

Rice’s Theorem in Higher Categorical Semantics

Effective presentations, homotopy invariance, and neural computation

Anonymous

16 October 2026 · Version 1.0

Abstract

Rice’s theorem concerns effective descriptions of behavior, rather than the physical substrate implementing those descriptions. We formulate this distinction using an acceptable numbering of partial computable functions and an extensional presentation into an infinity groupoid or an infinity topos. A nontrivial Boolean semantic predicate cannot have a total computable pullback to program codes. Boolean observations factor through zero truncation, so Postnikov truncation preserves this obstruction whenever the predicate retains its two witnesses. A separate realizability formulation identifies the partial-function assembly as having no nontrivial decidable subobject in the effective topos. For neural computation, we give an explicit, effective compiler from register machines to finite rational ReLU recurrent circuits, prove its simulation invariant, and transfer Rice’s theorem to their partial input-output semantics. We distinguish this result from fixed-horizon rational ReLU verification and verification of closed finite-state systems, both of which are decidable under stated hypotheses. An exact-arithmetic implementation and finite simulation checks accompany the proofs. The contribution is an assumption-explicit synthesis and reproducible construction, rather than a claim that classical undecidability or neural universality is new.

Keywords: Rice’s theorem; categorical semantics; infinity topos; realizability; recurrent neural networks; verification; ReLU.

1. Scope and contribution

The phrase “a property of a neural network” may refer to its syntax, a bounded computation, an unbounded trajectory, or its entire partial input-output function. These are different decision problems. Rice’s theorem [1] addresses the last kind when the model effectively represents all partial computable functions. Neither approximation power on compact sets nor a large parameter count supplies this hypothesis.

Higher categorical semantics adds equivalences, paths between equivalences, and further coherence. A Boolean predicate that respects these identifications is still a predicate on connected components. Its effectiveness must be assessed relative to a presentation by finite codes. This paper makes that requirement explicit and proves a relative Rice theorem in an arbitrary nondegenerate infinity topos. The theorem does not assert that an infinity topos, by itself, enforces computability.

The neural application uses exact rational states, unbounded iteration, and a finite rational ReLU circuit as the recurrent transition cell. The dimension can depend on the program. We provide the compiler equations and an executable implementation; no theorem below depends on experimental

behavior of a trained network. Earlier neural universality [6] and undecidability [7] are acknowledged. Our construction uses a different, explicitly specified interface from the weighted-language model of [7].

We work in classical external mathematics with a fixed universe of small sets and spaces. Internal topos statements are identified separately. All undecidability statements are uniform statements over finite descriptions, allowing unbounded running time and memory unless a bound is specified.

2. Effective descriptions and the diagonal argument

Definition 2.1 (acceptable numbering). Let

$$\varphi : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}, \quad \varphi_e(x) = \varphi(e, x),$$

be a universal partial computable evaluator. Its sections enumerate all unary partial computable functions. We assume a total computable specialization function s satisfying

$$\varphi_{s(a,z)}(x) \simeq \varphi_a(\langle z, x \rangle).$$

Here $\langle -, - \rangle$ is a computable pairing, and $\simeq$ means equal defined outputs or joint divergence. Let $\mathcal{R}$ be the set of these partial functions. Write $e \sim f$ when $\varphi_e = \varphi_f$ extensionally.

Lemma 2.2 (effective fixed point). For every total computable $t : \mathbb{N} \rightarrow \mathbb{N}$, there exists q with $\varphi_q = \varphi_{t(q)}$.

