Honest Universality

Nachum Dershowitz* and Evgenia Falkovich†
School of Computer Science, Tel Aviv University, Ramat Aviv, Israel

Abstract

We extend the notion of universality of a function, due to Turing, to arbitrary (countable) effective domains, taking care to disallow any cheating on the part of the representations used. Then, we can be certain that effective universal functions cannot simulate non-effective functions.

Keywords: universal function; representation; encoding; effectiveness; computability

1 Introduction

Alan Turing, in his groundbreaking 1936 paper on the undecidability of the Halting Problem and the Entscheidungsproblem [1], also invented the notion of a universal machine. He explained the idea as follows:

> **The universal computing machine.** It is possible to invent a single machine which can be used to compute any computable sequence. If this machine I is supplied with a tape on the beginning of which is written the [standard description] of some computing machine M, then I will compute the same sequence as M.

There are two domains that are standard in discussions of computability: (1) finite strings over finite alphabets—for which the Turing-computable partial functions are the effective ones, and (2) the natural numbers—for which the effective functions are identified with the partial recursive ones. Davis [2] and Rogers [3] have proposed general definitions of universality for Turing machines and for partial recursive functions, respectively.

For other (countable) domains, we will adopt the analogous notion of effectiveness of a model of computation that was developed in [4]. Armed with the appropriate concept of generic effectiveness (Section 3), we extend the notion

*Research done partly while on leave at INRIA-LIAMA (Sino French Lab in Computer Science, Automation and Applied Mathematics), Tsinghua University, Beijing, China.

†This work was carried out in partial fulfillment of the requirements for a Ph.D. degree at Tel Aviv University.

of universality to arbitrary sets of functions over arbitrary domains (Section 5), while addressing oft-ignored questions of "honesty" of representations of domains (Section 4), pairings of elements (Section 6), and encodings of programs (Section 2). Our main result (Theorem 5) assures us that—under fairly weak conditions—representations cannot boost computational power, so an effective universal function cannot encompass non-effective functions.

2 Function Encodings

In the simplest case, a universal function simply "carries on its shoulders" a whole set of functions. The functions we speak of may be partial (unless stated otherwise). In particular, a universal function is meant to simulate both total and partial functions. So universal functions are partial by their very nature, and may be undefined at some points.

Let Φ be some (usually, but not necessarily, countably infinite) set of unary functions over a domain D. Given a binary function ψ over D, let $\Psi = \{\psi_a : a \in D\}$ be the set of all specializations (cross-sections) $\psi_a = \lambda y.\, \psi(a, y)$ of ψ in its first parameter. We say that ψ is *universal* for Φ if $\Psi \supseteq \Phi$. This is the same as requiring there to be an *encoding* $\# : \Phi \to D$ such that $\varphi = \psi_{\#\varphi}$ for all $\varphi \in \Phi$. Equality of the two partial functions, the original one φ and the simulation $\psi_{\#\varphi}$, means that $\varphi(y) = \psi(\#\varphi, y)$ for all $y \in D$, where equality (here, as well as later) is "Kleene's", so the two sides must also agree with regard to the values at which either is undefined, in which case both must be undefined.

We are placing no demands on the encoding ($\#$), only that there be some parameter value ($a = \#\varphi$) for which the universal function (ψ) exhibits the identical argument-value relation as that of the given function (φ). The idea of universality is that one function can by itself cover a collection of functions, in an extensional sense, but not that it uses the same, or similar, means as the given programs.

If one wants to incorporate functions of greater arity than 1, then one requires a family of universal functions, one for each arity, which we may imagine as one and the same varyadic function. In that case, we demand that $\varphi(y_1, \ldots, y_\ell) = \psi(\#\varphi, y_1, \ldots, y_\ell)$, for all arities ℓ, and values $y_1, \ldots, y_\ell$.

In practice, of course, we are interested in *effective* universal functions, functions that can be expressed as programs. Then, a single universal function ψ provides an effective means of computing any and all computable functions φ, just by varying ψ's first parameter. We discuss what it means to be effective in the next section.

