A Formalization of the Church-Turing Thesis

Udi Boker and Nachum Dershowitz
School of Computer Science, Tel Aviv University
Tel Aviv 69978, Israel
{udiboker,nachumd}@tau.ac.il

Abstract

Our goal is to formalize the Church-Turing Thesis for a very large class of computational models. Specifically, the notion of an "effective model of computation" over an arbitrary countable domain is axiomatized. This is accomplished by modifying Gurevich's "Abstract State Machine" postulates for state-transition models. A proof is provided that *all* models satisfying these axioms, regardless of underlying data structure—and including all standard models—are equivalent to (up to isomorphism), or weaker than, Turing machines. To allow the comparison of arbitrary models operating over arbitrary domains, we employ a quasi-ordering on computational models, based on their extensionality.

1 Introduction

Motivation. The Church-Turing Thesis is inherently vague, relating to "effective" models in the broad, intuitive, sense. Though generally believed to be true, there are always, and especially recently, efforts to circumvent it, under the banner of "hypercomputation." See, for example, [11, 3, 8, 13]. It is thus important to try to understand what the formal effectiveness constraints are, that every such hypercomputational model must violate.

When defining a specific computational model, one first defines the model's domain and its manner of operation. For example, Turing machines operate over tapes of symbols, the λ-calculus over λ-terms, and counter machines over an n-tuple of natural numbers. In contrast, for axiomatizing effectiveness constraints, one ought to provide general axioms, not relating to a specific domain or operational mechanism.

Main Contribution. We provide four axioms for an "effective computational model." We believe these axioms to be general, both in the sense of capturing basic intuitive notions, and in the sense of allowing a large class of models, among which are all the standard models. A proof is provided that *all* models satisfying these axioms, regardless of underlying data structure, are equivalent to (up to isomorphism), or weaker than, Turing machines.

In order to achieve a wide effectiveness notion, we base our axioms on Gurevich's general definition of a "sequential algorithm" [6]. The models of the resulting class may operate over arbitrary domains, applying arbitrary internal mechanisms. In contrast to any particular model of computation, our axioms apply to models operating over *any* countable domain.

For comparing such a varied class of models with Turing machines, we employ a comparison notion based on the extensionality of models, a variation of the development in [2].

We believe that the axioms provided here may be a good starting point for further formalizations of effectiveness, as they demarcate the largest known class of computational models for which the Church-Turing Thesis is formally established.

Background. In 1936, Alonzo Church and Alan Turing each formulated a claim that a particular model of computation completely captures the conceptual notion of "effective" computability.

After Turing proved that his machines compute exactly the same numeric functions as does the lambda calculus, Kleene [9, p. 232] combined the two claims into one:

> So Turing's and Church's theses are equivalent. We shall usually refer to them both as *Church's thesis*, or in connection with that one of its ... versions which deals with "Turing machines" as *the Church-Turing thesis*.

The claim, then, is the following:

> **Church-Turing Thesis.** *All effective computational models are equivalent to, or weaker than, Turing machines.*

Goal. To formalize this thesis, we need to make precise what is meant by each of the terms: "effective," "computational model," and "weaker or equivalent." As suggested by Shoenfield [12, p. 26]:

> [I]t may seem that it is impossible to give a proof of Church's Thesis. However, this is not necessarily the case.... In other words, we can write down some axioms about computable functions which most people would agree are evidently true. It might be possible to prove Church's Thesis from such axioms.

This is the direction we follow.

In what follows, we *axiomatize* a large class of computational models that includes all known Turing-complete state-transition models, operating over any countable domain, and prove that they are all effectively computable, up to isomorphism of domains.

Approach. We formalize the informal notions of the Church-Turing Thesis as follows:

- Since the comparison is meant to be extensional, we allow a "computational model" to be any set of partial functions over some domain. That is, a (finitary) partial algebra (with—normally—an infinite vocabulary).
- Since we are dealing with mechanisms that may operate on different physical media and employ very different data structures, we adopt a variation of the quasi-ordering on extensional models developed in [2].
- To capture what is intended by "effective," we propose a set of axioms for models, adapting the postulates developed for the "Sequential Algorithm Thesis" of Gurevich.

