Comparing Computational Power

UDI BOKER, *School of Computer Science, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel. E-mail: udiboker@tau.ac.il*

NACHUM DERSHOWITZ, *School of Computer Science, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel. E-mail: nachumd@tau.ac.il*

All models are wrong but some are useful.

—George E. P. Box (1979)

Abstract

It is common practice to compare the computational power of different models of computation. For example, the recursive functions are strictly more powerful than the primitive recursive functions, because the latter are a proper subset of the former (which includes Ackermann's function). Side-by-side with this "containment" method of measuring power, it is also standard to base comparisons on "simulation". For example, one says that the (untyped) lambda calculus is as powerful—computationally speaking—as the partial recursive functions, because the lambda calculus can simulate all partial recursive functions by encoding the natural numbers as Church numerals.

The problem is that unbridled use of these two distinct ways of comparing power allows one to show that some computational models (sets of partial functions) are *strictly* stronger than themselves! We argue that a better definition is that model A is strictly stronger than B if A can simulate B via some encoding, whereas B cannot simulate A under *any* encoding. We show that with this definition, too, the recursive functions are strictly stronger than the primitive recursive. We also prove that the recursive functions, partial recursive functions, and Turing machines are "complete", in the sense that no injective encoding can make them equivalent to any "hypercomputational" model.[1]

Keywords: Computational models, Computational power, Simulation, Hypercomputation

1 Introduction

Our overall goal is to formalize the comparison of computational models. We seek a robust definition of relative power that does not itself depend on the notion of computability. It should allow one to compare arbitrary models over arbitrary domains via a quasi-ordering that successfully captures the intuitive concept of computational strength. Eventually, we want to be able to prove statements like "analogue machines are strictly more powerful than digital devices", even though the two models operate over domains of different cardinalities.

Since we are only interested here in the extensional quality of a computational model (the set of functions or relations that it computes), not complexity-based comparison or step-by-step simulation, we use the term "model" for any set of partial functions, and ignore all "mechanistic" aspects.

[1]This research was supported by the Israel Science Foundation (grant no. 250/05) and was carried out in partial fulfillment of the requirements for the Ph.D. degree of Udi Boker.

1.1 The Standard Comparison Method

There are basically two standard methods, Approaches C and S below, by which models have been compared over the years. These two approaches have been used in the literature in conjunction with each other; thus, they need to be able to work in harmony. That is, if models A and A' are deemed equivalent according to approach C, while A' is shown to be stronger than B by approach S, we should expect that it is legitimate to infer that A is also stronger than B.

Approach C (Containment). Normally, one would say that model A is at least as powerful as B if all (partial) functions computed by B are also computed by A. If A allows *more* functions than B, then it is standard to claim that A is *strictly* stronger. For example, general recursion (Rec) is more powerful than primitive recursion (Prim) (e.g. [12, p. 92]), and inductive Turing machines are more powerful than Turing machines [2, p. 86].

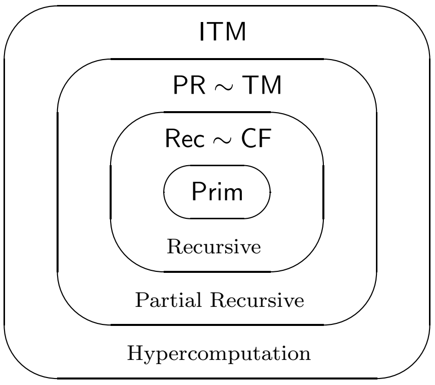
Approach S (Simulation). The above definition does not work, however, when models use different data structures (representations). Instead, A is deemed at least as powerful as B if A can *simulate* every function computable by B. Specifically, the simulation is obtained by requiring an injective encoding ρ from the domain of B to that of A, such that for every function g computed by B we have $g = \rho^{-1} \circ f \circ \rho$ for some function f computed by A, in which case A is said to be at least as powerful as B. (See Definition 2.4 below.) As one textbook states [11, p. 30]:

> 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'.

Similar statements may be found, for example, in [9, p. 27] and [4, p. 24].