In particular, nothing in our definition explicitly rules out a perverse, non-computable permutation of a standard enumeration of programs, like assigning even codes to total (universally halting) φ and odd codes to strictly partial ones. Such an encoding, however, would preclude a universal function ψ being effective, because, were it to be, then $\lambda i.\, \psi_{2i}$ would be an effective enumerator of programs for all the total recursive functions, an impossibility. If the universal function is itself effective, then no encoding can endow it with the ability to

simulate an uncomputable function. This is because the encoding $\#\varphi$ is simply a constant; thus, it can only supply finitely much additional information to ψ.

In point of fact, normally one is provided with a set of programs (standard descriptions of Turing machines, say) in some formalism (that is, programming language), which specifies the computed functions Φ, albeit with infinite duplication (there are always infinitely many programs with the same input-output behavior). There is no harm in this, as then, from our point of view, $\#\varphi$ is the code (e.g. the Gödel number) of any one out of the infinitely many programs that compute φ, which as it happens is an uncomputable coding. The related notion of an *interpreter* of one programming language in another, on the other hand, would suggest that the interpreter has an effective way of understanding the programs it is "interpreting", in which case we would want distinct codes for distinctly behaving programs. This way or that, the above definition of universality captures the intended extensional containment: a function ψ is universal for a set of programs if its specializations Ψ form a superset of the functions Φ computed by the given set of programs.

3 Effectiveness over Arbitrary Domains

For domains other than strings or numbers, we also need to be able to speak of effectiveness and effective universality. To that end, we need the characterization of effectiveness from [4]. For one thing, we may presume that all elements of the domain are reachable by means of given effective operations. Were that not the case, then even constant functions would be deemed non-effective. Moreover, all "basic" operations that are provided initially by a model of computation need to be effective. To be sure of that, one may simply require that the domain is generated by (finitely-many free) constructors and that initial operations are programmable from nothing more than said constructors. For example, the positive integers in binary are generated by the constant 1, and two unary operations, $\frown 0$ for $\times 2$, and $\frown 1$ for $\times 2 + 1$. Thus, an effective model (like a random-access machine) may incorporate basic arithmetic operations that are programmable using these constructors.

We also assume that there is an effective test of equality for the domain. Insisting that the domain be generated by (free) constructors, as opposed to (finitely-many) arbitrary operations, which may provide more than one way to refer to the same domain element, precludes the hiding of information in equalities between the values of distinct terms.

Besides the requirement that the given basic operations be programmable, effectiveness for general domains is just what one would expect. This generic notion of effectiveness allows us to talk formally about effectiveness of operations $\rho : C \rightharpoonup D$ from one domain to another, by requiring that everything can be built out of constructors for the union $C \cup D$ of the domains. That said and done, in what follows, we need not dwell on programming details.

The precise definition of effectiveness over arbitrary (countable) domains given in [4] is based on the generic abstract-state machine formalism of Gure-

vich [5]. States can be any (isomorphism-closed) class of logical structures over some fixed finite vocabulary. Furthermore, transitions must be effectively describable. This is formalized by saying that there is a fixed finite set of terms that "determines behavior" of the algorithm in a sense made precise in [5, 6]. (Transitions must also preserve the domain of states and commute with isomorphism.) In [7, 4], it has been shown how the Church-Turing thesis, namely, that no effective model can do more than a Turing machine can, follows provably from the formalization of the above characterization. Two other natural characterizations of effectiveness, based on Turing machines [8] and recursive functions (along the lines of [9] and others), are shown to be equivalent to this one in [10].

