

# AN ENGINEERING READING OF THE BOUNDED HALTING PROBLEM AND OBSERVABLE PROPERTIES OF A UNIVERSAL MACHINE

**Sergey Salishev**

St. Petersburg State University  
Saint Petersburg  
199034  
Russia  
s.salishev@spbu.ru

Article history:

Received 26.06.2026, Accepted 20.09.2026

## Abstract

The halting problem asks whether a program eventually halts, with unrestricted resources. Engineering systems instead certify bounded halting: given a program, an input, and a step budget  $T$ , decide whether the program halts within  $T$  steps. This paper studies two of its uses. First, it defines an exact finite-budget problem, placing the simulated program and the checker on the same unit-cost timescale, so the checker can be embedded in an adversarial program. A diagonal argument with explicit additive overhead shows that no fixed deterministic checker decides every instance within any fixed fraction of  $T$  below one, for large  $T$ . In the nondeterministic case,  $T$  bounds the branch depth allowed for a stable observable decision. Second, bounded-halting instances give adversarial tests for finitary universality, since a universal model must support arbitrary finite computations as  $T$  grows. The tests check resource growth with  $T$ , the absence of a fixed deliberation bound, and later access to intermediate data. They separate a fixed model with extendable external resources, a model family that grows with  $T$ , and a larger system built around a fixed component. For machine-learning systems, this identifies the resource growth a universality claim assumes and whether it fails for deployed models.

## Key words

bounded halting problem, universal machine, non-determinism, finite resources, observable properties, machine-learning systems

## 1 Introduction

Engineering systems operate with finite resource limits. A monitor checks whether a run reaches a bad state

before a deadline. A model checker or symbolic executor often has a fixed search depth [Biere et al., 1999; King, 1976; Cadar and Sen, 2013]. A runtime controller has limited memory, precision, and time. A machine-learning component has a finite context and a bounded inference or generation budget. The required result is an exact answer for a stated system boundary and resource budget  $T$ .

Bounded halting is the smallest formal version of that contract because a program must halt before returning an answer. So checking it is a practical test for correctness. Given a program, an input, and a budget  $T$ , it asks whether the program halts within  $T$  steps. Direct simulation may require all  $T$  steps. The engineering question is whether one fixed implementation can decide every instance exactly in  $\alpha T$  steps for some fixed  $\alpha < 1$ . This  $\alpha$  is the practically desired certification speedup over simulation. Theorem 1 and Theorem 2 show there is no such speedup in both deterministic and nondeterministic cases.

Although universality may seem unimportant for fixed resources, it comes into play when the extensibility of the deployed system and the trade-off between time and computation are considered.

The undecidability of unbounded halting is a universality test [Davis, 1958]. A universal model must support arbitrary finite simulations as the budget  $T$  grows. An exact bounded-halting instance is a stress test for the implementation. Thus, the same finite problem connects exact budgeted decision with certification of universal computation. The universal model should not have:

1. a fixed state set size independent of  $T$ ;
2. a fixed deliberation bound independent of  $T$ ;
3. a fixed forgetting horizon independent of  $T$ .

To be universal, a fixed computation model may permit extendable external resources, such as tape or stack, without changing its transition rules. A growing model family may change its parameter count, precision, architecture, or exactly addressable context range. A larger system may build up resources around a fixed component.

Large language models give a current motivation for this distinction. The Transformer architecture introduced in [Vaswani et al., 2017] is an example of a modern system. A practical implementation has a fixed parameter count, finite precision, bounded runtime, and a usable context limited by its positional encoding accuracy. Universality results commonly use a growing model family in which one or more of these resources increase with  $T$  [Pérez et al., 2021; Merrill and Sabharwal, 2023a], giving a misleading impression for a single fixed model. A larger system can also add storage, tool calls, or scheduler time around a fixed component. Examples include retrieval-augmented generation [Lewis et al., 2020] and multi-agent LLM systems [Wu et al., 2024]. In these cases, certification applies to the combined system.

The unbounded formulation of the halting problem is fundamental for computability theory. But the halting behavior of a closed engineering system with finitely many configurations is decidable in principle by cycle detection (Proposition 1). Its configuration graph may still be too large to search within the requested budget. Section 3 reviews classical undecidability, hierarchy, and speedup results relative to bounded halting.