Proof. Universality supplies a code b for

$$(z, x) \mapsto \varphi_{t(s(z,z))}(x).$$

Put $q = s(b, b)$. Then, as partial computations,

$$\varphi_q(x) \simeq \varphi_b(\langle b, x \rangle) \simeq \varphi_{t(s(b,b))}(x) = \varphi_{t(q)}(x).$$

No step decides whether either computation terminates. $\square$

Theorem 2.3 (Rice). If $P \subseteq \mathcal{R}$ contains some φ_a and omits some φ_b , there is no total computable $D : \mathbb{N} \rightarrow \{0, 1\}$ such that

$$D(e) = 1 \iff \varphi_e \in P.$$

Proof. Given such D , define the total computable code transformation

$$t(e) = \begin{cases} b & D(e) = 1, \\ a & D(e) = 0. \end{cases}$$

For the fixed point q of Lemma 2.2, $D(q) = 1$ implies $\varphi_q = \varphi_b \notin P$, and $D(q) = 0$ implies $\varphi_q = \varphi_a \in P$. Both contradict correctness. $\square$

The theorem permits an algorithm to inspect every bit of a program's syntax. It forbids a total correct answer when the required answer is extensional and nonconstant. It does not forbid semidecision procedures, incomplete sound analyses, or decisions on restricted subclasses.

3. Higher categorical and topos formulations

Write $\mathcal{S}$ for the infinity category of spaces and $\mathbf{2} = \{0, 1\}$ for a discrete space. An infinity groupoid is a space, with points, paths, and higher paths; $\pi_0 B$ is its set of connected components. The zero truncation $\tau_{\leq 0} B$ retains those components and discards higher homotopy.

Here Map denotes a mapping space. An object is n -truncated when every mapping space into it has vanishing homotopy groups above degree n ; $\tau_{\leq n}$ is left adjoint to the inclusion of these objects. An infinity topos is an accessible left exact localization of a presheaf infinity category $\text{Fun}(\mathcal{C}^{\text{op}}, \mathcal{S})$ for a small infinity category $\mathcal{C}$ [4]. Left exact means preserving finite limits.

Definition 3.1 (extensional presentation). An extensional presentation of computable behavior in a space B is a map

$$d : \mathbb{N}_{\text{disc}} \longrightarrow B$$

such that $\varphi_e = \varphi_f$ implies a path $d(e) \simeq d(f)$. A Boolean semantic observation is a map $p : B \rightarrow \mathbf{2}$. It is nontrivial on the presentation when $p(d(a)) = 1$ and $p(d(b)) = 0$ for some codes a, b . It is effectively decidable on this presentation when its composite with d is total computable.

This condition identifies all extensionally equal programs at the level of components. A groupoid containing only syntax isomorphisms or parameter permutations need not satisfy it. Merely localizing at chosen architecture symmetries does not establish the required hypothesis.

For an infinity category $\mathcal{C}$ and fixed interface objects X, Y , one may take $B = \text{Map}_{\mathcal{C}}(X, Y)$. Its points are operations, paths identify parallel operations, and higher paths express coherence. The presentation condition must still be verified for this choice.

Theorem 3.2 (higher semantic Rice theorem). For an extensional presentation d , no nontrivial Boolean semantic observation p is effectively decidable on codes.

Proof. The set $\{e : p(d(e)) = 1\}$ is saturated under $\sim$, because a map to a discrete space takes endpoints of every path to the same bit. It therefore defines a nontrivial subset of $\mathcal{R}$. A computable composite $p \circ d$ contradicts Theorem 2.3. $\square$

The semantic map p can exist mathematically even though its pullback has no computable realizer. In particular, existence of a morphism in $\mathcal{S}$ is not a claim that it is an algorithm.

Proposition 3.3 (truncation invariance). For every integer $n \geq 0$,

$$\text{Map}(B, \mathbf{2}) \simeq \text{Map}(\tau_{\leq n} B, \mathbf{2}) \simeq \text{Map}(\tau_{\leq 0} B, \mathbf{2}).$$

Consequently Theorem 3.2 applies to the induced presentation in each $\tau_{\leq n} B$, with the same bits and witnesses.

Proof. A zero-truncated target is n -truncated for $n \geq 0$. Apply the universal property of truncation twice [4]. The factorizations preserve the composite on every code. $\square$

For example, take $B = \mathcal{R}_{\text{disc}} \times K(\mathbb{Z}, k)$ with $k \geq 1$, where the connected Eilenberg–Mac Lane space $K(\mathbb{Z}, k)$ has its only nonzero homotopy group $\pi_k = \mathbb{Z}$. Set $d(e) = (\varphi_e, *)$. There is genuine higher homotopy, but every Boolean observation is independent of its connected second factor. This is an example of the theorem’s scope, not evidence that arbitrary higher invariants encode computability.