A *successor* function $s \in G$ is a unary function that enumerates its domain: $C = \{s^n(e) : n \in \mathbb{N}\}$ for some $e \in C$. For any finitely-generated domain, there clearly are effective successor functions. Just effectively enumerate all terms, (or just all constructor terms) thereby enumerating all domain values, while skipping over duplicate values (if required). Therefore, by standard search methods, any effective total injection f has an effective inverse f^{-1} such that $f^{-1}(f(x)) = x$ for all x in the domain. (Even the inverse of a partial injection f can be computed for its domain of definition by a breadth-first search, bounding the number of steps the program for $f(x)$ is allowed to run on any given potential inverse x.)

4 Domain Representations

To relate sets of functions (extensional models of computation) over two different domains, C and D, we will employ a liberal notion of "simulation", with inputs mapped from C to D via an arbitrary (not explicitly effective) injective representation ρ and outputs by the same, or another, representation σ. For example, one typically represents a Platonic number n as used by recursive functions as a string in unary (1^n) or in binary. Conversely, a string can be viewed as a number in a base the size of the alphabet. By the same token, graphs and other data structures can be represented as strings. What we don't really want is for the representation to include non-trivial computational information, like whether the graph has a Hamiltonian cycle.

Definition 1 (Simulation of Functions)**.** *For partial functions, $g : C \rightharpoonup C$ and $h : D \rightharpoonup D$, and (injective) representations $\rho, \sigma : C \xrightarrow{1\text{-}1} D$, we write $h \sqsupseteq_\rho^\sigma g$ and say that h* simulates *g via ρ and σ, if $g = \sigma^{-1} \circ h \circ \rho$, qua partial functions, that is, if $\sigma(g(x)) = h(\rho(x))$ for all x in C, where equality is Kleene's and σ is strict (in the sense that σ of undefined is undefined).*

Definition 2 (Simulation of Models)**.** *For sets of partial functions, $G \subseteq [C \rightharpoonup C]$ and $H \subseteq [D \rightharpoonup D]$, we write $H \sqsupseteq_\rho^\sigma G$ and say that H* simulates *G via ρ and σ if for all $g \in G$ there exists $h \in H$ such that $h \sqsupseteq_\rho^\sigma g$. We say that H* simulates *G, and write simply $H \sqsupseteq G$, if $H \sqsupseteq_\rho^\sigma G$ for some choice of representations ρ and σ.*

When ρ and σ are the identity function, $H \sqsupseteq^{\sigma}_{\rho} G$ is the superset relation, $H \supseteq G$. Simulation is transitive and reflexive.

When $\sigma = \rho$, it was shown in [11, proof of Thm. 4.7] that ρ is necessarily effectively computable (over $C \cup D$) if there is an effective function $\widehat{s} \in H$ that simulates a successor function $s \in G$. (The significance of successor was noted in [12].) Even when $\sigma \neq \rho$, it turns out that both must be effectively computable, provided that—in addition—the identity function is simulated effectively.

Proposition 3. *If there are effective simulations over domain D of the constructor and identity functions of domain C, via representations ρ and σ, then ρ and σ are effectively computable for $C \cup D$.*

Proof. Let s be a successor function for C, starting from constant $e \in C$, and let i be identity. It follows from the definitions that $\sigma(i(x)) = \widehat{i}(\rho(x))$, $\sigma(s(x)) = \widehat{s}(\rho(x))$, and $i(y) = y$, for all $x, y \in C$, where the hats indicate the respective simulating functions over D. Putting those together, we have $\widehat{i}^{-1}(\sigma(s(x)) = \rho(s(x))$, from which it follows that $\rho(s(x)) = \widehat{i}^{-1}(\widehat{s}(\rho(x)))$.

As explained at the end of the previous section, the inverse $\widehat{i}^{-1}$ of the simulation of the injection $i : C \xrightarrow{1\text{-}1} D$ is computable for elements of the image $\sigma(C)$ of σ in D. Hence, ρ is effectively computable, since $\rho(e)$ is just some constant, while $\rho(s(x))$ can be computed by a composition of effective functions from $\rho(x)$.

