\{\varphi \circ \psi : \varphi, \psi \in F\}$.*

Definition 48 (Unconditionally Neutral). *A family of encodings F is* unconditionally neutral *if $closure(G \cup F)$ is Neutral, for every neutral family G closed under composition.*

Definition 49 (Finite Changes Permutations). *A permutation $\pi : D \to D$ is a* finite changes permutation *if there is a constant $k \in \mathbf{N}$, such that $|\{x \in D : \pi(x) \neq x\}| = k$.*

Lemma 50. *If φ and ψ are bounded cycle permutations, such that $\varphi \circ \psi$ is bounded cycles, and π is a finite changes permutation, then $\varphi \circ \pi \circ \psi$ is a bounded cycle permutation.*

Proof. Suppose that a permutation π changes k elements of a domain D, for some constant $k \in \mathbf{N}$. Define $S = \{x \in D : \varphi \circ \pi \circ \psi(x) \neq \varphi \circ \psi(x)\}$. Since π changes only k elements of D, it follows that $|S| = k$. Thus, a cycle w.r.t $\varphi \circ \pi \circ \psi$ is a combination of at most k cycles w.r.t $\varphi \circ \psi$, therefore it is bounded. □

Theorem 51. *A family of encodings F is unconditionally neutral iff all its encodings are finite changes permutations.*

Proof. Suppose that G is a neutral family closed under composition, and F is a family of finite changes permutations. An encoding $\pi \in closure(G \cup F)$ is of the form $\pi = \alpha_1 \circ \beta_1 \circ \alpha_2 \circ \beta_2 \circ \alpha_3 \circ \beta_3 \circ \cdots \circ \alpha_n \circ \beta_n$, for some $n \in \mathbf{N}$, where $\alpha_i \in G$ and $\beta_i \in F$ for all $i \in \{1, N\}$. (Adding the identity as the first/last element of the composition ensures this canonic form). Since α_1 and α_2 are bounded cycle permutations, it follows, by Lemma 50, that $\alpha_1 \circ \beta_1 \circ \alpha_2$ is bounded cycles. Since $\alpha_2 \circ \alpha_3 \in G$, it follows that it is bounded. Thus, by

Lemma 50 $\alpha_1 \circ \beta_1 \circ \alpha_2 \circ \alpha_3$ is bounded cycles, and therefore $\alpha_1 \circ \beta_1 \circ \alpha_2 \circ \beta_2 \circ \alpha_3$ is bounded cycles. By induction on i (the index of the encodings of G and F from which π is composed), we have that π is a bounded cycle permutation.

For the other direction, suppose that $\pi : D \to D$ is a permutation changing infinite elements of D. By assumption, there is a denumerable set $S = \{C_1, C_2, \ldots\}$ of cycles w.r.t π of length ≥ 2. Assume, by the axiom of choice, a function $m : S \to D$, such that $m(C_i) \in C_i$, for all $C_i \in S$. Abbreviate, $m_i = m(C_i)$. Define the permutation

$$\tau(x) \quad = \quad \begin{cases} m_{i+1} & x = \pi(m_i) \text{ for some } i > 0 \\ \pi(m_{i-1}) & x = m_i \text{ for some } i > 1 \\ x & \text{otherwise} \end{cases}$$

Clearly, the cycles w.r.t τ are of length ≤ 2. Since $cycle_{\tau \circ \pi}(m_1)$ is unbounded, we have that $\tau \circ \pi$ is an unbounded cycle permutation. $\square$

11 Standard Computational Properties

It is standard to assume that a computational model, defined as a set of functions, is closed under functional composition. See, for example, [5, p. 17]. It is also common to assume that it includes the identity function, and, possibly, all the constant functions. For models operating over $\mathbf{N}$, it is common to assume a successor function as well. In this section we show that the results herein are valid, even when considering only computational models with the above properties.

Closure under Functional Composition. A model closed under functional composition can still be strongly equivalent to its supermodel (e.g. Example 13). Furthermore, considering only models closed under functional composition doesn't change the sufficient and necessary condition for a neutral family (Theorem 45), as all models involved in the proof are closed under functional composition.

