Three Paths to Effectiveness

Udi Boker[1,⋆] and Nachum Dershowitz[2,⋆⋆]

[1] School of Engineering and Computer Science, Hebrew University, Jerusalem 91904, Israel
udiboker@cs.huji.ac.il
[2] School of Computer Science, Tel Aviv University, Ramat Aviv 69978, Israel
nachum.dershowitz@cs.tau.ac.il

For Yuri, profound thinker, esteemed expositor, and treasured friend.

Abstract. Over the past two decades, Gurevich and his colleagues have developed axiomatic foundations for the notion of *algorithm*, be it classical, interactive, or parallel, and formalized them in a new framework of *abstract state machines*. Recently, this approach was extended to suggest axiomatic foundations for the notion of *effective computation* over arbitrary countable domains. This was accomplished in three different ways, leading to three, seemingly disparate, notions of effectiveness. We show that, though having taken different routes, they all actually lead to precisely the same concept. With this concept of effectiveness, we establish that there is – up to isomorphism – exactly one maximal effective model across all countable domains.

Keywords: ASM, effectiveness, recursive functions, Turing machines, computability, constructiveness

1 Introduction

Church's Thesis asserts that the recursive functions are the only numeric functions that can be effectively computed. Similarly, Turing's Thesis stakes the claim that any function on strings that can be mechanically computed can be computed, in particular, by a Turing machine. For models of computation that operate over arbitrary data structures, however, these two standard notions of what constitutes effectiveness may not be directly applicable; as Richard Montague asserts [9, pp. 430–431]:

> Now Turing's notion of computability applies directly only to functions on and to the set of natural numbers. Even its extension to functions defined on (and with values in) another denumerable set S cannot be accomplished in a completely unobjectionable way. One would be inclined to choose a one-to-one correspondence between S and the set of natural numbers, and to call a function f on S computable if the function of natural numbers induced by f under this correspondence is computable

⋆ Supported in part by a Lady Davis postdoctoral fellowship.
⋆⋆ Supported in part by the Israel Science Foundation (grant no. 250/05).

> in Turing's sense. But the notion so obtained depends on what correspondence between S and the set of natural numbers is chosen; the sets of computable functions on S correlated with two such correspondences will in general differ. The natural procedure is to restrict consideration to those correspondences which are in some sense 'effective', and hence to characterize a computable function on S as a function f such that, for some effective correspondence between S and the set of natural numbers, the function induced by f under this correspondence is computable in Turing's sense. But the notion of effectiveness remains to be analyzed, and would indeed seem to coincide with computability.

One may ask, for example: What are the computable functions over the algebraic numbers? Does one obtain different sets of computable functions depending on which representation ("correspondence") one chooses for them?

Before we can answer such questions, we need a most-general notion of algorithm. Sequential algorithms – that is, deterministic algorithms without unbounded parallelism or (intra-step) interaction with the outside world – have been analyzed and formalized by Gurevich in [6]. There it was proved that any algorithm satisfying three natural formal postulates (given below) can be emulated, step by step, by a program in a very general model of computation, called "abstract state machines" (ASMs). This formalization was recently extended in [1] to handle partial functions. But an algorithm, or abstract state machine program, need not yield an effective function. Gaussian elimination, for example, is a perfectly well-defined algorithm over the real numbers, even though the reals cannot all be effectively represented and manipulated.

We adopt the necessary point of view that effectiveness is a notion applicable to collections of functions, rather than to single functions (cf. [10]). A single function over an arbitrary domain cannot be classified as effective or ineffective [9,14], since its effectiveness depends on the context. A detailed discussion of this issue can be found in [3].

To capture what it is that makes a sequential algorithm mechanically computable, three different generic formalizations of effectiveness have recently been suggested:

- In [3], the authors base their notion of effectivity on finite constructibility. Initial data are inductively defined to be effective if they only contain a Herbrand universe, in addition to finite data and functions that can be shown constructible in the same way.
- In [5], Dershowitz and Gurevich require an injective mapping between the arbitrary domain and the natural numbers. Initial data are effective if they are tracked – under that representation – by recursive functions, as in the traditional definition of "computable" algebras [15].
- In [12], Reisig bases effectiveness on the natural congruence relation between vocabulary terms that arises in the theory of ASMs. Initial data are effective if the induced congruence between terms is Turing-computable.

Properly extending these approaches to handle partial functions, and to refer to a set of algorithms, it turns out that these three notions are essentially one and the same.