Furthermore, $\sigma(x) = \sigma(i(x)) = \widehat{i}(\rho(x))$ is computable. □

When functions take more than one argument, the representation function ρ should be extended to tuples: $\rho\langle x_1, \ldots, x_\ell\rangle = \langle\rho(x_1), \ldots, \rho(x_\ell)\rangle$. Otherwise, the above definitions are unchanged.

5 Universality Across Domains

Using the above notion of simulation, we arrive at the following generic definition of a universal function:

Definition 4 (Universality)**.** *Let Φ be some set of unary functions (over a domain C). A binary partial function ψ (over domain D) is* universal *for Φ if Ψ $(= \{\lambda y.\, \psi(a, y) \colon a \in D\})$ simulates Φ.*

In other words, ψ is universal for Φ if there is some injective input representation $\rho : C \xrightarrow{1\text{-}1} D$, and an injective output representation $\sigma : C \xrightarrow{1\text{-}1} D$, plus an arbitrary encoding $\# : \Phi \to D$ of the functions in Φ, such that $\psi(\#\varphi, \rho(x)) = \sigma(\varphi(x))$ for all $\varphi \in \Phi$ and $x \in C$. Note that if ψ is universal ($\Psi \sqsupseteq \Phi$) and ψ' simulates ψ (hence, $\Psi' \sqsupseteq \Psi$), then ψ' is also universal ($\Psi' \sqsupseteq \Phi$ by transitivity). Cf. [13, Thm. 1].

Our main result is that an effectively computable universal function can only simulate effective functions, as long as it simulates the constructors of a domain. Thus, the representation cannot in fact provide the universal function with any information that might allow computation of the uncomputable.

Theorem 5. *Let Φ be some set of unary functions over a domain C, including constructors and identity. Then, if there is an effective universal function (over any domain D) for Φ, then all the simulated functions $\varphi \in \Phi$ are also effective.*

Proof. Let ψ be the universal function. We have $\varphi = \sigma^{-1} \circ \psi_{\#\varphi} \circ \rho$, for any $\varphi \in \Phi$. By Proposition 3, ρ and σ are effective, since the simulations ($\widehat{s} = \psi_{\#s}$, $\widehat{\imath} = \psi_{\#i}$, etc.) of the constructors and identity (s, i, and the like) are effective. So φ is an effective composition of effective functions. □

In other words, the universal function cannot underhandedly simulate harder functions than it itself is capable of computing.

Nevertheless, just because ρ is effective does not mean that it cannot hide some information, albeit computable, in the representation, simply by mapping $x \in C$ to a "tuple" $[x, f(x), g(x), \dots]$ in D, for finitely many computable functions f, g, etc., such as Hamiltonianism. (The square brackets here stand for any standard tupling operation that effectively converts a sequence into a single element.) But if there is an effective injective ρ and universal function ψ, it can be shown that there is also an effective bijective ρ' and universal function ψ', with the latter doing all the hard work. So, restricting the notion of simulation to bijective representations, though that is not the way things are usually done, could make sense.

6 Data Pairing

One potential problem with the above definition of universal function is that some models of computation—like Turing machines—do not take their inputs separately, but, rather, all functions are unary (string-to-string for Turing machines). In such cases, one needs to be able to represent pairs (and tuples) as single elements. One standard pairing function for the naturals (used be Gödel) is the injection $\langle i, j \rangle := 2^i 3^j$. Another, among many, is the (Cantor) bijection $\langle i, j \rangle := \frac{1}{2}(i + j)(i + j + 1) + j$, or this one (used by Kalmár and others): $\langle i, j \rangle := 2^i(2j + 1) - 1$. For strings, one usually uses an injection like $\langle u, w \rangle := u;w$, where ";" is some symbol not in the original string alphabet.