Effectivity axioms. We understand an "effective computational model" to be some set of "effective procedures" that share the same domain representation. Any "effective procedure" should satisfy four postulates (formally defined as Axioms 1–4 in Sections 2–3):

1. **Sequential Time Axiom.** The procedure can be viewed as a set of states, a specified initial state, and a transition function from state to state.

   This postulate reflects the view of a computation as some transition system, as suggested by Knuth [10, p. 7]:

> [D]efine a *computational method* to be a quadruple (Q, I, Ω, f), in which Q is a set containing subsets I and Ω, and f is a function from Q into itself.... The four quantities Q, I, Ω, f are intended to represent respectively the states of the computation, the input, the output, and the computational rule.

2. **Abstract State Axiom.** Its states are (first-order) structures of the same finite vocabulary. States are closed under isomorphism, and the transition function preserves isomorphism.

   Formalizing the states of the transition system as mathematical structures, as proposed by Gurevich [6, p. 78]:

   > What would their states be? The huge experience of mathematical logic indicates that any kind of static mathematical reality can be faithfully represented as a first-order structure.

3. **Bounded Exploration Axiom.** There is a finite bound on the number of vocabulary-terms that affect the transition function.

   The postulate ensures that the transition system has effective behavior. One way to look at it is [6, p. 82]:

   > We assume informally that any algorithm A can be given by a finite text that explains the algorithm without presupposing any special knowledge.

4. **Initial Data Axiom.** The initial state comprises only finite data in addition to the domain representation; the latter is isomorphic to a Herbrand universe.

   The fourth postulate adds the effectiveness of the initial data [4, p. 195]:

   > [I]f non-computable inputs are permitted, then non-computable outputs are attainable.

   To preclude this we insist that the initial data does not contain more than a finite amount of information. The underlying data type is required to be isomorphic to a Herbrand universe, so that nothing more than equality of data elements is given at the outset of a computation.

The first three postulates axiomatize a "sequential procedure" along the lines of Gurevich's sequential algorithm [6]. The fourth restricts procedures to be wholly effective.

Since all procedures of a specific computational model should have some common mechanism, the minimum requirement is that they share the same domain representation.

Proof sketch. To prove the thesis—under these assumptions—we prove that any procedure operating over an *arbitrary* countable structure, and satisfying the initial-data postulate, can be mapped (using a rigorous notion of mapping) to some **while**-like computer program, which in turn can be mapped to a Turing machine.

Hence, if one defines any model that satisfies the four axioms, this model is necessarily bounded by Turing computability, while if one claims that a model is hypercomputational, it must violate some of the axioms.

Proofs are omitted for lack of space.[1]

[1] They are included in the full version of this paper, at `http://www.cs.tau.ac.il/~udiboker/files/CttFormalizationFull.pdf`.

2 Sequential Procedures

We axiomatize "sequential procedures" along the lines of Gurevich's sequential algorithms [6]. We later axiomatize, in Section 3, "effective procedures" as their subclass. We start by providing the standard definition of mathematical structures, upon which sequential procedures are defined.

2.1 Structures

The states of a procedure should be a full instantaneous description of it. We represent them by (first order) *structures*, using the standard notion of structure from mathematical logic. For convenience, these structures will be *algebras*; that is, having purely functional vocabulary (without relations).

Definition 1 (Structures).

- *A* domain D *is a (nonempty) set of elements.*
- *A* vocabulary $\mathcal{F}$ *is a collection of function names, each with a fixed finite arity.*
- *A* term *of vocabulary $\mathcal{F}$ is either a nullary function name (constant) in $\mathcal{F}$ or takes the form $f(t_1, \ldots, t_k)$, where f is a function name in $\mathcal{F}$ of positive arity k and $t_1, \ldots, t_k$ are terms.*
- *A* structure *S of vocabulary $\mathcal{F}$ is a domain D together with interpretations $[\![f]\!]_S$ over D of the function names $f \in \mathcal{F}$.*
- *A* location *of vocabulary $\mathcal{F}$ over a domain D is a pair, denoted $f(\overline{a})$, where f is a k-ary function name in $\mathcal{F}$ and $\overline{a}$ is a k-tuple of elements of D. (If f is a constant, then $\overline{a}$ is the empty tuple.)*
- *The* value *of a location $f(\overline{a})$ in a structure S, denoted $[\![f(\overline{a})]\!]_S$, is the domain element $[\![f]\!]_S(\overline{a})$.*
- *Structures S and S' with vocabulary $\mathcal{F}$ over the same domain* coincide *over a set T of $\mathcal{F}$-terms if $[\![t]\!]_S = [\![t]\!]_{S'}$ for all terms $t \in T$.*