The Successor Function. The robustness of the (general and partial) recursive functions is due to several properties, among which is the inclusion of a successor function (Theorem 34). It turns out, however, that a model including the successor function and closed under functional composition, can still be strongly equivalent to its supermodel. We demonstrate this with the following example:

Example 52. *Define the permutation π over $\mathbf{N}$,*

$$\pi(n) \quad = \quad \begin{cases} 1 & n = 0 \\ n+2 & n \text{ is odd} \\ n-2 & n \text{ is even} \end{cases}$$

Let s be the successor function over $\mathbf{N}$, define the model $A = \{\pi^{-i} \circ s \circ \pi^i : i \in \mathbf{N}\}$, and define $s' = \pi \circ s \circ \pi^{-1}$. Since $(\pi^{-i} \circ s \circ \pi^i)(1)$ is even, for all $i \in \mathbf{N}$, and $s'(1) = 3$, it follows that $s' \notin A$. Define $B = closure(A)$. Clearly, every function $f \in B$ is of the form $\pi^{-i_1} \circ s \circ \pi^{i_1 - i_2} \circ s \circ \pi^{i_2 - i_3} \cdots \pi^{i_{n-1} - i_n} \circ s \circ \pi^{i_n}$, for some $n \in \mathbf{N}$. Since every function $f \in B \setminus A$ is composed of at least two functions that are composed of s, it follows that $|\mathbf{N} \setminus \text{rng } f| \geq 2 > 1 = |\mathbf{N} \setminus \text{rng } s'|$, for every $f \in B$. Thus, $s' \notin B$. Clearly, $B \subseteq \pi(B)$. Since $s' \in \pi(B)$, it follows that $B \subsetneq \pi(B)$, and therefore B is unstable.

The Identity and Constant Functions. Adding, or removing, the identity function from a model has no influence on its stability or robustness,

as $\rho^{-1} \circ (identity \circ f \circ identity) \circ \rho = \rho^{-1} \circ f \circ \rho$, for every injection ρ and function f.

Define the set of all constant functions over a domain D, by $K = \{\lambda x.k : k \in D\}$. Adding, or removing, K from a model A over D, such that $A \cap K \in \{K, \emptyset\}$, has no influence on the stability or robustness of A, as $\rho\langle A \circ K\rangle = \rho\langle K \circ A\rangle = K$, w.r.t total functions, for every injection ρ and model A.

12 Discussion

Containment and Computational Power. As an arbitrary set can have the same cardinality as its proper superset, it turns out that an arbitrary computational model (set of functions) can be of equivalent computational power to its supermodel (proper superset of functions). However, what seems natural (for the last century) for arbitrary sets, is not so natural for computational models. Indeed, it turns out that there are models, which are robust, in the sense that no encoding can make them equivalent to their supermodels, for example, Turing machines and the recursive functions (Sections 6 and 7). Insisting that containment implies a computational power relation, leads to unreasonable limits on the possible mappings between domains (Sections 9, 10).

A Well Defined Computational Model. It is standard to view a computational model as a set of functions (i.e. to consider its extensionality). See, for example, [9, pp. 9–10], or [11, p. 30]: "The computational power of the model is represented by the extension of the set of all functions computable according to the model." On the other hand, for considering a set of functions as a computational model, it is standard to require that the set is closed under functional composition, includes the identity, and possibly, includes all the constant functions; For models operating over $\mathbf{N}$ it is also standard to require a successor function. We argue that a well defined computational model should also be stable (Definition 21), which is not guaranteed by the properties above (Section 11). Furthermore, a computational model that represents a class of computability power (as the recursive functions) should also be robust (Definition 22).

Effectivity. A different approach to comparing models over different domains is to require some manner of effectiveness of the encoding; see [4, p. 21] and [5, p. 290], for example. There are basically two approaches:

1. One can demand an informal effectiveness: "The coding is chosen so that it is itself given by an informal algorithm in the unrestricted sense" [9, p. 27].

2. One can require effectiveness of the encoding function via a specific model, usually Turing machines: "The Turing-machine characterization is especially convenient for this purpose. It requires only that the expressions of the wider classes be expressible as finite strings in a fixed finite alphabet of basic symbols" [9, p. 28].

Effectivity is a useful notion; however, it is unsuitable for our purposes. The first, informal approach is too vague, while the second can add computational power when dealing with subrecursive models and is inappropriate when dealing with non-recursive models.