There are several ways to proceed. The pairing function could reside in the source domain C, or in the target domain D, or in the representation of C as D. Regardless, this need raises a critical issue. Unless we demand that pairing be effective, there could be a machine that does too much, computing even non-effective functions. For example, a naïve definition might simply ask that pairing be injective and that $\varphi(x) = \psi\langle \#\varphi, x \rangle$ for all $\varphi \in \Phi$ and $x \in D$ (omitting parenthesis around the pair). The problem is that an injective pairing could cheat and include the answer in the "pair". n contrast with encodings, which can only provide a finite amount of information regarding a function φ, a dishonest pairing $\langle \#\varphi, x \rangle$ may add an infinite quantity, by providing different data for each argument x of φ. For Turing machines, say, the pair $\langle u, w \rangle$ might be represented as $u;w$ when machine u halts on input w and as $u:w$ (with a colon instead of a semicolon) when it doesn't. Better yet, one could map

$\langle \#\varphi, y\rangle \mapsto [\varphi(y), \#\varphi, y]$, where the square brackets are some ordinary tupling function for the domain. Then a putative universal machine could effortlessly "compute" virtually anything, computable or otherwise, just by reading the encoded input pair.

So, we clearly need for pairing to be effectively computable, as Davis and Rogers also insist. But we are talking about models in which no function takes two arguments, so we might not have an appropriate notion of computable binary function at our disposal. To capture effectiveness of pairing in such circumstances, we demand the existence of component-wise successor functions. Let $\langle \cdot, \cdot \rangle : D \times D \xrightarrow{1\text{-}1} D$ be a pairing function for D. The component-wise successor functions operate as follows: $s_1 : \langle a, b\rangle \mapsto \langle s(a), b\rangle$ and $s_2 : \langle a, b\rangle \mapsto \langle a, s(b)\rangle$. If s, s_1 and s_2 are effective, then we will say that *pairing is effective*. This is because one can program pairing so that $\langle z, y\rangle := s_1^i(s_2^j\langle e, e\rangle)$, where $z = s^i(e)$ and $y = s^j(e)$. And if pairing is effective, then its two projections (inverses), $\pi_1 : \langle a, b\rangle \mapsto a$ and $\pi_2 : \langle a, b\rangle \mapsto b$, are likewise effective. (Generate all representations of pairs in a zig-zag fashion, until the desired one is located. What the projections do with non-pairs is left up in the air.)

Another concern is that requiring that pairing be computable is too liberal for the purpose. One does not really want the pairing function to do all the hard real work itself. For example, the mapping could include $\varphi(x)$ in the pair, at least for φ that are known to be total (like, for the primitive recursive functions, of which there are infinitely many), or for all functions that halt within some recursive bound. That would make it a trivial matter to be universal for those functions—just transcribe the answer from the input.

If, in addition to being effective, pairing is bijective, then we will deem it *honest*, since then there is no room for hiding information. For such a pairing with effective projections, there is an effective means of forming a pair $\langle a, b\rangle$ (by enumerating all of D until the two projections give a and b, respectively). Note that, with bijectiveness alone, without effectivity, one could still hide a fair amount of uncomputable information in a bijective mapping. For instance, imagine that 0 is the code of the totality predicate and that the rest of the naturals code the partial-recursive functions in a standard order. Map pairs $(i+1, z)$ to $3\langle i, z\rangle$, where $\langle \cdot, \cdot \rangle$ is a standard pairing; map $(0, z)$ to $3j+1$ when z is the (code of the) jth total (recursive) function; and map $(0, z)$ to $3k+2$ when z is the kth non-total (partial recursive) function. Now, let U be some standard effective universal function. Then, for y divisible by 3, $\psi(y) := U(y/3)$ would cover all the partial-recursive functions, whereas $\psi(y) := y \equiv 1 \pmod 3$ would yeild the uncomputable totality predicate, when $y = (0, z)$ is not divisible by 3.

In the presence of a (not necessarily honest) pairing function, we say that ψ is universal if $\Psi \sqsupseteq \Phi$, where, this time, $\psi_a = \lambda y.\, \psi\langle a, y\rangle$.