It is convenient to think of a structure S as a memory, or data-storage, of a kind. For example, for storing an (infinite) two dimensional table of integers, we need a structure S over the domain of integers, having a single binary function name f in its vocabulary. Each entry of the table is a location. The location has two indices, i and j, for its row and column in the table, marked $f(i, j)$. The content of an entry (location) in the table is its value $[\![f(i, j)]\!]_S$.

It is sometimes useful to provide a location of a structure S by a term $f(t_1, \ldots, t_k)$. In this case, it stands as a shortcut for $f([\![t_1]\!]_S, \ldots, [\![t_k]\!]_S)$.

Definition 2 (Structure Union). *A structure S of vocabulary $\mathcal{F}$ over domain D is the* union *of structures S' and S'' of vocabularies $\mathcal{F}'$ and $\mathcal{F}''$, respectively, over D, denoted $S = S' \uplus S''$, if $\mathcal{F} = \mathcal{F}' \uplus \mathcal{F}''$, $[\![l]\!]_S = [\![l]\!]_{S'}$ for every location l in S', and $[\![l]\!]_S = [\![l]\!]_{S''}$ for every location l in S''.*

Definition 3 (Update). *An* update *of location l over domain D is a pair, denoted $l := v$, where v is an element of D.*

Definition 4 (Structure Modification). *The* modification *of a structure S into a structure S' of the same vocabulary and domain, denoted $\Delta(S, S')$, is the set of updates $\{l := v' \mid [\![l]\!]_S \neq [\![l]\!]_{S'} = v'\}$.*

Definition 5 (Structure Mapping). *Let S be structure of vocabulary $\mathcal{F}$ over domain D and $\rho : D \to D'$ an injection from D to domain D'. A* mapping *of S by ρ, denoted $\rho(S)$, is a structure S' of vocabulary $\mathcal{F}$ over D', such that $\rho([\![f(\overline{a})]\!]_S) = [\![f(\rho(\overline{a}))]\!]_{S'}$ for every location $f(\overline{a})$ in S.*

Structures S and S' of the same vocabulary over domains D and D', respectively, are *isomorphic*, denoted $S \simeq S'$, if there is a bijection $\pi : D \leftrightarrow D'$, such that $S' = \pi(S)$.

2.2 Axiomatic Sequential Procedures

The axiomatization of a "sequential procedure" is very similar to that of Gurevich's sequential algorithm [6], with the following two main differences, allowing for the computation of a specific function, rather than expressing an abstract algorithm:

- The vocabulary includes special constants *"In"* and *"Out."*
- There is a single initial state, up to changes in *In.*

Axiom 1 (Sequential Time). *The procedure can be viewed as a collection $\mathcal{S}$ of* states*, a sub-collection $\mathcal{S}_0 \subseteq \mathcal{S}$ of* initial states*, and a* transition function $\tau : \mathcal{S} \to \mathcal{S}$ *from state to state.*

Axiom 2 (Abstract State).

- States. *All states are first-order structures of the same finite* vocabulary $\mathcal{F}$.
- Input. *There are nullary function names In and Out in $\mathcal{F}$. All initial states ($\mathcal{S}_0 \subseteq \mathcal{S}$) are over the same* domain D*, and are equal up to changes in the value of In. (The initial states may be referred to as a single state S_0).*
- Isomorphism Closure. *The procedure states are closed under isomorphism. That is, if there is a state $S \in \mathcal{S}$, and an isomorphism π via which S is isomorphic to a $\mathcal{F}$-structure S', then S' is also a state in $\mathcal{S}$.*
- Isomorphism Preservation. *The transition function preserves isomorphism. That is, if states S and S' are isomorphic via π, then $\tau(S)$ and $\tau(S')$ are also isomorphic via π.*
- Domain Preservation. *The transition function preserves the domain. That is, the domain of S and $\tau(S)$ is the same for every state $S \in \mathcal{S}$.*