The contributions are as follows:

1. Theorem 1 states the deterministic fixed-budget diagonal bound with explicit additive overhead, and Corollary 1 extracts the implementation consequence. For any fixed  $\alpha < 1$ , exact uniform decision of  $BH_T$  cannot run in  $\alpha T$  time on all inputs once  $T$  is large enough.
2. Theorem 2 gives the nondeterministic reading: branching supplies width, while a stable exact decision inherits the branch-depth bound on deterministic subinstances. The same result is recorded for finite probability-one randomized implementations in Corollary 2.
3. The paper gives implementation-level tests for universality. Proposition 1 prohibits fixed-size models. Proposition 2 rules out fixed bounded deliberation, and Proposition 3 records finite forgetting as an obstruction to universal computation.
4. Table 1 applies observable tests to standard automata, multi-agent systems, and finite-resource Transformer variants, showing that deployed individual Transformer models do not support universal computation.

## 2 Preliminaries

Standard deterministic and nondeterministic Turing machines over finite alphabets are used throughout

[Hopcroft et al., 2001]. A configuration contains the finite control state, the tape contents that have been reached so far, and the head positions. One transition has unit cost. The exact choice of one-tape or multi-tape model only changes fixed or standard simulation overheads; the diagonal statements below state the overhead assumption explicitly.

Let  $U$  be a fixed universal simulator. On input  $\langle M \rangle$  and  $x$ ,  $U(\langle M \rangle, x)$  simulates  $M(x)$ . The existence of a universal machine goes back to Turing’s original construction [Turing, 1937]; the modern textbook presentation is standard [Hopcroft et al., 2001]. Two clocks are distinguished. The *simulated clock* counts transitions of  $M$ . The *decision clock* counts transitions of a machine attempting to decide a bounded-halting predicate. They are distinguished for accounting, but the diagonal arguments place both on the same fixed unit-cost transition scale and compare them for a fixed budget  $T$ .

**Definition 1 (Deterministic bounded halting).** For a deterministic machine  $M$ , input  $x$ , and integer  $T > 0$ , define  $BH_T(M, x)$  by

$$BH_T(M, x) = 1 \iff M(x) \text{ accepts} \\ \text{within } T \text{ transitions.}$$

*Rejecting before  $T$ , looping beyond  $T$ , and not halting before  $T$  all give value 0. The analogous halting version is equivalent for the present purpose after the usual change of final states.*

**Definition 2 (Nondeterministic bounded halting).**

For a nondeterministic machine  $N$ , input  $x$ , and integer  $T > 0$ , define  $NBH_T(N, x)$  by

$$NBH_T(N, x) = 1 \iff \text{some branch accepts} \\ \text{within depth } T.$$

*Thus  $T$  bounds the height of the computation tree. The definition is the finite-budget version of the usual existential acceptance semantics for nondeterministic time.*

**Definition 3 (Observable property).** An observable property is a property inferred from the implementation boundary (for example, memory size, transition budget, external interfaces, or retention policy) without requiring a complete semantic analysis of the model being implemented.

Observable properties are intentionally coarse. They are useful because implementation facts can refute universality even when the internal model is complicated. A finite-state transducer may be large, probabilistic, or difficult to inspect. If the whole closed system has a finite set of configurations that is fixed independently of  $T$ , then it fails as a standalone universal machine for all budgets.

The definitions above also fix the role of the budget. In Definitions 1 and 2,  $T$  denotes the number of local transitions that the simulated run or branch is allowed to take.

Two kinds of decider for  $BH_T$  are compared below. A *uniform* decider is a single machine  $H(p, x, T)$  that takes a program code  $p$ , an input  $x$ , and a budget  $T$ , and returns  $BH_T(p, x)$ , the value of Definition 1 for the machine coded by  $p$ . A *fixed-budget* decider  $H_T(p, x)$  is specialized to one budget: it returns  $BH_T(p, x)$  for that fixed  $T$ , and  $\{H_T\}$  denotes the resulting family. A uniform decider may be constrained by the need to read an encoding of  $T$ , but the lower bound below is stronger because it is phrased for the fixed-budget family, thus removing the possible artifact of reading a long unary clock from the input.

This convention is close to how resource-bounded verification is used. A model checker, symbolic executor, or test harness is often asked to search up to a specific depth [Biere et al., 1999; King, 1976; Cadar and Sen, 2013]. The answer is expected for that chosen depth. Constants and lower-order terms therefore matter operationally even when they would disappear in a complexity-class statement.