Definition 3.4 (relative effective decision in a topos). Let $\mathcal{E}$ be a nondegenerate infinity topos, so its initial and terminal objects differ. Let $\Delta : \mathcal{S} \rightarrow \mathcal{E}$ be the constant-object functor, and

put $\mathbf{2}_{\mathcal{E}} = 1 \amalg 1 = \Delta \mathbf{2}$. Consider

$$d : \Delta \mathbb{N} \rightarrow B, \quad p : B \rightarrow \mathbf{2}_{\mathcal{E}}.$$

Let $d_e : 1 \rightarrow B$ be the section induced by the numeral e . Require $d_e \simeq d_f$ whenever $\varphi_e = \varphi_f$. A relative effective decision is a total recursive $D : \mathbb{N} \rightarrow \mathbf{2}$ together with a homotopy $p \circ d \simeq \Delta D$.

Theorem 3.5 (relative topos Rice theorem). If there are codes a, b with $pd_a \simeq 1$ and $pd_b \simeq 0$, no relative effective decision exists.

Proof. Were D such a decision, equality $\varphi_e = \varphi_f$ would give a path between the constant sections $D(e)$ and $D(f)$ of $1 \amalg 1$. The two coproduct injections are distinct in a nondegenerate topos, by disjointness of coproducts. Thus $D(e) = D(f)$ externally. The two witnesses give $D(a) = 1$ and $D(b) = 0$. Apply Theorem 2.3. $\square$

No geometric point or enough-points assumption is used. The theorem concerns this specified external realization of codes. A nonconstant predicate elsewhere in B need not be nontrivial on the presented behaviors. For an arbitrary subobject $U \hookrightarrow B$, a characteristic map to the subobject classifier $\Omega_{\mathcal{E}}$ expresses a proposition; a decision requires a factorization through $\mathbf{2}_{\mathcal{E}}$. These are distinct requirements.

The mapping-space argument of Proposition 3.3 also holds in $\mathcal{E}$ as

$$\mathrm{Map}_{\mathcal{E}}(B, \mathbf{2}_{\mathcal{E}}) \simeq \mathrm{Map}_{\mathcal{E}}(\tau_{\leq n} B, \mathbf{2}_{\mathcal{E}}).$$

Thus the same relative obstruction survives Postnikov truncation. This uses a reflective universal property; we do not assume that Postnikov truncation preserves arbitrary pullbacks. Separately, a geometric inverse-image functor $f^* : \mathcal{E} \rightarrow \mathcal{F}$, with $\mathcal{F}$ nondegenerate, preserves the presentation hypotheses and the two witnesses. Theorem 3.5 then applies in $\mathcal{F}$. An observation that a further quotient makes constant is outside its nontriviality hypothesis.

Proposition 3.6 (Lawvere’s diagonal pattern). In a cartesian closed infinity category with terminal object, suppose $u : A \times A \rightarrow Y$ represents every map $A \rightarrow Y$ up to homotopy: for every $h : A \rightarrow Y$, some $a : 1 \rightarrow A$ satisfies $u(a, -) \simeq h$. Then every endomorphism $v : Y \rightarrow Y$ has a homotopy fixed point.

Proof. Put $h = v \circ u \circ \mathrm{diag}_A$ and choose a representing h . Evaluation at a gives $y = u(a, a) \simeq v(u(a, a)) = v(y)$. $\square$

This is Lawvere’s diagonal argument [2] with equalities replaced by homotopies. For $Y = \mathbf{2}$, negation has no homotopy fixed point, obstructing the representation premise. Lemma 2.2 supplies the effective self-reference actually used by Rice’s theorem. A universal partial evaluator is not a total evaluation morphism representing every total Boolean function; we do not identify these premises.