Axiom 3 (Bounded Exploration). *There exists a finite set T of "critical" terms, such that $\Delta(S, \tau(S)) = \Delta(S', \tau(S'))$ if S and S' coincide over T, for all states $S, S' \in \mathcal{S}$.*

The isomorphism constraints reflects the fact that we are working at a fixed level of abstraction. See [6, p. 89]:

> A structure should be seen as a mere representation of its isomorphism type; only the isomorphism type matters. Hence the first of the two statements: distinct isomorphic structures are just different representations of the same isomorphic type, and if one of them is a state of the given algorithm A, then the other should be a state of A as well.

Domain preservation simply ensures that a specific "run" of the procedure is over a specific domain. The bounded-exploration axiom ensures that the behavior of the procedure is effective. This reflects the informal assumption that the program of an algorithm can be given by a finite text [6, p. 90].

Elements of a procedure A are indexed $\mathcal{F}_A$, τ_A, etc.

Definition 6 (Runs).

1. *A* run *of procedure A is a finite or infinite sequence $S_0 \rightsquigarrow_\tau S_1 \rightsquigarrow_\tau S_2 \rightsquigarrow_\tau \cdots$, where S_0 is an initial state and every $S_{i+1} = \tau_A(S_i)$.*
2. *A run $S_0 \rightsquigarrow_\tau S_1 \rightsquigarrow_\tau S_2 \rightsquigarrow_\tau \cdots$* terminates *if it is finite or if $S_i = S_{i+1}$ from some point on.*
3. *The* terminating state *of a terminating run $S_0 \rightsquigarrow_\tau S_1 \rightsquigarrow_\tau S_2 \rightsquigarrow_\tau \cdots$ is its last state if it is finite, or its stable state if it is infinite.*
4. *If there is a terminating run beginning with state S and terminating in state S', we write $S \rightsquigarrow^!_\tau S'$.*

Definition 7 (Procedure Extensionality). *Let A be sequential procedure over domain D. The* extensionality *of A, denoted* ext *A, is the partial function $f : D \to D$, such that $f(x) = [\![Out]\!]_{S'}$ whenever there's a run $S \rightsquigarrow^!_\tau S'$ with $[\![In]\!]_S = x$, and is undefined otherwise.*

Equality, Booleans and Undefined. In contradistinction with ASM's, we do not have built in equality, booleans, or undefined in the definition of procedures. That is, a procedure needs not have boolean terms ('True' and 'False') or connectives ('and' and 'or') pre-defined in its vocabulary; rather, they may be defined like any other function. It also should not have a special term for undefined values, though the value of the function implemented by the procedure is not defined when its run doesn't terminate. The equality notion is also not presumed in the procedure's initial state; furthermore, it cannot be a part of the initial state of an "effective procedure" (axiomatize later as a sequential procedure, satisfying Axiom 4). Nevertheless, the internal mechanism (transition function) of an effective procedure (e.g. a computer program) may perform equality checks, as the four axioms don't prevent it.

3 Effective Models

We are interested in sequential procedures that can be satisfying the "initial-data" postulate (Axiom 4, below), which states that it may only have finite initial data in addition to the domain representation ("base structure"). An "effective model" is some set of "effective procedures" that share the same domain representation.

We formalize the finiteness of the initial data by allowing the initial state to contain an "almost-constant structure."

Definition 8 (Almost-Constant Structure). *A structure F is* almost constant *if all but a finite number of locations have the same value.*

Since we are heading for a characterization of effectiveness, the domain over which the procedure actually operates should have countably many elements, which have to be nameable. Hence, without loss of generality, one may assume that naming is via terms.

Definition 9 (Base Structure). *A structure S of finite vocabulary $\mathcal{F}$ over a domain D is a* base structure *if all the domain elements are the value of a unique $\mathcal{F}$-term. That is, for every element $e \in D$ there exists a unique $\mathcal{F}$-term t s.t. $[\![t]\!]_S = e$.*

A base structure is isomorphic to the standard free term algebra (Herbrand universe) of its vocabulary.