2 Algorithms

We work within the abstract-state-machine framework of [6], modified to make terminal states explicit and to allow partial operation to "hang", as in [1]. We begin by recalling Gurevich's Sequential Postulates, formalizing the following intuitions: (I) we are dealing with discrete deterministic state-transition systems; (II) the information in states suffices to determine future transitions and may be captured by logical structures that respect isomorphisms; and (III) transitions are governed by the values of a finite and input-independent set of (variable-free) terms. See [5] for historical support for these postulates.

Postulate I (Sequential Time). *An algorithm determines the following:*

1. *A nonempty set $\mathcal{S}$ of* states *and a nonempty subset $\mathcal{S}_0 \subseteq \mathcal{S}$ of* initial states.
2. *A partial* next-state *transition function $\tau : \mathcal{S} \rightharpoonup \mathcal{S}$.*

A *terminal state* is one for which no transition is defined. Let $\mathcal{O} \subseteq \mathcal{S}$ denote the (possibly empty) set of terminal states. We write $x \rightsquigarrow_\tau x'$ when $x' = \tau(x)$. A *computation* is a finite or infinite chain $x_0 \rightsquigarrow_\tau x_1 \rightsquigarrow_\tau \cdots$ of states.

Since transitions are functions, the states of an algorithm must contain all the information necessary to determine the future of a computation, a full "instantaneous description" of all relevant aspects of the computation's current status.

(It may appear that a recursive function is not a state-transition system, but in fact the definition of a recursive function comes together with a computation rule for evaluating it. As Rogers [13, p. 7] writes, for instance, "We obtain the computation uniquely by working from the inside out and from left to right".)

Logical structures are ideal for capturing all the salient information stored in a state. All structures in this paper are over first-order finite vocabularies, have countably many elements in their domains (base sets), and interpret symbols as partial operations. All relations are viewed as truth-valued functions, so we refer to structures as (partial) algebras (with partial functions). We assume that structures include Boolean truth values, standard Boolean operations, and that vocabularies include symbols for these.

Postulate II (Abstract State). *The states $\mathcal{S}$ of an algorithm are partial algebras over a finite vocabulary $\mathcal{F}$, such that the following hold:*

1. *If $x \in \mathcal{S}$ is a state of the algorithm, then any algebra y isomorphic to x is also a state in $\mathcal{S}$, and y is initial or terminal if x is initial or terminal, respectively.*
2. *Transitions τ preserve the domain; that is,* Dom $\tau(x)$ = Dom x *for every non-terminal state $x \in \mathcal{S} \setminus \mathcal{O}$.*
3. *Transitions respect isomorphisms, so, if $\zeta : x \cong y$ is an isomorphism of non-terminal states $x, y \in \mathcal{S} \setminus \mathcal{O}$, then $\zeta : \tau(x) \cong \tau(y)$.*

Such states are "abstract", because the isomorphism requirement means that transitions do not depend in any essential way on the specific representation of the domain embodied in a given state.

Since a state x is an algebra, it interprets function symbols in $\mathcal{F}$, assigning a value $c \in \text{Dom } x$ to the "location" $f(a_1, \ldots, a_k)$ in x for every k-ary symbol $f \in \mathcal{F}$ and values $a_1, \ldots, a_k$ in Dom x. For location $\ell = f(a_1, \ldots, a_k)$, we write $[\![\ell]\!]_x$ for the value $f^x(a_1, \ldots, a_k)$ that x assigns to ℓ. Similarly, for term t, $[\![t]\!]_x$ is its value under the interpretations given to all the symbols in t by x. If the interpretation of any subterm is undefined, then so is the whole term. We use $\perp$ to denote an undefined value for a location or term. All terms in this paper are ground terms, that is, terms without variables.

We shall assume that all elements of the domain are accessible via terms in initial states (or else the superfluous elements may be removed with no ill effect). But note that a transition may cause accessible elements to become inaccessible, as explained in [12].

It is convenient to view each state as a collection of the graphs of its operations, given in the form of a set of location-value pairs, each written conventionally as $f(\bar{a}) \mapsto c$, for $\bar{a} \in \text{Dom } x$, $c \in \text{Dom } x$. Define the *update set* $\Delta(x)$ of state x as the changed location-value pairs, $\tau(x) \setminus x$. When x is a terminal state and $\tau(x)$ is undefined, we will indicate that by setting $\Delta(x) = \perp$.