4. A realizability interpretation

An assembly over partial recursive application consists of a set X and a nonempty set $E_X(x) \subseteq \mathbb{N}$ of realizers for each $x \in X$. A morphism $f : X \rightarrow Y$ is a set function tracked by a partial computable t : for every $e \in E_X(x)$, $t(e)$ is defined and belongs to $E_Y(f(x))$. The standard assembly embedding into Hyland’s effective topos Eff is full and faithful [3]. This is an elementary realizability topos; we do not identify it with a Grothendieck infinity topos.

Define the behavior assembly

$$\text{Beh} = (\mathcal{R}, E), \quad E(f) = \{e : \varphi_e = f\},$$

and the Boolean assembly with realizer sets $E(0) = \{0\}$, $E(1) = \{1\}$.

Theorem 4.1 (realizability Rice theorem). Every morphism $\text{Beh} \rightarrow \mathbf{2}$ in Eff is constant. Equivalently, Beh has no nontrivial decidable subobject.

Proof. Fullness yields an assembly tracker t for the underlying function. Every natural number is an index for a partial computable function, so t is defined on every natural number and returns the required Boolean value. If the function were nonconstant, t would decide a nontrivial extensional index set, contradicting Theorem 2.3. Conversely, each constant function has a constant tracker. Decidable subobjects are classified by maps to $1 \amalg 1$, giving the final claim. $\square$

This makes effectiveness part of the morphism notion. It does not say that every proposition about a partial function is trivial: characteristic maps to Ω_{Eff} need not provide Boolean decisions.

5. An explicit rational ReLU recurrent model

Definition 5.1 (register programs). A program has finitely many labels and registers $c_0, \dots, c_{m-1} \in \mathbb{N}$. Its instructions are:

- $\text{INC}(j, k)$: increase c_j by one and go to k ;
- $\text{DEC}(j, k_0, k_+)$: if $c_j = 0$ go to k_0 ; otherwise decrease it by one and go to k_+ ;
- HALT : return c_0 and remain in the same configuration.

Input x initializes $(c_0, c_1, \dots) = (x, 0, \dots)$ at a designated start label. Halting may occur initially. The finite-program model, allowing a finite number of scratch registers, computes exactly the partial recursive functions [5]. To recall the effective simulation, a tape is represented by two stacks encoded as base- b integers. Push uses multiplication by the constant b and addition of a digit; pop uses quotient and remainder by b . Increment, decrement, zero-test loops and scratch registers implement these operations. The finite control and the two stack routines simulate each Turing step. Conversely a Turing machine directly simulates register instructions. Hardwiring parameters and a universal interpreter give an acceptable numbering. Hence there is a total effective translation from Definition 2.1's codes to these programs.

Definition 5.2 (neural recurrent circuit). A cell is a finite acyclic circuit with rational affine gates and $\sigma(t) = \max(0, t)$ gates, taking $z \in \mathbb{Q}^d$ to $F(z) \in \mathbb{Q}^d$. Recurrence repeats the cell: $z_{t+1} = F(z_t)$. Skip connections and an affine output layer are permitted. Such a cell can also be arranged as a finite layered ReLU network by copying intermediate values, representing a signed value by its positive and negative parts if needed. We claim a recurrent circuit theorem, without assuming a single-layer update of the form $\sigma(Az + b)$.

Encode a configuration with label ℓ as

$$\iota(\ell, c) = (p_0, \dots, p_{L-1}, r_0, \dots, r_{m-1}), \quad p_k = \begin{cases} 1 & k = \ell, \\ 0 & k \neq \ell, \end{cases} \quad r_j = 2^{-c_j}.$$

Call these states well formed. They lie in $[0, 1]^{L+m}$, and the program counter is one-hot. Define

$$Z(r) = \sigma(2r - 1), \quad G(b, r) = \sigma(r + b - 1).$$

For encoded counters, $Z(r_j)$ is exactly the indicator of $c_j = 0$. For $b \in \{0, 1\}$ and $0 \leq r \leq 1$, $G(b, r) = br$; the circuit uses this ReLU expression, not multiplication of two state variables.