Proposition 1. *Let S be a base structure over vocabulary G and domain D, then:*

- *The vocabulary G has at least one nullary function.*
- *The domain D is countable.*
- *Every domain element is the value of a unique location of S.*

Example 1. *A structure over the natural numbers with constant zero and unary function successor, interpreted as the regular successor, is a base structure.*

Example 2. *A structure over binary trees with constant nil and binary function cons, interpreted as in Lisp, is a base structure.*

Axiom 4 (Initial Data). *The initial state consists of an infinite base structure and an almost-constant structure. That is, the initial state S_0 is $BS \uplus AS \uplus \{In\}$ for some infinite base structure BS and almost-constant structure AS.*

An *effective procedure* must satisfy Axioms 1–4. An *effective model* E must be some set of effective procedures that share the same base structure. That is, $BS_A = BS_B$ for every effective procedures A and B in E.

Big steps and small steps. A computational model might have some predefined complex operations, as in a RAM model with built-in integer multiplication. A view of such a model as a sequential procedure allows the initial state to include these complex functions as oracles [6]. Since we are demanding effectiveness, we cannot allow arbitrary functions as oracles, and force the initial state to include only finite data over and above the domain representation (Axiom 4). Hence, the view of a model at the required abstraction level is done by "big steps," which may employ complex functions, while these complex functions are implemented by a finite sequence of "small steps" behind the scenes. That is, a function that is the extensionality of an effective procedure may be included (as an oracle) in the initial state of another effective procedure. (Cf. the "turbo" steps of [5].)

4 Computational Power

Since we are dealing with models that operate on different data structures, we adopt a variant of the quasi-ordering on extensional power developed in [2]. The notion of [2] is based on a surjective mapping between model domains, while here, in order to meet also more conservative comparison notions, we allow only bijective mappings. As a result, we consider Turing machines (TM) to be at least as powerful as a model A only if TM implements all functions implemented by A ($\mathrm{TM} \supseteq A$) or TM is isomorphic to such a model ($\mathrm{TM} \simeq B \supseteq A$).

We consider only deterministic computational models, hence their computational power can be viewed as a set of functions. As models may have non-terminating computations, we deal with sets of 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." An extension of this assumption to the general case can be found in [2].

Definition 10 (Computational Model).

- *A* computational model *A over domain D is an abstraction of an object that computes a set of partial functions $f : D \to D$, which may be interpreted as total functions $f : D \to D \cup \{\perp\}$.*

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

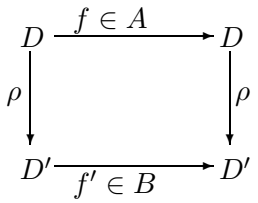

Figure 1: Model Mapping

- *We write* $\mathrm{dom}\ A$ *for the domain over which model* A *operates.*
- *The* extensionality *of a model* A*, denoted* $\mathrm{ext}\ A$*, is the set of functions that* A *computes. That is, the extensionality is an algebra, of possibly infinite vocabulary, over* D*.*
- *For models* A *and* B*, and a function* f *we shall write* $f \in A$ *as shorthand for* $f \in \mathrm{ext}\ A$*, and* $A \subseteq B$ *as short for* $\mathrm{ext}\ A \subseteq \mathrm{ext}\ B$*.*

To deal with models operating over different domains it is incumbent to map the domain of one model to that of the other. As a result, we get *model mapping*, which is a mapping of the model's extensionality, as defined in Definition 5. Model mapping is illustrated by Figure 1.

Definition 11 (Computational Power).

1. *Model* B *is* (computationally) at least as powerful *as model* A*, denoted* $B \succsim A$*, if there is a bijective structure mapping* $\pi : \mathrm{dom}\ B \to \mathrm{dom}\ A$ *such that* $\pi(B) \supseteq A$*.*
2. *Models* A *and* B *are* computationally equivalent *if* $A \precsim B \precsim A$*, in which case we write* $A \approx B$*.*

Proposition 2. *The computational power relation* $\precsim$ *between models is a quasi-order. Computational equivalence* $\approx$ *is an equivalence relation.*

The above notion, when replacing the requirement for a bijective mapping with an injective mapping, is the notion of "implemented" used in [7, p. 52] and of "incorporated" used in [14, p. 29].