Equipotence. To show that two models are of equivalent power by the simulation method, one needs to find two injections, each showing that every function computed by one can be simulated by the other. For example, the Turing-computable partial functions (TM), the untyped lambda calculus (Λ), and the partial recursive functions (PR) were all shown to be of equal computational power, in the seminal work of Church [3], Kleene [8] and Turing [13].

More Powerful. To show that model A is strictly more powerful than model B, one normally shows that A is at least as powerful as some model A' that comprises more functions than B ($A' \supsetneq B$). (See, for example, [10].) Figure 1 illustrates this standard conception, according to which the computable functions (CF), computed by halting Turing machines, are considered strictly more powerful than primitive recursion, since CF is equivalent to Rec—by simulation, and Rec is strictly more powerful than Prim—by containment.



ITM = Inductive Turing Machines
PR = Partial Recursion
TM = Turing Machines
Rec = General Recursion
CF = Halting Turing Machines
Prim = Primitive Recursion

FIG. 1. Computational power hierarchy

1.2 The Problem and Solution

Unfortunately, it turns out that these two approaches, which form the standard method of comparing computational power, are actually incompatible. We provide examples, in Section 3, of cases in which model A is strictly more powerful than B by the first approach, whereas B is at least as powerful as A by the second. It follows that the combination of these two standard approaches allows for models to be strictly stronger than themselves!

Specifically, in Example 3.1 below, we describe a model that is a proper subset of the recursive functions, but can, nevertheless, simulate all of them. This raises the question whether, for instance, it could possibly also be the case that the primitive recursive functions are of equivalent power to Turing machines, via some "wild" simulation. Could it be that the recursive functions are of equivalent computational power to some proper superset, containing non-recursive functions?

To resolve this issue, we begin (in Definition 2.6 below) with the basic comparison notion "as powerful as" ($\succsim$), using the simulation approach (Approach S), which naturally extends containment (Approach C) to models operating over different domains. Then the "strictly more powerful" partial ordering ($\succ$) is derived from the quasi-ordering $\succsim$ by saying that $A \succ B$ if $A \succsim B$ but not $B \succsim A$; in other words, only when there is no injection via which B can simulate A.

To compare models operating over different domains requires some sort of mapping between the domains. One possible alternative might be to require a domain mapping that is not only injective, but that also possesses additional properties, like surjectiveness. It turns out, however, that bijective mappings not only cannot provide a sufficiently general comparison notion, but to work in harmony with the containment approach (Approach C) they would have to be limited to permutations with bounded orbits, an unpalatable restriction (Theorem 3.4).

One is tempted to judge computational models to be "well-defined" only when they cannot be shown by simulation to be of equivalent power to any proper superset of the functions they compute. We call such models "complete" (Definition 4.1).

The question then is: Are classic models, such as Turing machines, well-defined? In Section 4, we show that general recursive functions, partial recursive functions, and Turing machines are indeed all complete models in this sense (Theorems 4.7, 4.8, and 4.12). Accordingly, we obtain a criterion by which to verify that a model operating over a denumerable domain is hypercomputational (Corollary 4.10).

2 Comparing Power

We treat here only deterministic computational models; hence, we deal with partial functions, referred to plainly as "functions" below. 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" function values.

As usual, two partial functions (f and g) over the same domain (D) are deemed (*semantically* or *extensionally*) *equal* (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 2.1 (Model of Computation)
Let D be an arbitrary domain (any set of elements). A *model of computation over D* is any set of functions $f : D \to D \cup \{\perp\}$. We write dom A for the domain over which model A operates.

Since models are sets: When $A \subseteq B$, 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. Moreover, whenever we claim that $A \subseteq B$, we mean to also imply that the two models operate over the same domain.

2.1 Injective Mappings

To deal with models operating over different domains, however, it is incumbent to map the domain of one model to that of the other.

Definition 2.2 (Encoding)
Let D_A and D_B be the domains of two models. An *encoding* is an injection $\rho : D_B \cup \{\perp\} \to D_A \cup \{\perp\}$, with the restriction that $\rho(y) = \perp$ iff $y = \perp$ (i.e. ρ is total, one-one, and strict).

We write $\rho \circ M$ for $\{\rho \circ g : g \in M\}$ and $M \circ \rho$ for $\{f \circ \rho : f \in M\}$, where ρ is an encoding and M is a model.