13 Further Research

There are various directions in which one can extend the work described above:

Multivalued Representations. It may be useful to allow several encodings of the same element, as long as there are no two elements sharing one representation, something injective encodings disallow. Consider, for example, representing rationals as strings, where '1/2', '2/4', '3/6', ..., could encode the same number. See, for example, [14, p. 13]. To extend the notion of computational power (Definition 3) to handle multivalued representations, we would say that model $A \succsim B$ if there is a partial surjective function $\eta : \text{dom } A \to \text{dom } B$ ($\eta(y) = \perp$ iff $y = \perp$), such that there is a function $f \in A$ for every function $g \in B$, with $\eta(f(x)) = g(\eta(x))$ for every $x \in \text{dom } \eta$. This follows along the lines suggested in [14, p. 16]. The corresponding definitions and results need to be extended accordingly.

Different Domain and Range. The simulation definition (Definition 2) naturally extends to models $M : D^k \to D$ with multiple inputs, by using the same encoding ρ for each input component. See, for example, [11, p. 29].

A more general definition is required for models with distinct input and output domains. This can be problematic as the following example illustrates:

Example 53. *Let* RE *be the recursively enumerable sets of naturals. We define infinitely many non-r.e. partial predicates* $\{h_i\}$*, which can be simulated by* RE*. Let*

$$h(n) = \begin{cases} 0 & \textit{program } n \textit{ halts uniformly} \\ 1 & \textit{otherwise} \end{cases}$$

$$h_i(n) = \begin{cases} 0 & n < i \vee h(n) = 0 \\ \perp & \textit{otherwise}\,. \end{cases}$$

We have that $\text{RE} \succsim_\rho \text{RE} \cup \{h_i\}$*, where*

$$\rho(n) = 2n + h(n)$$

$$h_i'(n) = \begin{cases} 0 & \lfloor n/2 \rfloor < i \vee n \bmod 2 = 0 \\ \perp & \textit{otherwise} \end{cases}$$

$$\rho(f) = \begin{cases} f(\lfloor n/2 \rfloor) & f \in \text{RE} \\ h_i'(n) & f = h_i\,. \end{cases}$$

Without loss of generality, we are supposing that $\rho(0) = h(0) = 0$*.*

Different Cardinalities. It may sometimes be unreasonable to insist that the encoding be injective, since the domain may have elements that are distinct, but virtually indistinguishable by the programs. For example, a model may operate over the reals, but treat all numbers $[n : n + 1)$ as representations of $n \in \mathbf{N}$.

Nondeterministic Models. The computational models we have investigated are deterministic (Definition 1). The corresponding definitions and results should be extended to nondeterministic models, as well.

References

[1] M. Burgin. How we know what technology can do. *Communications of the ACM*, 44:82–88, Nov. 2001.

[2] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[3] N. Cutland. *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, Cambridge, 1980.

[4] E. Engeler. *Formal Languages: Automata and Structures*. Lectures in Advanced Mathematics. Markham Publishing Company, Chicago, IL, 1968.

[5] F. Hennie. *Introduction to Computability*. Addison-Wesley, Reading, MA, 1977.

[6] N. D. Jones. *Computability and Complexity From a Programming Perspective*. The MIT Press, Cambridge, Massachusetts, 1997.

[7] S. Kleene. Lambda-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

[8] B. M. Moret. *The Theory of Computation*. Addison-Wesley, Reading, MA, 1998.

[9] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York, 1966.

[10] H. T. Siegelmann. *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser, Boston, 1998.

[11] R. Sommerhalder and S. C. van Westrhenen. *The Theory of Computability: Programs, Machines, Effectiveness and Feasibility*. Addison-Wesley, Workingham, England, 1988.

[12] G. J. Tourlakis. *Computability*. Reston Publishing Company, Reston, VA, 1984.

[13] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936–37. Corrections in vol. 43 (1937), pp. 544-546. Reprinted in M. Davis (ed.), "The Undecidable," Raven Press, Hewlett, NY, 1965. Available at: http://www.abelard.org/turpap2/tp2-ie.asp.

[14] K. Weihrauch. *A Simple Introduction to Computable Analysis*. Fern Universität, Hagen, Germany, July 1995. Available at ftp://ftp.eccc.uni-trier.de/pub/eccc/books/Weihrauch/book.ps.