Definition 6 (Unary Universality)**.** *Let Φ be some set of unary functions (over a domain C). A unary function ψ (over domain D) is* universal *for Φ, via injective pairing function $\langle \cdot, \cdot \rangle$ (over D), if Ψ $(= \{\lambda y.\, \psi\langle a, y\rangle : a \in D\})$ simulates Φ. If, in addition, pairing is bijective, then we call the universal function* honest.

That is, ψ is universal if $\psi\langle \#\varphi, \rho(x)\rangle = \sigma(\varphi(x))$ for $\varphi \in \Phi$ and $x \in C$. Of course,

we are interested in the case where both pairing and the universal function are effective.

Theorem 7. *Let Φ be some set of unary functions over a domain C, including constructors and identity. Then, if there is an effective unary universal function (over any domain D) for Φ, via an effective injective pairing, then all the simulated functions $\varphi \in \Phi$ are also effective.*

Proof. Apply Theorem 5 to $\psi'(z, y) := \psi\langle z, y\rangle$. □

Suppose $\Phi = \{\varphi_z\}_z$ is some standard enumeration of (the definitions of) the partial-recursive functions. Based on Davis's second definition of a universal Turing machine (which relies on a notion of effective mappings between strings and numbers, namely, recursive in Gödel numberings), Rogers defines the property "universal(III)" of a unary numerical function ψ to be that $\varphi_z(x) = \gamma(\psi\langle z, x\rangle)$ for some effective (recursive) bijection γ and effective (but perhaps dishonest) pairing $\langle\cdot,\cdot\rangle$. Let's say that such a ψ is *Rogers-universal*.

We may conclude the following from the definitions:

Theorem 8. *If a function is Rogers-universal, then it is universal in the sense of Definition 6. Furthermore, there is an honest effective universal function.*

Proof. Let ψ be the given Rogers-universal function. Take the standard enumeration for the encoding $\#$, identity for the input representation ρ, and γ^{-1} (which is well-defined and effective) for the output representation σ. Then, $\psi\langle\#\varphi_z, \rho(x)\rangle = \psi\langle z, x\rangle = \sigma(\varphi_z(x))$, as required.

For the second part, take some bijective pairing $\langle\!\langle\cdot,\cdot\rangle\!\rangle$ with effective projections π_1 and π_2, and let $\psi' := \lambda r.\, \psi\langle\pi_1(r), \pi_2(r)\rangle$ be our honest universal function. Putting those together, we get $\psi'\langle\!\langle z, x\rangle\!\rangle = \psi\langle z, x\rangle = \sigma(\varphi_z(x))$. □

7 Discussion

We have generalized the classical notion of universality to apply to arbitrary models over arbitrary domains. Indeed, for every effective model in the sense of [10], there is a universal program ψ over its own domain, which can be expressed as an effective abstract state machine. Moreover, any such ψ is universal for every effective model over any other effective domain, via a bijective representation between domains. (A bijection can be induced from the successor functions of the two domains.) We have seen that these universal functions cannot simulate more than can actually be done effectively within the model of computation itself.

Some directions for further thought are the following:

1. Extend the notion of universal evaluation to partial evaluations as in Kleene's s_{mn} Theorem [14] and the Futamura projections [15].

2. Consider steppers and interpreters, which take programs and simulate their step-by-step behavior, in a controlled fashion. Abstract state machines provide a means for capturing the step-by-step behavior of arbitrary classical algorithms.

3. Clarify the relation between a universal abstract state machine and non-effective universal functions. See [16, Sect. 6].

4. Consider a Turing machine simulation in which initial tapes may have infinite effective content, rather than being all blank at the ends, an issue that has been raised in the recent Wolfram competition [17].

Acknowledgements

We thank the referees for their expeditiousness and their helpful suggestions.

References

[1] Alan M. 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.turingarchive.org/browse.php/B/12` (viewed March 1, 2012). [doi: 10.1112/plms/s2-42.1.230].