Definition 2.3 (Function Simulation)
Let D_A and D_B be the domains of two models. We say that function $f : D_A \to D_A$ *simulates* function $g : D_B \to D_B$ *via* injection ρ if $\rho^{-1} \circ f \circ \rho = g$, or, equivalently, $f \circ \rho = \rho \circ g$.

Since ρ is an injection, ρ^{-1} is a partial function. See Fig. 2.

We will say that one model simulates another if every function of the latter is simulated by some function of the former:

Model A computes all functions of model B via mapping ρ.

$$\begin{tikzcd}
D_B \arrow[r, "g \in B"] \arrow[d, "\rho"'] & D_B \arrow[d, "\rho"] \\
D_A \arrow[r, "f \in A"'] & D_A
\end{tikzcd}$$

FIG. 2. Model Simulation

DEFINITION 2.4 (Model Simulation)
Model A *simulates* model B *via* injection $\rho : \text{dom } B \to \text{dom } A$, denoted $A \succsim_\rho B$, if $\rho \circ B \subseteq A \circ \rho$.

This is the notion of "incorporated" used in [11, p. 29].

EXAMPLE 2.5
Turing-computable functions (CF) simulate the recursive functions (Rec) via a unary representation of the natural numbers.

As a degenerate case, with the identity encoding ι $(\lambda x.x)$, we have $A \succsim_\iota B$ iff $A \supseteq B$. The containment approach (C) to comparison of models (see the introduction) uses this simple relation.

The simulation-based approach (S) is embodied in the following:

DEFINITION 2.6 (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 $A \sim B$.

PROPOSITION 2.7
The computational power relation $\succsim$ between models is a quasi-order. Computational equivalence $\sim$ is an equivalence relation.

Transitivity of $\succsim$ follows from the fact that the composition of injections is an injection.

EXAMPLE 2.8
The (untyped) λ-calculus (Λ) is computationally equivalent to the partial recursive functions (PR), via Church numerals, on the one hand, and via Gödelization, on the other.

Since domain encodings imply function mappings—by simulation (Definition 2.3), we extend them to (partial) functions and models, as follows:

DEFINITION 2.9 (Function Mappings)
An injective encoding $\rho : \text{dom } B \to \text{dom } A$ between the domains of two models A and B induces a mapping

$$\rho(g) \quad = \quad \rho \circ g \circ \rho^{-1}$$

of functions $g \in B$ to functions over the domain of A. Viewing partial functions as sets of pairs, this is:

$$\rho(g) \quad = \quad \{(\rho(x), \rho(y)) \; : \; (x, y) \in g\} \; .$$

The same encoding induces a mapping

$$\rho\langle f\rangle \quad = \quad \rho^{-1} \circ f \circ \rho$$

from $f \in A$ to functions over dom B. These mappings extend to sets of functions M 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 partial function f extending $\rho(g)$ (i.e. $f \restriction_{\mathrm{rng}\,\rho} = \rho(g) \restriction_{\mathrm{rng}\,\rho}$) simulates g via ρ, while $\rho\langle f\rangle$ is the only function simulated by f.

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:

Theorem 2.10
For all models A and B and injections ρ, $A \succsim_\rho B$ iff $B \subseteq \rho\langle A\rangle$.

Proof. By definition, $B \subseteq \rho\langle A\rangle$ iff for every $g \in B$ there is an $f \in A$, such that $g = \rho^{-1} \circ f \circ \rho$. This is the same as requiring that $\rho \circ g = \rho \circ \rho^{-1} \circ f \circ \rho = f \circ \rho$, which is what is demanded by $A \succsim_\rho B$. (Cf. Fig. 2.) ∎

Corollary 2.11
For all models A and injections ρ, $A \succsim \rho\langle A\rangle$.

Proposition 2.12
For all models B and C and injections ρ, $B \subseteq C$ implies that $\rho\langle B\rangle \subseteq \rho\langle C\rangle$.

Proposition 2.13
For all models B and C and injections ρ, $B \subsetneq C$ implies that $\rho(B) \subsetneq \rho(C)$.