The transition function of an algorithm must be describable in a finite fashion, so its description can only refer to finitely many locations in the state by means of finitely many terms over its vocabulary.

Postulate III (Effective Transition). *An algorithm with states $\mathcal{S}$ over vocabulary $\mathcal{F}$ determines a finite set T of* critical terms *over $\mathcal{F}$, such that states that agree on the values of the terms in T also share the same update sets. That is,*

$$\text{if } x =_T y \text{ then } \Delta(x) = \Delta(y) \ ,$$

for any two states $x, y \in \mathcal{S}$.

Here, $x =_T y$, for a set of terms T, means that $[\![t]\!]_x = [\![t]\!]_y$ for all $t \in T$.

Whenever we refer to an "algorithm" below, we mean an object satisfying the above three postulates, what we like to call a "classical algorithm".

Definition 1. *An algorithm A with states $\mathcal{S}$* computes *a partial function $f : D^k \rightharpoonup D$ if there is a subset $\mathcal{I}$ of its initial states, with locations for input values, such that running the algorithm yields the correct output values of f. Specificially:*

1. *The domain of each state in $\mathcal{I}$ is D.*
2. *There are k distinct locations $\ell_1, \ldots, \ell_k$ such that all possible input values are covered. That is, $\{\langle [\![\ell_1]\!]_x, \ldots, [\![\ell_k]\!]_x \rangle : x \in \mathcal{I}\} = D^k$.*
3. *All states in $\mathcal{I}$ agree on the values of all locations other than $\ell_1, \ldots, \ell_k$.*
4. *There is a term t (in the vocabulary of the algorithm) such that for all $a_0, \ldots, a_k \in D$, if $f(a_1, \ldots, a_k) = c$, then there is some initial state $x_0 \in \mathcal{I}$, with $[\![\ell_j]\!]_{x_0} = a_j$ $(j = 1, \ldots, k)$, initiating a terminating computation $x_0 \rightsquigarrow_\tau \cdots \rightsquigarrow_\tau x_n$, where $x_n \in \mathcal{O}$ and such that $[\![t]\!]_{x_n} = c$.*
5. *Whenever $f(a_1, \ldots, a_k)$ is $\perp$, there is an initial state $x_0 \in \mathcal{I}$, with $[\![\ell_j]\!]_{x_0} = a_j$ $(j = 1, \ldots, k)$, initiating an infinite computation $x_0 \rightsquigarrow_\tau x_1 \rightsquigarrow_\tau \cdots$.*

A (finite or infinite) set of algorithms, all with the same domain, will be called a *model (of computation)*.

3 Effective Models

We turn now to examine the three different approaches to understanding effectiveness. Informally, they each add a postulate along the following lines:

Postulate IV (Effective Initial State). *The initial states $\mathcal{S}_0$ of an* effective *algorithm are finitely representable.*

3.1 Distinguishing Models

Every state x induces a congruence on all terms, under which terms are congruent whenever the state assigns them the same value:

$$s \simeq_x t \;\Leftrightarrow\; [\![s]\!]_x = [\![t]\!]_x \;.$$

Isomorphic states clearly induce the same congruence.

We call a state "distinguishing" if its induced congruence is semi-decidable in the standard sense. That is, a state is distinguishing if there is a Turing machine (or a similar device, as a partial recursive function operating on strings) that can act as "state manager", receiving two terms as input and returning true whenever both terms are defined and congruent, false when both are defined but not congruent, and diverging otherwise. This is the effectiveness notion explored in [12] (which, however, considers only a single state and total functions).

Definition 2 (Distinguishing Model).
- *A state is* distinguishing *if its induced congruence is semi-decidable.*
- *An algorithm is* distinguishing *if all its initial states are.*
- *A model is* distinguishing *if every congruence induced by a finite set of initial states (across different algorithms) is semi-decidable.*

3.2 Computable Models

We say that an algebra $\mathcal{A}$ over (a possibly infinite) vocabulary $\mathcal{F}$ with domain D *simulates* an algebra $\mathcal{B}$ over (a possibly infinite) vocabulary $\mathcal{G}$ with domain E if there exists an injective "encoding" $\rho : E \rightharpoonup D$ such that for every partial function $f : E^k \rightharpoonup E$ of $\mathcal{B}$ there is a partial function $\widehat{f} : D^k \rightharpoonup D$ of $\mathcal{A}$, such that $f(x)$ is defined exactly when $(\widehat{f} \circ \rho)(x)$ is defined, for every $x \in E$, and that $f(x) = (\rho^{-1} \circ \widehat{f} \circ \rho)(x)$ whenever $f(x)$ is defined. In that case, we say that $\widehat{f}$ *tracks* f under ρ.