### 3 Related Work and Positioning

The paper uses standard notions from computability and automata theory: Turing machines, universal simulation, finite automata, pushdown automata, and nondeterministic acceptance [Turing, 1937; Hopcroft et al., 2001]. The new emphasis is the finite engineering reading placed on top of these standard definitions.

#### 3.1 Bounded Halting and Asymptotic Hierarchies

Bounded halting is a standard topic in complexity theory. The time-hierarchy tradition begins with Hartmanis and Stearns [Hartmanis and Stearns, 1965]. Trevisan's Berkeley notes use a universal bounded-computation language [Trevisan, 2008]:

$$U = \{(\langle M \rangle, x, t, \ell) : \exists y, |y| \leq \ell, \\ M(x, y) \text{ accepts within } t \text{ steps}\}.$$

The integers  $t$  and  $\ell$  are encoded in unary in that presentation. A nondeterministic verifier guesses  $y$  and checks the  $t$ -step computation, so  $U \in NP$ .

The analogous class-style statement for the present notation is the language family

$$L_p^{NBH} = \{(\langle N \rangle, x) : NBH_{p(n)}(N, x) = 1\},$$

where  $n = |\langle N \rangle, x|$ . It belongs to  $NTIME(O(p(n)))$ , up to the fixed universal simulation overhead. Alternatively, if  $T$  itself is supplied in unary, then

$$L_{unary}^{NBH} = \{(\langle N \rangle, x, 1^T) : NBH_T(N, x) = 1\}.$$

This is an  $NP$  language. A nondeterministic machine can verify a yes-instance by following one branch of length  $p(n)$ , while a deterministic exhaustive simulator traverses the computation tree. For a fixed model whose nondeterministic branching degree is bounded by  $b$ , this gives the usual contrast

$$\begin{aligned} \text{branch verification} &: NTIME(O(p(n))), \\ \text{tree traversal} &: DTIME(b^{p(n)}p(n)^{O(1)}). \end{aligned}$$

This class comparison is useful for asymptotic separation. The engineering contract studied here asks a sharper constant-level question. Trevisan's notes use polynomial bounds such as  $|x|^3$  and prove hierarchy statements in terms such as  $o(n^3)$ . Such notation is natural for separating complexity classes and leaves constants outside the statement: a lower bound stated only against  $o(n^3)$  still permits solving the same family in  $0.1 n^3$  steps. That speedup is still  $\Theta(n^3)$ , hence asymptotically invisible, but it can be decisive for an implementation. The present paper therefore treats  $T$  as a hard budget.

The nondeterministic hierarchy also has a classical literature. Cook proved a nondeterministic time hierarchy theorem, later sharpened by Seiferas, Fischer, and Meyer [Cook, 1973; Seiferas et al., 1978]. The finite reading below works alongside those results and isolates the implementation question faced by a system that must return one stable exact bit for the concrete predicate  $NBH_T$ .

#### 3.2 Speedup and Fixed Implementations

The present fixed-implementation result addresses a different question from classical speedup theorems. Linear speedup reduces multiplicative constants by changing the simulating machine, for instance by packing several simulated tape symbols into one transition [Hartmanis and Stearns, 1965]. Blum's speedup theorem is different again: it exhibits functions with no fastest program, so further speedups are obtained by changing the program [Blum, 1967]. Both statements are compatible with the present claim. Here the implementation boundary, machine model, and exact output contract are fixed first; the question is whether that fixed artifact can decide every  $T$ -step bounded-halting instance in  $\alpha T$  internal steps for some fixed  $\alpha < 1$ .

#### 3.3 Finitary and Infinitary Readings

Each fixed budget  $BH_T$  is decidable by the bounded search  $\exists t \leq T$ . The unbounded question has the form  $\exists t$  with no finite bound and is only *semidecidable*: acceptance is eventually witnessed, but non-acceptance never certifies  $t < \infty$ . Classical undecidability of halting is exactly this gap, and it is located only in the passage from a finite budget to no budget [Tao, 2008].