Proof. The mapping $f \mapsto \rho(f)$ for functions is injective: Let $\rho(f) = \rho(g)$, that is, $\rho \circ f \circ \rho^{-1} = \rho \circ g \circ \rho^{-1}$. Then $f = \rho^{-1} \circ \rho \circ f \circ \rho^{-1} \circ \rho = \rho^{-1} \circ \rho \circ g \circ \rho^{-1} \circ \rho = g$. Hence, if $B \subsetneq C$, then $\rho(C)$ has a function not in $\rho(B)$. ∎

2.2 Bijective Mappings

Stronger notions of equivalence of models under simulation can be based on bijections, for which $\pi\langle A\rangle = \pi^{-1}(A)$:

Definition 2.14 (Strong Equivalence)
Models A and B are *strongly equivalent*, denoted $A \simeq B$, if there are bijections π and τ such that $A \succsim_\pi B \succsim_\tau A$.

Example 2.15
Let $A = \{f_{i,j} \, : \, i, j > 0\}$ and $B = A \cup \{f_{1,0}\}$, where $f_{i,j} = \lambda n.(\lfloor\sqrt{n}\rfloor + i)^2 + j \bmod (2\lfloor\sqrt{n}\rfloor + 2i + 1)$, be two sets of total functions over the natural numbers. They are strongly equivalent, in that $A \succsim_\pi B \succsim_\iota A$, for a permutation π of the naturals. See Example 3.6 below for more details.

Definition 2.16 (Isomorphism)
Models A and B are *isomorphic*, denoted $A \equiv B$, if there is a bijection π such that $A \gtrsim_{\pi} B \gtrsim_{\pi^{-1}} A$.

Example 2.17
The programming language, Lisp, with only pure lists as data, is isomorphic to the partial recursive functions via the Gödel pairing function: $\pi(\mathbf{nil}) = 0$; $\pi(\mathbf{cons}(x, y)) = 2^{\pi(x)}(2\pi(y) + 1)$.

Example 2.18
Turing machines (TM) and the partial recursive functions (PR) are isomorphic. (See Theorem 4.11 below.)

Obviously:

Proposition 2.19
Isomorphism of models implies their strong equivalence.

When models operate over $\mathbb{N}$ and the bijection π is recursive, one may speak of "recursive isomorphism": function f is *recursively isomorphic* to g if there is a recursive permutation π, such that $f = \pi\langle g\rangle$ [9, pp. 52–53].[2]

By the same argument as for injections (Theorem 2.10):

Theorem 2.20
For all models A and B and bijections π, $A \gtrsim_{\pi} B$ iff $A \supseteq \pi(B)$.

Corollary 2.21
For all models A and bijections π, A and $\pi(A)$ are isomorphic ($A \equiv \pi(A)$).

Proof. Applying the theorem twice, we have $\pi(A) \gtrsim_{\pi} A$ and $A = \pi^{-1}(\pi(A)) \gtrsim_{\pi^{-1}} \pi(A)$. ∎

We will be needing the following two lemmata:

Lemma 2.22
For all models B and C and bijections π, $B \subsetneq C$ implies that $\pi\langle B\rangle \subsetneq \pi\langle C\rangle$.

Proof. Since $\pi \circ \pi^{-1}$ is total, by an analogous argument to that of Proposition 2.12, the function mapping $f \mapsto \pi\langle f\rangle$ is injective. Hence, if $B \subsetneq C$, then $\pi\langle C\rangle$ has a function not in $\pi\langle B\rangle$. ∎

Lemma 2.23
If $A \simeq B \subsetneq C$, for models A, B, and C, then there is a model $D \supsetneq A$, such that $C \simeq D$.

Proof. Suppose $B \gtrsim_{\pi} A$ for bijection π. By Theorem 2.10, $A \subseteq \pi\langle B\rangle$. Let $D = \pi\langle C\rangle$, for which we have $C \simeq D$. Since $B \subsetneq C$, it follows from the previous lemma that $A \subseteq \pi\langle B\rangle \subsetneq \pi\langle C\rangle = D$. ∎

[2]Moreover: "A property of a k-ary relations on $\mathbb{N}$ 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 $\mathbb{N}$. Thus, one may claim: "[Recursion] theory essentially studies . . . those properties of sets and functions which remain invariant under recursive permutations. For example, recursiveness, r.e.-ness, m-completeness are such invariants" [12, p. 333].