Definition 3 (Computable Model).
- *A state is* computable *if it is simulated by the partial recursive functions.*
- *An algorithm is* computable *if all its initial states are.*

– *A model is* computable *if all its algorithms are, via the same encoding.*

This is a standard notion of "computable algebra" [8,11,7,15], adopted by [5] (which, however, considers only a single algorithm and total functions).

The choice of the partial recursive functions as the starting point for defining effective algorithms over arbitrary domains is natural, considering the Church-Turing Thesis. The main question that one may raise is whether the allowance of an injective representation between the arbitrary domain and the natural numbers is sensible. We show, in the following lemma, that as long as all domain elements are reachable by ground terms, the required injective representation implies the existence of a bijection between the domain and the natural numbers. Hence, the initial functions of a computable algorithm are isomorphic to some partial-recursive functions, which makes their effectiveness hard to dispute.

Lemma 1. *Let M be a computable model over domain D. Then there is a bijection $\pi : D \leftrightarrow \mathbb{N}$ such that, for each partial function $f : D^k \rightharpoonup D$ of each initial state of each algorithm in M, there is a partial recursive function $\tilde{f} : \mathbb{N}^k \rightharpoonup \mathbb{N}$ that tracks f under π.*

Proof. Let ρ be the injective representation from D to $\mathbb{N}$, via which all initial functions of M are partial recursive. For each initial function f, we denote its partial recursive counterpart by $\widehat{f}$. That is, $f = \rho^{-1} \circ \widehat{f} \circ \rho$.

Consider one specific algorithm A of M with vocabulary $\mathcal{F}$. By definition, all elements of D are reachable in each initial state by terms of $\mathcal{F}$. Therefore, all elements of Im $\rho \subseteq \mathbb{N}$ are reachable by terms of $\mathcal{F}$, as interpreted by the partial-recursive tracking fiunctions.

Let $\{c_j\}_j$ be a computable enumeration of all terms over $\mathcal{F}$. One can construct a computable enumeration of all the $\mathcal{F}$-terms that are *defined* in the tracking interpretations by interleaving the computations of the $\{c_j\}_j$ terms in the standard (Cantor's) zigzag fashion (a computation step for the first term, followed by one for the second term and two for the first, then one for the third, two for the second and three for the first, etc.). Accordingly, an enumeration $\{d_j\}_j$ can be set up, assigning a unique $\mathcal{F}$-term for every number in Im ρ, by enumerating the defined $\mathcal{F}$-terms, as above, and ignoring those terms that evaluate to a number already obtained.

In this fashion, we can define a recursive bijection $\eta :$ Im $\rho \leftrightarrow \mathbb{N}$, letting $\eta(n)$ be the unique j such that $[\![d_j]\!] = n$. Note that η^{-1} is also recursive, as $\eta^{-1}(m) = [\![d_m]\!]$.

Let $\pi : D \leftrightarrow \mathbb{N}$ be the bijection $\pi = \eta \circ \rho$, and, for each function $\widehat{f} \in \widehat{\mathcal{F}}$, define the partial recursive function $\tilde{f}$ to be $\eta \circ \widehat{f} \circ \eta^{-1}$. Then, for each function $f : D^k \rightharpoonup D$ of each initial state of each algorithm in M,

$$f = \rho^{-1} \circ \widehat{f} \circ \rho = \rho^{-1} \circ \eta^{-1} \circ \eta \circ \widehat{f} \circ \eta^{-1} \circ \eta \circ \rho = \pi^{-1} \circ \eta \circ \widehat{f} \circ \eta^{-1} \circ \pi = \pi^{-1} \circ \tilde{f} \circ \pi \ .$$

□

3.3 Constructive Models

Let x be an algebra over vocabulary $\mathcal{F}$, with domain D. A finite vocabulary $\mathcal{C} \subseteq \mathcal{F}$ *constructs* D if x assigns each value in D to exactly one term over C.