For a decrement instruction at ℓ acting on j , put

$$a_{\ell 0} = \sigma(p_\ell + Z(r_j) - 1), \quad a_{\ell +} = \sigma(p_\ell - Z(r_j)).$$

The first branch is active precisely at an active zero test, the second at an active positive test. Route p_ℓ to k for an increment, $a_{\ell 0}$ to k_0 and $a_{\ell +}$ to k_+ for a decrement, and p_ℓ back to ℓ for a halt. Each new control coordinate is the sum of its incoming activations. Update each counter by

$$r'_j = r_j - \frac{1}{2} \sum_{\ell: \text{INC}(j, -)} G(p_\ell, r_j) + \sum_{\ell: \text{DEC}(j, -, -)} G(a_{\ell +}, r_j).$$

All sums are finite and determined by the program. These are rational affine and ReLU operations only.

Theorem 5.3 (effective exact simulation). There is a total computable compiler $M \mapsto F_M$ satisfying

$$F_M(\iota(C)) = \iota(\text{step}_M(C))$$

for every register configuration C . Consequently the well-formed states are forward invariant, and iteration simulates every finite number of register steps exactly.

Proof. If $c_j = 0$, $r_j = 1$ and $Z(r_j) = 1$; otherwise $r_j \leq 1/2$ and $Z(r_j) = 0$. Hence the branch activations are Boolean. Exactly one instruction and one successor branch are active. The control-coordinate sums therefore give the next one-hot label. An active increment changes r_j to $r_j - r_j/2 = 2^{-(c_j+1)}$. An active positive decrement changes it to $2r_j = 2^{-(c_j-1)}$. In that case $c_j \geq 1$, so the result is at most one. An inactive instruction or a zero branch adds no counter change. Halt leaves all coordinates fixed. These are exactly the register rules. Induction proves the iteration statement. The finite instruction list directly determines a finite DAG with rational coefficients, proving computability of compilation. $\square$

No invariant is claimed for arbitrary real states outside the encoded set. The counter values approach zero as register contents grow. Bounded state magnitude therefore does not supply finite memory: exact states require arbitrarily many bits. Converting input x into 2^{-x} is computable but need not be efficient in the binary length of x .

Read a compiled circuit by checking its designated halt labels and, at halt, decoding $r_0 = 2^{-y}$ to return y . Exact rational decoding is effective: in lowest terms the numerator is one and the denominator is a power of two. A rational-cell execution is an ordinary partial computable procedure. Let $\nu_N : \mathbb{N} \rightharpoonup \mathbb{N}$ denote this partial neural behavior.

6. Neural Rice theorems and concrete specifications

Theorem 6.1 (compiler transfer). Suppose neural codes admit a total computable compiler C with $\nu_{C(e)} = \varphi_e$. For every nonempty proper $P \subset \mathcal{R}$, no algorithm that halts on every compiled code can correctly decide whether $\nu_{C(e)} \in P$.

Proof. Composing the alleged decision with C gives the total decision forbidden by Theorem 2.3. $\square$

Theorem 5.3 and the register translation provide this compiler. Membership in the compiler image need not itself be decidable: Theorem 6.1 is already a promise impossibility. It also rules out a total

decision on any larger effective model containing the compiled circuits with the same semantics. Nontriviality must occur on the represented behaviors; universality ensures it for every nonempty proper $P \subset \mathcal{R}$.

Corollary 6.2. Each of the following problems is undecidable for the compiled neural class:

Property of ν_N	Formal condition	Opposite witnesses
Termination at input zero	$\nu_N(0) \downarrow$	constant zero / nowhere defined
Totality	$\forall x \nu_N(x) \downarrow$	constant zero / nowhere defined
Implements zero everywhere	$\nu_N = (x \mapsto 0)$	constant zero / constant one
Never returns forbidden value one	$\forall x (\nu_N(x) \downarrow \Rightarrow \nu_N(x) \neq 1)$	constant zero / constant one

Proof. Each property is extensional, and its two witnesses are partial computable functions. Apply Theorem 6.1. $\square$

The first property is semidecidable by simulation; its undecidability does not imply absence of a semidecision procedure. No claim is made that every Rice property has the same arithmetical complexity.