Two classical vocabularies name the same gap. In domain-theoretic language, the unbounded predicate

$BH_\infty$  is the directed supremum of the decidable approximations  $\{BH_T\}$  [Abramsky and Jung, 1994]: it is the semidecidable  $\exists t$ , with no decidable  $t < \infty$  read off at one largest finite  $T$ . Nonstandard analysis makes the same cut with a hyperfinite budget  $T^*$ : running for  $T^*$  steps is an internally bounded search, so by transfer  $BH_{T^*}$  is decidable [Robinson, 1966], and since any acceptance occurs at a standard finite step  $t \leq T^*$ , it settles halting. The problem is decidable in  $T^*$  but not in any standard finite  $T$ .

### 3.4 Machine-Learning Systems

Machine-learning systems motivate the implementation-boundary question. On the idealized side, attention models can be universal once arbitrary numerical precision and unbounded intermediate computation are assumed: Pérez et al. prove Turing-completeness for hard-attention Transformers under such assumptions [Pérez et al., 2021], and Merrill and Sabharwal show that the number of generated chain-of-thought tokens is itself a computational resource [Merrill and Sabharwal, 2023a]. Formal Transformer limitations are also known: self-attention is limited in language recognition unless heads or layers grow with input length [Hahn, 2020]; saturated Transformers sit within constant-depth threshold circuits [Merrill et al., 2022]; and log-precision Transformers face what Merrill and Sabharwal call a parallelism tradeoff: one forward pass chains at most as many steps as its fixed depth, so a longer sequential task needs generated tokens or repeated calls [Merrill and Sabharwal, 2023b]. For recurrent models, Weiss, Goldberg, and Yahav show that finite precision lowers language-recognition power and blocks unbounded counting tasks that arbitrary-precision idealizations can solve [Weiss et al., 2018].

This paper uses those results to separate ideal model assumptions from deployed finite-resource systems. A fixed finite-precision model with fixed context and bounded runtime is analyzed here through the same observable-resource tests as any other implementation.

## 4 Deterministic Finite Reading

The unbounded halting problem says that no total computable function decides halting for all programs and inputs [Davis, 1958]. A finite implementation changes this statement in a simple way.

**Proposition 1 (Fixed finite state).** *Consider a closed deterministic implementation whose set of reachable configurations has size  $C$ , where the same  $C$  is fixed independently of the requested bounded-halting budget  $T$ . For any loaded request, its halting behavior is decidable by simulating at most  $C + 1$  transitions.*

*Proof.* During a run, either the machine reaches a final configuration within the first  $C + 1$  transitions, or two configurations repeat. In the second case determinism

forces the future run to repeat the same cycle forever. Hence the machine does not halt after the first repeated configuration unless it already halted before the repetition.

This proposition is sometimes used to dismiss the halting problem as irrelevant to real machines. That conclusion is misleading. The configuration bound  $C$  can be so large that enumeration has no engineering value, and for the fixed machine that has actually been built this is exactly the point: exact classification may be finite in principle while still requiring the full price of search. The practical question asks what the fixed implementation can decide within the stated budget.

The bounded problem gives the practical replacement. For a concrete budget  $T$ , the trivial exact method is to simulate  $M(x)$  for  $T$  steps and observe whether acceptance occurs. The following diagonal theorem states the sense in which this method is worst-case optimal for exact uniform decision. It is stated as a fixed-budget family theorem to isolate the decision cost from the cost of reading the encoding of  $T$ .

**Theorem 1 (Deterministic diagonal bound).** *Fix an effective coding of deterministic machines and a unit-cost transition model. Let  $c$  be a constant large enough to cover the following wrapper overhead: after receiving the answer of a candidate decider, the wrapper can either enter a fixed nonaccepting infinite loop or accept within at most  $c$  additional transitions.*

*For all sufficiently large  $T$ , there is no deterministic machine  $H_T$  such that, for every machine code  $p$  and input  $x$ ,*

$$H_T(p, x) = BH_T(p, x)$$

*and  $H_T$  halts with that answer in at most  $T - c$  transitions. Consequently, no exact uniform bounded-halting decider can have worst-case decision time below  $T - O(1)$  on all fixed-budget diagonal instances.*

*Proof.* Assume that such an  $H_T$  exists. Construct a deterministic machine  $D_T$  which, on input  $z$ , runs  $H_T(z, z)$ . If the answer is 0, then  $D_T$  accepts. If the answer is 1, then  $D_T$  enters a fixed infinite loop. By the choice of  $c$ , whenever  $H_T(z, z)$  halts within  $T - c$  transitions and returns 0, the accepting run of  $D_T(z)$  finishes within  $T$  transitions.