Definition 4 (Constructive Model).
- *A state is* constructive *if it includes constructors for its domain, plus operations that are almost everywhere "blank", meaning that all but finitely-many locations have the same default value (say* undef*).*
- *An algorithm is* constructive *if its initial states are.*
- *A model is* constructive *if all its algorithms are, via the same constructors.*

Moreover, constructive algorithms can be bootstrapped: Any state over vocabulary $\mathcal{F}$ with domain D is constructive if $\mathcal{F}$ can be extended to $\mathcal{C} \uplus \mathcal{G}$ so that $\mathcal{C}$ constructs D and every $g \in \mathcal{G}$ has a constructive algorithm over $\mathcal{C}$ that computes it.

This is the approach advocated in [3] (which, however, considers only total functions).

As expected:

Theorem 1 ([3, Thm. 3]). *The partial recursive functions form a constructive model.*

Conversely:

Theorem 2 (cf. [3, Thm. 4]). *Every constructive model can be simulated by the partial recursive functions via a bijective encoding.*

Though this theorem in [3] does not speak of a bijective encoding, its proof therein in point of fact uses a bijective encoding. That proof refers only to total functions; however, partial functions can be handled similarly. Also, one needs to show, inductively, that constructive initial functions used in bootstrapped constructive algorithms are tracked by partial recursive functions.

4 Equivalence of Definitions of Effectiveness

The following equivalence is demonstrated in the remainder of this section.

Theorem 3. *A model of computation is computable if and only if it is constructive if and only if it is distinguishing.*

Returning to the example of algebraic numbers, this means that one obtains the same set of effectively-computable (partial) functions over the domain of algebraic numbers, regardless of which definition of effectiveness one adopts. In particular, the effective partial functions over algebraic numbers – obtained in any of these ways – are isomorphic to the partial recursive functions over the natural numbers, and, by results in [2], no representation can yield additional functions.

4.1 Computable and Distinguishing

Theorem 3a. *Computable models are distinguishing.*

Proof. Given a computable model M over domain D, there is, by definition, an injective representation $\rho : D \rightharpoonup \mathbb{N}$ such that, for every function f in an initial state of one of the algorithms in M, there is a partial recursive function $\widehat{f}$ over $\mathbb{N}$ that tracks f under ρ. Since every finite subset M' of algorithms in M can have only finitely many initial functions, it follows that a single partial recursive function can check for equality of the values of two terms in the initial states of any such M' by using partial recursive implementations of the $\widehat{f}$'s to compute the numerical counterparts of the terms and – if and when that computation terminates – testing equality of the resultant numerals. Hence, M is distinguishing. □

Theorem 3b. *Distinguishing models are computable.*

Proof. Let M be a distinguishing model over domain D and consider some specific algorithm $A \in M$ with vocabulary $\mathcal{F}$. Let $\{t_i\}_i$ be any recursive enumeration of terms over $\mathcal{F}$, and let $\simeq$ be the partial recursive function that semi-decides the congruence relation of terms in A. One can define a recursive enumeration $\{c_j\}_j$ of all $\mathcal{F}$-terms that are defined in M by interleaving the computations of $t_i \simeq t_i$ for all terms t_i, in the standard zigzag fashion. A term t_i is added to the enumeration once the corresponding congruence computation ends.

Define the injective representation $\rho : D \rightharpoonup \mathbb{N}$ by

$$\rho(x) := \min_j \{[\![c_j]\!] = x\} \ .$$

For any initial function f of A, define the partial recursive function

$$\widehat{f}(n) := \min_i c_i \simeq f(c_n) \ ,$$

where $f(c_n)$ is the term obtained by enclosing the term c_n with the symbol f. These numerical $\widehat{f}$ track their original counterparts f over D, as follows:

$$\begin{aligned}
\widehat{f}(\rho(x)) &= \widehat{f}(\min_j \{[\![c_j]\!] = x\}) \\
&= \min_i \{c_i \simeq f(c_k)\} \text{ where } k = \min_j \{[\![c_j]\!] = x\} \\
&= \min_i \{[\![c_i]\!] = [\![f(c_k)]\!]\} \\
&= \min_i \{[\![c_i]\!] = f([\![c_k]\!])\} \\
&= \min_i \{[\![c_i]\!] = f(x)\} \\
&= \rho(f(x)) \ .
\end{aligned}$$

Similarly for operators of other arities.