3 Comparing Submodels

Unfortunately, the above standard definition of "simulates" (Approach S, Definition 2.4) allows for the possibility that a model be equivalent to one of its strict supermodels.

EXAMPLE 3.1
The set of "even" recursive functions (R_2) is of equivalent computational power to the set of all recursive functions, where

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

We have that $R_2 \gtrsim_{\lambda n.2n} \mathsf{Rec}$.

This example also shows that the standard comparison method, combining Approaches C and S (see Section 1.1), and denoted temporarily by $\succ'$, is ill-defined as it allows situations where $A \succ' B \succ' A$ for models A, B. For example, the set of "odd" recursive functions (R_1, defined analogously) is of equivalent power to the set of all recursive functions, by the same argument as above. We have that, $R_1 \gtrsim \mathsf{Rec} \supsetneq R_2 \gtrsim \mathsf{Rec} \supsetneq R_1$, thus $R_1 \succ' R_2 \succ' R_1$.

It turns out that the equivalence of a model and its strict supermodel is possible even when the encoding ρ is a bijection and the model is closed under functional composition. Hence, some models are actually isomorphic to some of their strict supermodels. If we choose to restrict ourselves to encodings that preclude such anomalies, then, not only should we restrict ourselves to bijective encodings, but the bijections must be "narrow":

DEFINITION 3.2 (Narrow Permutations)
A permutation $\pi : D \to D$ is *narrow* if all its orbits (cycles) are bounded in length by some constant. In other words, if $\exists k \in \mathbb{N}.\ \forall x \in D.\ |\{\pi^n(x) : n \in \mathbb{N}\}| \leq k$.

PROPOSITION 3.3
A permutation $\pi : D \to D$ is narrow iff there is a positive constant $k \in \mathbb{Z}^+$, such that for all $x \in D$ we have $\pi^k(x) = x$. In other words, if $\pi^k = \iota$.

PROOF. One direction is trivial. For the second, if π's orbits are bounded by k, we have $\pi^{k!}(x) = x$ for every $x \in D$. ∎

THEOREM 3.4
For every encoding $\rho : D \to D$, there are models A and B, such that $A \gtrsim_\rho B \supsetneq A$, iff ρ is not a narrow permutation.

PROOF. Suppose π is a narrow permutation with orbit size bounded by k, and assume $A \gtrsim_\pi B \supseteq A$. For every function $g \in B$, there is, by assumption, some function $f_1 \in A$, such that $\pi^{-1} \circ f_1 \circ \pi = g$. Since f_1 is also in B, there is, by k-fold repetition, a function $f_k \in A$, such that $f_k = \pi^{-k} \circ f_k \circ \pi^k = g$. Therefore, $B = A$.

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

Case (i). Suppose ρ is non-surjective, and let $c \in D \setminus \operatorname{rng} \rho$. Define $B = \{\lambda x.\rho^i(c) : i \in \mathbb{N}\}$ and $A = B \setminus \{\lambda x.c\}$. Since $c \notin \operatorname{rng} \rho$, it follows that $A \subsetneq B$. Since for all i we have that $\rho^{-1} \circ \lambda x.\rho^{i+1}(c) \circ \rho = \lambda x.\rho^i(c)$, it follows that $A \succsim_\rho B$.

Case (ii). Suppose that ρ is surjective, but not injective, and let $c \in D$ be such that $\rho(a) = \rho(b) = c$, for some $a \neq b$ in D. Since ρ is a (single-valued) function, it follows that at least one of a and b, say a, is not in $\{\rho^i(c) : i \in \mathbb{N}\}$. So, let $B = \{\lambda x.\rho^i(a) : i \in \mathbb{N}\}$ and $A = B \setminus \{\lambda x.a\}$. By the same argument as in case (i), we have $A \succsim_\rho B \supsetneq A$.