Let  $d$  be the code of  $D_T$  and run  $D_T(d)$ . If  $H_T(d, d) = 1$ , then by correctness  $D_T(d)$  accepts within  $T$  transitions. But the definition of  $D_T$  makes it loop forever after receiving the answer 1, a contradiction. If  $H_T(d, d) = 0$ , then  $D_T(d)$  accepts within  $T$  transitions, so correctness requires  $H_T(d, d) = 1$ , again a contradiction. Therefore no such  $H_T$  exists.

**Corollary 1 (No fixed-factor exact shortcut).** *Under the assumptions of Theorem 1, fix any constant  $\alpha < 1$ . For all sufficiently large  $T$ , no exact deterministic*

decider for  $BH_T$  can halt on all inputs within  $\alpha T$  transitions.

*Proof.* Choose  $T$  large enough that  $\alpha T \leq T - c$ . A decider running within  $\alpha T$  transitions would then satisfy the forbidden  $T - c$  bound of Theorem 1.

The bound is a worst-case, uniform statement, and many programs can be classified statically or by short partial execution [Cousot and Cousot, 1977; Cook et al., 2011]. For any exact method that handles all program-input pairs, there are diagonal instances that require more than  $T - c$  transitions. Corollary 1 makes the practical consequence explicit: after the machine model and wrapper overhead are fixed, even a constant-factor exact shortcut such as  $T/2$  is impossible uniformly for all sufficiently large budgets.

The theorem concerns exact decision; probabilistic and heuristic predictions use a different contract. The constant  $c$  covers wrapper steps after the decider returns and depends on the fixed machine model, coding, and simulator. A universal accelerator may speed up many programs, but the all-input contract includes the diagonal program. The theorem gives a reason for using bounded halting as an engineering universality certification test. A proposed universal accelerator can speed up many useful programs. Under the exact all-input bounded-halting contract, the diagonal program still forces the accelerator to meet the bound it is asked to decide. The diagonal program is adversarial, but the universality claim includes adversarial programs.

The usual infinite contradiction is recovered by letting the budget escape every finite bound. The engineering reading keeps the finite number  $T$  in view: a device that claims to decide all  $T$ -step behavior has to spend, on some inputs, essentially the same order of resources as the behavior being classified.

## 5 The Budget as a Branching Limit

The phrase “bounded halting problem” carries an engineering reading that the asymptotic tradition obscures. In that tradition the bound is a function of input length and the question is which class of functions grows faster; that comparison, and how it relates to the present predicates, is taken up in Section 3. In the engineering reading the bound is a contract: the run is cut off after  $T$  steps because a deadline, energy budget, recursion limit, or search depth says so. A monitor with a budget of  $10^9$  transitions must decide whether the watched run accepts by that budget. The number  $10^9$  is the actual limit. The hard-budget reading therefore asks the sharper question: can an exact decider uniformly answer a  $T$ -step question in  $\alpha T$  steps for some fixed  $\alpha < 1$ ? The diagonal bound says no, once the model overhead has been fixed.

For deterministic machines the finite picture is a path of configurations

$$C_0 \rightarrow C_1 \rightarrow \cdots \rightarrow C_T.$$

Deciding  $BH_T$  asks whether an accepting configuration appears on that prefix. The direct algorithm walks the prefix. A faster exact algorithm must somehow infer a property of the unseen suffix of the prefix. The diagonal theorem shows why this cannot be guaranteed for all programs: the program being analyzed can use the analyzer’s early answer to make the answer false.

For nondeterministic machines the finite picture is a rooted tree. The root is the initial configuration, and every node has one child for each legal nondeterministic transition. The predicate  $NBH_T$  asks whether there is an accepting node at depth at most  $T$ . The width of the tree may be exponential in  $T$ , but the branch-depth budget is still a sequential resource along every certificate path. A certificate for acceptance is a path, and the path contains at most  $T$  local transitions that must be checked.

## 6 Nondeterministic Finite Reading

Nondeterminism is often described as parallel exploration of many possible computations. The bounded predicate  $NBH_T$  makes the resource interpretation precise. The budget  $T$  is the maximum branch depth, while the total number of nodes in the tree may be exponential in  $T$ . A branch may be guessed, but an accepting branch of length  $T$  still contains  $T$  successive transitions.