It is left to show how the specific injective representation from D to $\mathbb{N}$, which was defined according to one algorithm of M, suits all other algorithms

of M. Consider any algorithm $B \in M$ with vocabulary $\mathcal{F}'$ and let $\{c'_j\}_j$ be some recursive enumeration of all defined terms over $\mathcal{F}'$. By the definition of distinguishing, there is a partial recursive function semi-deciding the equivalence of any two terms of the algorithms A and B. This allows one to translate between the term enumerations of A and B and also have partial recursive functions that track the initial functions of B. For any initial function g of B, define the partial recursive function

$$\widehat{g}(n) := \min_i \left\{ c_i \simeq g\left(c'_{\min_\ell\{c'_\ell \simeq c_n\}}\right)\right\} \ .$$

These numerical $\widehat{g}$ track their original counterparts g in B, as follows:

$$\begin{aligned}
\widehat{g}(\rho(x)) &= \widehat{g}(\min_j\{[\![c_j]\!] = x\}) \\
&= \min_i \{c_i \simeq g(c'_k)\} \text{ where } k = \min_\ell \left\{c'_\ell \simeq c_{\min_j\{[\![c_j]\!]=x\}}\right\} \\
&\qquad\qquad\qquad\qquad = \min_\ell \left\{[\![c'_\ell]\!] = [\![c_{\min_j\{[\![c_j]\!]=x\}}]\!]\right\} \\
&\qquad\qquad\qquad\qquad = \min_\ell\{[\![c'_\ell]\!] = x\} \\
&= \min_i \{[\![c_i]\!] = [\![g(c'_k)]\!]\} \\
&= \min_i \{[\![c_i]\!] = g([\![c'_k]\!])\} \\
&= \min_i \{[\![c_i]\!] = g(x)\} \\
&= \rho(g(x)) \ .
\end{aligned}$$

Similarly for operators of other arities.

It follows that M is computable under the auspices of ρ. □

4.2 Computable and Constructive

Computability is based on the recursiveness of the initial functions, under an injective representation of the arbitrary domain D as natural numbers $\mathbb{N}$. Furthermore, the requirement that all domain elements are reachable by terms implies that there is also a bijective mapping from D to $\mathbb{N}$ via which the initial functions are partial recursive.

Theorem 3c. *Constructive models are computable.*

Proof. Let M be a constructive model over domain D and let M' consist of algorithms for all the constructive functions and all the almost-everywhere-blank functions in M's initial states. By Theorem 2, the set of functions computed by M' is simulatable by the partial recursive functions via some representation $\rho : D \rightharpoonup \mathbb{N}$. Hence, all initial functions of M are tracked by partial recursive functions, making M computable. □

Theorem 3d. *Computable models are constructive.*

Proof. For any computable model M over domain D, there is, by Lemma 1, a bijection $\pi : D \leftrightarrow \mathbb{N}$ such that every function f in the initial states of M's algorithms is tracked under π by some partial recursive function g. By Theorem 1, there is a constructive model that computes all the partial recursive functions $\mathbb{P}$ over $\mathbb{N}$. Since algorithms (according to Postulate II) are closed under isomorphism, so are constructive models. Hence, there is a constructive model P over $\pi^{-1}(\mathbb{N})$, with some set of constructors, that computes all functions $\pi \circ g \circ \pi^{-1}$ that are tracked by functions $g \in \mathbb{P}$, and – in particular – computes all initial functions of M. Since all M's initial functions are constructive, M is constructive. □

Theorem 3 is the conjunction of Theorems 3a–3d.

5 Conclusions

Thanks to Theorem 3, it seems reasonable to just speak of "effectiveness", without distinguishing between the three equivalent notions discussed in the previous sections. Having shown that three prima facie distinct definitions of effectiveness over arbitrary domains comprise exactly the same functions strengthens the impression that the essence of the underlying notion of computability has in fact been captured.

Fixing the concept of an effective model of computation, the question naturally arises as to whether there are "maximal" effective models, and if so, whether they are really different or basically one and the same. Formally, we consider an effective computational model M (consisting of a set of functions) over domain D to be *maximal* if adding any function $f \notin M$ over D to M gives an ineffective model $M \cup \{f\}$. It turns out that there is exactly one effective model (regardless of which of the three definitions one prefers), up to isomorphism.

Theorem 4. *The set of partial recursive functions is the unique maximal effective model, up to isomorphism, over any countable domain.*

Proof. We first note that the partial recursive functions are a maximal effective model. Their effectiveness was established in Theorem 1. As for their maximality, the partial recursive functions are "interpretation-complete", in the sense that they cannot simulate a more inclusive model, as shown in [2,4]. By Theorem 2, they can simulate every effective model, leading to the conclusion that there is no effective model more inclusive than the partial recursive functions.