Case (iii). Suppose that ρ is an unbounded-orbit permutation. Let σ be a function that chooses a representative within each orbit: for all $x, y \in D$, $\sigma(x) = \sigma(y)$ iff $\rho^i(x) = y$ for some $i \in \mathbb{Z}$. Define $B = \{\lambda x.\rho^i(\sigma(x)) : i \in \mathbb{N}\}$ and $A = B \setminus \{\lambda x.\sigma(x)\}$. Since the orbits of ρ are unbounded, it must be that $A \subsetneq B$. By the argument of case (i), we again have $A \succsim_\rho B$. ∎

Corollary 3.5
There are models isomorphic to strict supermodels of themselves.

Proof. Let π be a non-narrow permutation of some domain D. By the above theorem, there are models A and B such that $A \succsim_\pi B \supsetneq A$, and (by Theorem 2.10) $\pi\langle A \rangle \supseteq B$. Since $A \equiv \pi\langle A \rangle$ (Corollary 2.21), it follows that A is isomorphic to a strict supermodel of itself, viz. $\pi\langle A \rangle$. ∎

We provide next an example of a specific computational model, consisting of computable functions, that is isomorphic to a strict supermodel of itself via a computable permutation.

Example 3.6
Let K be a set of "basic functions" over $\mathbb{N}$, containing all the constant functions κ_k ($\lambda n.k$), plus the identity, ι. We present two models, A and B, both containing the basic functions and closed under function composition, such that the smaller one (A) simulates every function of the infinitely larger one (B).

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:

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

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

$$f_{i,j}(n) \quad = \quad g_i(n) + j \bmod (g_{i+1}(n) - g_i(n)) \ .$$

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:

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

Thus, B has functions to jump anywhere in subsequent rows, while $A \subsetneq B$ 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 $\mathbb{N}$, such that $A \gtrsim_\pi B$, namely:

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

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

$$\pi = \{(0),\ (1\,2\,3),\ (4\,5\,6\,7\,8),\ \ldots\}.$$

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

$$\begin{aligned} \pi(\iota) &= \iota \in K \subseteq A \\ \pi(\kappa_k) &= \kappa_{\pi(k)} \in K \subseteq A \\ \pi(f_{i,j}) &= f_{i,j+1} \in F \subseteq A, \text{ for } i > 0,\ j \geq 0\,. \end{aligned}$$

THEOREM 3.7
The primitive recursive functions (Prim) are strictly weaker than the recursive functions.

PROOF. Clearly, Rec $\gtrsim_\iota$ Prim. So, assume, on the contrary, that Prim $\gtrsim_\rho$ Rec, for some 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 $\rho(S(n)) = S'(\rho(n))$, ρ is primitive 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 ρ is a recursive injection and rng $h \subseteq$ rng ρ, it follows that $t = \rho^{-1} \circ h \in$ Rec. Presumably, then, 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, $h \notin$ Prim. ∎

4 Completeness

As shown in the previous section, a model can be of equivalent power to its strict supermodel. There are, however, models that are not susceptible to such an anomaly.

Definition 4.1 (Complete)
A model is *complete* if it is not of equivalent power to any of its strict supermodels. That is, A is complete if $A \gtrsim B \supseteq A$ implies $A = B$ for all B.

Completeness gives the converse of Proposition 2.19:

Theorem 4.2
If a model is complete, then all strongly equivalent models are isomorphic.

Proof. Let A be a complete model and assume $A \gtrsim_\pi B \gtrsim_\tau A$ for model B and bijections π, τ. By Theorem 2.10, $\pi\langle A\rangle \supseteq B$ and $\tau\langle B\rangle \supseteq A$. Were $\pi\langle A\rangle \supsetneq B$, then, by Lemma 2.22, $\tau\langle\pi\langle A\rangle\rangle \supsetneq \tau\langle B\rangle \supseteq A$, which would contradict the completeness of A. Thus, $B = \pi\langle A\rangle$, and, therefore, $A = \pi^{-1}\langle B\rangle$. ■

The formulation of this result can be strengthened somewhat:

Lemma 4.3
If model A is complete and $A \gtrsim_\rho B \gtrsim_\pi A$, for model B, injection ρ and bijection π, then A and B are isomorphic.