Corollary 6.3 (behavioral equivalence). There is no total algorithm deciding $\nu_N = \nu_{N'}$. If neural semantics has an extensional presentation d into a space or topos as in Section 3, every nontrivial Boolean observation remains undecidable on neural codes. If, additionally, $d(N) \simeq d(N')$ holds *only if* $\nu_N = \nu_{N'}$, deciding that path-existence relation is impossible.

Proof. An equivalence decision against a fixed constant-zero circuit would decide the third row of Corollary 6.2. The semantic observation claim is Theorem 3.2 or 3.5 after composition with C . Under the additional faithfulness condition, path existence is exactly behavioral equality. $\square$

Proposition 6.4 (unbounded recurrent safety). For rational ReLU cells, deciding from a rational initial state whether a designated rational half-space is ever reached is undecidable, even when the simulated trajectory remains in the unit cube.

Proof. Specialize an arbitrary program and input to a register program run from zero. Compile it, and let H be the half-space $\sum_h \text{halt } p_h \geq 1/2$. On well-formed trajectories its left side is zero or one. The trajectory reaches H exactly when the program halts. The initial state and half-space are rational and effective. A reachability decision would decide halting. $\square$

For arbitrary architectures, internal trajectory safety is not automatically a property of the eventual input-output function: two implementations can have the same outputs and different intermediate states. Proposition 6.4 therefore gives its own reduction. It does not silently infer every dynamical safety problem from extensionality.

7. Decidable boundaries

Proposition 7.1 (bounded rational ReLU verification). Given a finite rational ReLU cell, a fixed finite horizon T , a rational polyhedral initial-state set, and a finite Boolean combination of

rational linear state constraints, existence of a violating trajectory of length at most T is decidable. The same applies to a finite feedforward rational ReLU network.

Proof. Unroll the cell T times. For each ReLU relation $y = \max(0, z)$, choose one of the linear cases

$$(z \geq 0 \wedge y = z) \quad \text{or} \quad (z \leq 0 \wedge y = 0).$$

There are finitely many choices. Add the initial and violation constraints and decide the resulting finite systems of rational linear equalities and strict or weak inequalities. Boolean combinations can be put into finite disjunctive form. Linear real arithmetic is decidable, so finite enumeration decides existence. Universal safety is the complement of this existential problem. A finite feedforward graph is already a finite unrolling. $\square$

This argument establishes decidability, not an efficient algorithm; exact neural verification methods such as Reluplex [8] address computational cost. It also decides equality of two finite rational ReLU feedforward maps on a rational polyhedral domain, by testing whether any output coordinate differs. Such maps are total without an unbounded execution loop.

Proposition 7.2 (closed finite-state recurrence). A deterministic closed transition system on a known finite state space S has decidable reachability and eventual halting from a specified initial state.

Proof. Simulate until the target is reached or a state repeats. At most $|S|$ distinct states are visited before repetition. Determinism makes all future states repeat the previously observed cycle, so a missing target will never appear. $\square$

A fixed-width numerical neural system with bounded storage and specified arithmetic can meet this hypothesis. An unbounded external tape, growing context, input stream, tool memory, or arbitrary-precision arithmetic can invalidate it. Noncomputable real parameters also fall outside the effective indexing assumed here; no physical hypercomputation conclusion follows from them.

8. Proof certificates and useful verification

Theorem 8.1 (no complete bilateral effective certification). Let P be a nontrivial extensional property of the compiled neural behaviors. There cannot exist two computably enumerable certificate relations $V_+(N, c)$ and $V_-(N, c)$ that are sound for P and its complement, respectively, and such that every compiled N has a certificate on one side.