Next, we show that the partial recursive functions are the unique maximal effective model, up to isomorphism. Consider some maximal effective model M over domain D. By Theorem 2, the partial recursive functions can simulate M via a bijection π. Since effectiveness is closed under isomorphism, it follows that there is an effective model M' over D isomorphic to the partial recursive functions via π^{-1}. Hence, M' contains M, and by the maximality of M we get that $M' = M$. Therefore, M is isomorphic to the partial recursive functions, as claimed. □

The Church-Turing Thesis, properly interpreted for arbitrary countable domains (see [3]), asserts that the partial recursive functions (or Turing machines) constitute the most inclusive effective model, up to isomorphism. However, this claim only speaks about the extensional power of an effective computational model, not about its internal mechanism. Turing, in his seminal work [16], justified the thesis by arguing that every "purely mechanical" human computation can be represented by a Turing machine whose steps more or less correspond to the manual computation. Indeed, Turing's argument convinced most people about the validity of the thesis, which had not been the case with Church's original thesis regarding the effectiveness of the recursive functions (let alone Church's earlier thoughts regarding the lambda calculus). Notwithstanding its wide acceptance, neither the Church-Turing Thesis nor Turing's arguments purport to characterize the internal behavior of an effective computational model over an arbitrary domain.

On the other hand, Gurevich's abstract state machines are the most general descriptive form for (sequential) algorithms known. As such, they can express the precise step-by-step behavior of arbitrary algorithms operating over arbitrary structures, whether for effective computations or for hypothetical ones. The work in [3,5,12] specializes this model by considering effective computations. The additional effectiveness axiom proposed in [3] and adopted in our Definition 4 of constructive models does not rely on the definition of Turing machines or of the partial recursive functions, thereby providing a complete, generic, stand-alone axiomatization of effective *computation* over any countable domain.

References

1. Blass, A., Dershowitz, N., Gurevich, Y.: Exact exploration. Technical Report MSR-TR-2009-99, Microsoft Research, Redmond, WA (2010). Available at `http://research.microsoft.com/pubs/101597/Partial.pdf`. A short version to appear as "Algorithms in a world without full equality" in the Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (Brno, Czech Republic), Lecture Notes in Computer Science. Springer (Aug. 2010)
2. Boker, U., Dershowitz, N.: Comparing computational power. Logic Journal of the IGPL **14** (2006) 633–648
3. Boker, U., Dershowitz, N.: The Church-Turing thesis over arbitrary domains. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.): Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday. Lecture Notes in Computer Science, Vol. 4800. Springer (2008) 199–229
4. Boker, U., Dershowitz, N.: The influence of domain interpretations on computational models. Journal of Applied Mathematics and Computation **215** (2009) 1323–1339
5. Dershowitz, N., Gurevich, Y.: A natural axiomatization of computability and proof of Church's Thesis. Bulletin of Symbolic Logic **14** (2008) 299–350
6. Gurevich, Y.: Sequential abstract state machines capture sequential algorithms. ACM Transactions on Computational Logic **1** (2000) 77–111
7. Lambert, Jr., W.M.: A notion of effectiveness in arbitrary structures. The Journal of Symbolic Logic **33** (1968) 577–602

8. Mal'tsev, A.: Constructive algebras I. Russian Mathematical Surveys **16** (1961) 77–129
9. Montague, R.: Towards a general theory of computability. Synthese **12** (1960) 429–438
10. Myhill, J.: Some philosophical implications of mathematical logic. Three classes of ideas. The Review of Metaphysics **6** (1952) 165–198
11. Rabin, M.O.: Computable algebra, general theory and theory of computable fields. Transactions of the American Mathematical Society **95** (1960) 341–360
12. Reisig, W.: The computable kernel of abstract state machines. Theoretical Computer Science **409** (2008) 126–136
13. Rogers, Jr., H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1966)
14. Shapiro, S.: Acceptable notation. Notre Dame Journal of Formal Logic **23** (1982) 14–20
15. Stoltenberg-Hansen, V., Tucker, J.V.: 4. In: Effective Algebra. Handbook of Logic in Computer Science, Vol. 4. Oxford University Press (1995) 357–526
16. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society **42** (1936–37) 230–265. 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`