Proof. Suppose A is complete, and $A \gtrsim_\rho B \gtrsim_\pi A$ for injection ρ and bijection π. It follows that $\pi\langle B\rangle = A' \supseteq A$. Thus, $A \gtrsim_\rho B \gtrsim_\pi A' \supseteq A$. Therefore, from the completeness of A, it follows that $A' = A$. Hence, $A = A' \gtrsim_{\pi^{-1}} B$, and $A \simeq B$. By the previous theorem, $A \equiv B$. ■

Theorem 4.4
If a model is complete, then all strongly equivalent models are also complete.

Proof. Suppose that A is complete and $A \simeq B \subsetneq C$, for models B, C. By Lemma 2.23, $C \simeq D \supsetneq A$ for some D. Were $B \gtrsim C$, then $A \simeq B \gtrsim C \simeq D$, contradicting the completeness of A. Hence, B is also complete. ■

Theorem 4.5
If model A is complete and $A \simeq B \subsetneq C$, for models B and 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 complete, then so is B; hence $B \not\gtrsim C$ and also $A \not\gtrsim C$. Hence, $C \succ A$. ■

We turn now to specific computational models.

Definition 4.6 (Hypercomputational Model)
Model H is *hypercomputational* if there is an injective encoding ρ, such that $\rho\langle H\rangle \supsetneq \mathsf{Rec}$.

Theorem 4.7
The recursive functions (Rec) are complete. That is, they cannot simulate any hypercomputational model.

Proof. Assume $\mathsf{Rec} \gtrsim_\rho H \supseteq \mathsf{Rec}$, for some H, and let $S \in H$ be the successor function. Analogous to the proof of Theorem 3.7, $\rho \in \mathsf{Rec}$ and ρ^{-1} is partial recursive (in PR). For every $h \in H$, there is an $f \in \mathsf{Rec}$, such that $h = \rho^{-1} \circ f \circ \rho$; thus, by the closure of PR under function composition, $h \in \mathsf{PR}$. Actually, h is total, since $\operatorname{rng}(f \circ \rho) = \operatorname{rng}(\rho \circ h) \subseteq \operatorname{rng} \rho$. Therefore, every $h \in H$ is recursive; hence $H = \mathsf{Rec}$. ∎

By the same token:

THEOREM 4.8
The partial recursive functions (PR) are complete.

COROLLARY 4.9
The general recursive functions (Rec) and partial recursive functions (PR) are not strongly equivalent to any of their strict submodels or strict supermodels.

PROOF. Not being strongly-equivalent to strict supermodels is just Theorems 4.7 and 4.8. Non-equivalence to strict submodels follows from Lemma 2.22. ∎

By the above theorems, we can provide a means of showing that a model is hypercomputational:

COROLLARY 4.10
A model H, operating over a denumerable domain, is hypercomputational if any one of the following conditions is satisfied:

1. $H \supsetneq \mathsf{Rec}$.
2. $H \succ \mathsf{Rec}$.
3. $H \succsim_\rho K \supsetneq \mathsf{Rec}$ for some model K and injection ρ.
4. $H \supsetneq K \succsim_\pi \mathsf{Rec}$ for some model K and bijection π.

This justifies the use of the standard comparison method (Section 1.1) in the particular case of the recursive functions.

THEOREM 4.11
Turing machines (TM), over a binary alphabet, and the partial recursive functions (PR) are isomorphic.

PROOF. Since PR is complete, it is sufficient, by Lemma 4.3, to show that $\mathsf{PR} \succsim_\rho \mathsf{TM} \succsim_\pi \mathsf{PR}$, for some injection ρ and bijection π. Since it is well-known that $\mathsf{PR} \succsim \mathsf{TM}$ via Gödelization, it remains to show that $\mathsf{TM} \succsim_\pi \mathsf{PR}$, for some bijection π. Define (as in [7, p. 131]) the bijection $\pi : \mathbb{N} \to \{0,1\}^*$, by

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

For example, $\pi(i)$ is $\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$ for $i = 0, 1, 2, \ldots$.