Proof. On a compiled input, dovetail the two certificate enumerations, searching all candidate certificates. Completeness eventually finds one; soundness determines the correct bit and prevents contradictory successful certificates. This is a total correct decision on the compiler image, contrary to Theorem 6.1. $\square$

The statement permits complete certification of one side when that side is semidecidable, proof-carrying models from a restricted class, and sound tools that return “unknown.” It also permits a bounded specification whose bound is part of the input. If a training or synthesis pipeline were claimed to produce a correct yes-or-no certificate for every arbitrary compiled recurrent model and a nontrivial extensional specification, Theorem 8.1 would refute that uniform guarantee. It says nothing about the success rate on a data distribution or a finite benchmark.

9. Artifacts, evidence, and authorship

The accompanying standard-library Python implementation compiles instructions into a serialized DAG containing only rational affine and ReLU nodes. Execution uses exact fractions. Its independent integer interpreter is compared with neural transitions on 2,700 enumerated instruction-boundary states and 4,800 seeded multistep transitions. Additional checks cover 121 addition runs, absorbing halt, serialization, rejected malformed encodings, a growing counter checked at 1,101 successive times, and the closed finite-state repetition bound. Seven tests pass. These finite checks corroborate the compiler implementation; the mathematical proofs establish the universal statements. The artifacts do not contain a Lean, Coq, or other machine-checked proof and do not implement an undecidability solver.

The manuscript includes its Markdown, LaTeX, PDF and HTML representations, the compiler, tests, examples, reference metadata and SHA-256 checksums. Commands and interpretation limits are in `README.md`. HTML uses MathML for browser-native mathematical rendering.

Authorship and automation. Submitted under the responsible human author’s established public byline, Anonymous. OpenAI Codex assisted with research, mathematical formulation, proof drafting, implementation and artifact preparation. The responsible human author authorized submission. Automated checks concern the finite implementation evidence and document generation; they are not independent peer review. No assertion of editorial acceptance or publication is made by the manuscript.

References

1. H. G. Rice. “Classes of recursively enumerable sets and their decision problems.” *Transactions of the American Mathematical Society* 74(2), 358–366 (1953). [Publisher PDF](#). DOI: [10.1090/S0002-9947-1953-0053041-6](https://doi.org/10.1090/S0002-9947-1953-0053041-6).
2. F. W. Lawvere. “Diagonal arguments and cartesian closed categories.” *Lecture Notes in Mathematics* 92, 134–145 (1969). Reprinted with author commentary, *Reprints in Theory and Applications of Categories* 15, 1–13 (2006). [Journal version](#).
3. J. M. E. Hyland. “The Effective Topos.” In A. S. Troelstra and D. van Dalen (eds.), *The L. E. J. Brouwer Centenary Symposium*, Studies in Logic and the Foundations of Mathematics 110, 165–216 (1982). [Author-hosted text](#). DOI: [10.1016/S0049-237X\(09\)70129-6](https://doi.org/10.1016/S0049-237X(09)70129-6).
4. J. Lurie. *Higher Topos Theory*. Annals of Mathematics Studies 170, Princeton University Press (2009), especially Section 5.5.6 on truncated objects. [Author’s preprint](#).
5. M. L. Minsky. “Recursive Unsolvability of Post’s Problem of ‘Tag’ and other Topics in Theory of Turing Machines.” *Annals of Mathematics* 74(3), 437–455 (1961). DOI: [10.2307/1970290](https://doi.org/10.2307/1970290).
6. H. T. Siegelmann and E. D. Sontag. “On the Computational Power of Neural Nets.” *Journal of Computer and System Sciences* 50(1), 132–150 (1995). [Author-hosted published paper](#).
7. Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May and Kevin Knight. “Recurrent Neural Networks as Weighted Language Recognizers.” *NAACL-HLT*, Volume 1, 2261–2271 (2018). [Publisher text and metadata](#). DOI: [10.18653/v1/N18-1205](https://doi.org/10.18653/v1/N18-1205).
8. Guy Katz, Clark Barrett, David L. Dill, Kyle Julian and Mykel J. Kochenderfer. “Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks.” *CAV* (2017). [Author preprint](#).