[2] Martin Davis. The definition of universal Turing machine. *Proc. Amer. Math. Soc.*, **8**:1125–1126, 1957. [doi: 10.2307/2032691].

[3] Hartley Rogers, Jr. On universal functions. *Proceedings of the American Mathematical Society*, **16**(1):39–44, February 1965. Available at `http://www.jstor.org/stable/2033997`. [doi: 10.2307/2033997].

[4] Udi Boker and Nachum Dershowitz. The Church-Turing thesis over arbitrary domains. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 199–229. Springer, 2008. Available at `http://nachum.org/papers/ArbitraryDomains.pdf` (viewed March 1, 2012). [doi: 10.1007/978-3-540-78127-1].

[5] Yuri Gurevich. Sequential abstract state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, July 2000.

[6] Andreas Blass, Nachum Dershowitz, and Yuri Gurevich. Exact exploration and hanging algorithms. In *Proceedings of the 19th EA Annual Conferences on Computer Science Logic (Brno, Czech Republic)*, volume

6247 of *Lecture Notes in Computer Science*, pages 140–154, Berlin, Germany, August 2010. Springer. Available at `http://nachum.org/papers/HangingAlgorithms.pdf` (viewed March 15, 2012); longer version at `http://nachum.org/papers/ExactExploration.pdf` (viewed March 15, 2012)]. [doi: 10.1007/978-3-642-15205-4_14].

[7] Nachum Dershowitz and Yuri Gurevich. A natural axiomatization of computability and proof of Church's Thesis. *Bulletin of Symbolic Logic*, **14**(3):299–350, September 2008. Available at `http://nachum.org/papers/Church.pdf` (viewed March 1, 2012). [doi: 10.2178/bsl/1231081370].

[8] Wolfgang Reisig. The computable kernel of Abstract State Machines. *Theoretical Computer Science*, **409**(1):126–136, December 2008. Draft available at `http://www2.informatik.hu-berlin.de/top/download/publications/Reisig2004_hub_tr177.pdf` (viewed April 10, 2012). [doi: 10.1016/j.tcs.2008.08.041].

[9] Michael O. Rabin. Computable algebra, general theory and theory of computable fields. *Transactions of the American Mathematical Society*, **95**(2):341–360, 1960. [doi: 10.2307/1993295].

[10] Udi Boker and Nachum Dershowitz. Three paths to effectiveness. In Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, editors, *Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume **6300** of *Lecture Notes in Computer Science*, pages 36–47, Berlin, Germany, August 2010. Springer. Available at `http://nachum.org/papers/ThreePathsToEffectiveness.pdf` (viewed March 1, 2012). [doi: 10.1007/978-3-642-15025-8_7].

[11] Udi Boker and Nachum Dershowitz. Comparing computational power. *Logic Journal of the IGPL*, **14**(5):633–648, 2006. Available at `http://nachum.org/papers/ComparingComputationalPower.pdf` (viewed March 1, 2012). [doi: 10.1007/978-3-540-78127-1].

[12] Stewart Shapiro. Acceptable notation. *Notre Dame Journal of Formal Logic*, **23**(1):14–20, 1982. [doi: 10.1305/ndjfl/1093883561].

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

[14] Stephen C. Kleene. *Introduction to Metamathematics*. D. Van Nostrand, New York, 1952. [doi: 10.1007/978-0-8176-4769-8].

[15] Yoshihiko Futamura. Partial evaluation of computation process – An approach to a compiler compiler. *Systems, Computers, Controls*, **2**(5):45–50, 1971.

[16] Andreas Blass and Yuri Gurevich. The linear time hierarchy theorems for RAMs and abstract state machines. *J. of Universal Computer Science*, **3**(4):247–278, April 1997.

[17] Alex Smith. Universality of Wolfram's 2, 3 Turing machine. Technical report, Wolfram Science, 2007. Available at `http://www.wolframscience.com/prizes/tm23/TM23Proof.pdf` (viewed March 15, 2012).