As per [7, pp. 131–133], $\mathsf{TM} \succsim_\pi \mathsf{RAM}$ (Random Access Machine); $\mathsf{RAM} \supseteq \mathsf{CM}$ (Counter Machine) by [7, pp. 116–118]; and $\mathsf{CM} \succsim_\iota \mathsf{PR}$ by [7, pp. 207–208]. We have that $\mathsf{PR} \succsim \mathsf{TM} \succsim_\pi \mathsf{RAM} \succsim_\iota \mathsf{CM} \succsim_\iota \mathsf{PR}$; thus, $\mathsf{TM} \equiv \mathsf{PR}$. (The exact definitions of RAM and CM are immaterial, as they are only intermediaries in $\mathsf{TM} \succsim_\pi \mathsf{RAM} \succsim_\iota \mathsf{CM} \succsim_\iota \mathsf{PR}$.) ∎

Theorem 4.12
Turing machines (TM) are complete.

Proof. By Theorem 4.11, TM $\simeq$ PR. Since PR is complete, it follows, by Theorem 4.4, that TM is complete. ∎

Conjecture 4.13
The lambda calculus (Λ) is incomplete.

This is because we believe that the lambda calculus cannot identify Church numerals directly. Since Turing machines can—by viewing lambda terms as strings, and the lambda calculus simulates Turing machines, it would follow that the lambda calculus is incomplete.

We do not know if the primitive recursive functions (Prim) are complete, Theorem 3.7 notwithstanding.

5 Discussion

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

Inductive Domains. The completeness of the (general and partial) recursive functions is due to several properties, among which is the inclusion of a successor function (Theorem 4.7). The results herein can be extended to show that computational models operating over other inductively-defined domains are also complete.

Intensional Properties of Completeness. Intuitively, a properly defined computational model should be complete. What is, however, "properly defined"? One can look for the intensional properties of a model that guarantee completeness. That is, what internal definitions that constitute a model (e.g. a finite set of instructions, over a finite alphabet, ...) guarantee completeness.

Different Domain and Range. The simulation definition (Definition 2.4) 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 5.1
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

$$\begin{aligned} h(n) &= \begin{cases} 0 & \text{program } n \text{ halts uniformly} \\ 1 & \text{otherwise} \end{cases} \\ h_i(n) &= \begin{cases} 0 & n < i \vee h(n) = 0 \\ \perp & \text{otherwise .} \end{cases} \end{aligned}$$

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

$$\begin{aligned} \rho(n) &= 2n + h(n) \\ h'_i(n) &= \begin{cases} 0 & \lfloor n/2 \rfloor < i \vee n \bmod 2 = 0 \\ \perp & \text{otherwise} \end{cases} \end{aligned}$$

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

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

Firm comparison. Comparison by an injective mapping between domains might be too permissive, as shown in Example 5.1 above. Accordingly, one may add other constraints on top of the mapping. For example, adding the requirement that the "stronger" model can distinguish the range of the mapping. That is, requiring a total function in the "stronger" model, whose range is exactly the range of the comparison mapping. This approach is further developed in [1].

Multivalued Representations. It may be useful to allow several encodings of the same element, so 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, e.g., [14, p. 33].) To extend the notion of computational power (Definition 2.6) to handle multivalued representations, we would say that model $A \succsim B$ if there is a partial surjective function $\eta : \mathrm{dom}\ A \to \mathrm{dom}\ B$ ($\eta(y) = \perp$ 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 $x \in \mathrm{dom}\ \eta$. This follows along the lines suggested in [14, pp. 52–53]. The corresponding definitions and results need to be extended accordingly.

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 $n \in \mathbb{N}$.

Effectivity. A different approach to comparing models over different domains is to require some manner of effectiveness of the encoding; see [5, p. 21] and [6, p. 290], for example. Two basic methods are usually applied for effective encoding:

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.

Nondeterministic Models. The computational models we have investigated are deterministic (Definition 2.1). The corresponding definitions and results should be extended to nondeterministic models, as well.

Acknowledgements

We thank Arnon Avron, Yaacov Choueka, and Mayer Goldberg for their encouragement and helpful comments.

References

[1] Udi Boker and Nachum Dershowitz. A hypercomputational alien. *Journal of Applied Mathematics & Computing*, to appear.

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

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

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

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

[6] Fred Hennie. *Introduction to Computability.* Addison-Wesley, Reading, MA, 1977.

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

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

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

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

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

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

[13] Alan 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] Klaus Weihrauch. *Computable Analysis — An Introduction.* Springer-Verlag, Berlin, 2000.

Received 27 June 2004