It should be mentioned that, generally, a model might be isomorphic to one computing more functions [1]. Fortunately, TM are an exception and are not susceptible to this anomaly [1].

5 Effective Equals Computable

Theorem 1. *Turing machines are an effective model.*

Theorem 2. *Turing machines* (TM) *are at least as powerful as any effective model. That is,* $\mathrm{TM} \succsim E$ *for every model* E *satisfying the effectivity axioms.*

6 Discussion

Postulate minimality. An effective procedure should satisfy, by our definitions, Axioms 1–4. In the introduction, we argued for the necessity of the postulates from the intuitive point of view of effectiveness. Moreover, omitting any of them allows for models that compute more than Turing machines:

1. The Sequential Time Axiom is necessary if we wish to analyze computation, which is a step-by-step process. Allowing for transfinite computations, for example, would allow a model to precompute all values of a recursively-enumerable function.

2. In the context of effective computation, there is no room for infinitary functions, for example. Without closure under isomorphism there would be no value to the Bounded-Exploration Axiom, allowing the assigning of any desired value to the *Out* location.

3. By omitting the Bounded-Exploration Axiom, a procedure need not have any systematical behavior, hence may "compute" any function by simply assigning the desired value at the *Out* location. That is, for each initial state S there is a state S', s.t. $\tau(S) = S'$ and $[\![Out]\!]_{S'}$ is the 'desired' value.

4. Omitting the Initial-Data Axiom, one may "compute" any function (e.g. a halting oracle), by simply having all its values in the initial state. Such functions could also be encoded in equalities between locations, were the initial data not (isomorphic to) a free term algebra.

Algorithm *vs.* model. In [6], Gurevich proved that any algorithm satisfying his postulates can be represented by an Abstract State Machine. But an ASM is designed to be "abstract," so is defined on top of an arbitrary structure that may contain *non-effective* functions. Hence, it may compute non-effective functions. We have adopted Gurevich's postulates, but added an additional postulate (Axiom 4) for effectivity: an algorithm's initial state may contain only finite data in addition to the domain representation. Different runs of the same procedure share the same initial data, except for the input; different procedures of the same model share a base structure. We proved that—under these assumptions—the class of all effective procedures is of equivalent computational power to Turing machines.

Constructivity. With our definition of base structure (Definition 9) domain elements are of no importance, as long as there is a way to construct them. Hence, the domain is effective up to isomorphism. One may wish to require that domain elements themselves be effective (not up to isomorphism), requiring additional axioms.

References

[1] Udi Boker and Nachum Dershowitz. Comparing computational power. *Logic Journal of the IGPL*, to appear. Available at: `http://www.cs.tau.ac.il/~udiboker/files/ComparingPower.pdf`.

[2] Udi Boker and Nachum Dershowitz. How to compare the power of computational models. In *Proceedings of First Computability in Europe Conference (CiE 2005)*, Lecture Notes in Computer Science. Springer-Verlag, to appear. Available at: `http://www.cs.tau.ac.il/~udiboker/files/HowToCompare.pdf`.

[3] B. Jack Copeland. Hypercomputation. *Minds and Machines*, 12:461–502, 2002.

[4] Martin Davis. The myth of hypercomputation. In Christof Teuscher, editor, *Alan Turing: Life and Legacy of a Great Thinker*, pages 195–212. Springer, 2003.

[5] N. G. Fruja and R. F. Stärk. The hidden computation steps of Turbo Abstract State Machines. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Abstract State Machines — Advances in Theory and Applications, 10th International Workshop, ASM 2003, Taormina, Italy*, pages 244–262. Springer-Verlag, Lecture Notes in Computer Science 2589, 2003.

[6] Yuri Gurevich. Sequential Abstract State Machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1:77–111, 2000.

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

[8] Tien D. Kieu. Quantum algorithm for hilbert's tenth problem. *International Journal of Theoretical Physics*, 42:1461–1478, 2003.

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

[10] Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, Reading, MA, 1968.

[11] Toby Ord. Hypercomputation: Computing more than the turing machine. *Technical report, University of Melbourne, Melbourne, Australia*, 2002.

[12] Joseph R. Shoenfield. *Recursion Theory*, volume 1 of *Lecture Notes In Logic*. Springer-Verlag, Heidelberg, New York, 1991.

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

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