A nondeterministic universal simulator decides  $NBH_T(N, x)$  by guessing one branch and simulating at most  $T$  transitions, up to the fixed overhead of universal simulation. This section asks the matching lower-bound question for a fixed budget.

The nondeterministic lower-bound situation is more delicate than the deterministic one. For ordinary nondeterministic acceptance one accepting branch is enough. Negating such a statement asks for a universal condition over all branches, which is why nondeterministic hierarchy theorems require more subtle arguments than the elementary deterministic diagonal proof. The hierarchy background is reviewed in Section 3. From the engineering standpoint, branching changes width, while the finite reading of  $T$  as depth remains.

For systems that use nondeterminism as an actual decision procedure, the specification also states how an observer obtains a single answer. The next statement records the strong engineering form.

**Theorem 2 (Stable nondeterministic bound).** *Fix the model and overhead constant  $c$  as in Theorem 1. There is no nondeterministic family  $A_T$  that exactly decides  $NBH_T$  in at most  $T - c$  transitions on every branch and has stable observable output, meaning that all terminating branches return the same bit for each input.*

*Proof.* Suppose such a family  $A_T$  exists. Restrict its inputs to deterministic machines. For deterministic  $M$ , the predicate  $NBH_T(M, x)$  is identical to  $BH_T(M, x)$ , because the computation tree has one path. Since  $A_T$  has

stable output, a deterministic machine can follow any fixed legal branch of  $A_T$  and obtain the same answer within the same time bound. This gives a deterministic family deciding  $BH_T$  within  $T - c$ , contradicting Theorem 1.

The same reduction against Corollary 1 sharpens this to a constant factor: no stable nondeterministic decider settles  $NBH_T$  within  $\alpha T$  on every branch for a fixed  $\alpha < 1$ . This answers the question left open in Section 3: the branch-depth budget cannot be beaten, even by a constant factor.

Without stability, a recognizer may accept on one branch and provide no one-path witness for a negative answer. With stability, the deterministic obstruction applies to deterministic subinstances. Such an instance has one path, and transition  $t+1$  depends on transition  $t$ . Parallel speculation becomes exact only after the true continuation is verified. Branching can reduce search width but cannot remove this causal depth.

**Corollary 2 (Probability-one exactness).** *No finite randomized implementation whose possible branches all have positive probability can decide  $NBH_T$  with probability 1 and halt on every branch within  $T - c$  transitions.*

*Proof.* Probability-one correctness implies that no positive-probability branch returns a wrong bit. Thus all branches return the same exact bit, giving the stable decider prohibited by Theorem 2.

Random sampling may instead provide a confidence level, a failure probability, or an “unknown” outcome. Probability-one exactness in the finite model retains the diagonal bound.

## 7 Observable Properties of Nonuniversal Machines

The previous sections use bounded halting to compare a decision procedure with the computation it decides. The direction is now reversed. For a universal machine, the bounded halting problem cannot be decided faster than simulation. Given an implementation, what can be inferred about its universality by using bounded halting as a test, without understanding all its internal semantics?

The following properties are observable in the sense of Definition 3. They are practical refutation tests: if an implementation has one of them as a standalone machine, then it cannot be a universal machine for exact all-budget computation. Passing the tests leaves the constructive simulation proof as the next step.

Addressable memory, update rounds, context length, arithmetic precision, and persistent storage can often be determined from the implementation boundary.

### 7.1 Configuration Space Independent of $T$

For every finite  $T$ , a Turing-machine run visits finitely many configurations. The obstruction is one finite configuration set fixed before  $T$  is chosen. This differs from a fixed computation model that permits its tape or stack to grow as  $T$  grows.

If the whole closed implementation has such a fixed finite number of configurations after input loading, then its behavior is ultimately periodic by Proposition 1. This includes ordinary finite-state machines and also very large finite controllers with bounded memory. Classification depends on fixedness. A machine with  $2^{10^{12}}$  possible states is still finite if the same state set is used for every requested  $T$ ; the changing work space required to pay for arbitrary larger searches is absent.

A system may include neural components, probabilistic choices, caches, and schedulers. If all are bounded independently of  $T$ , the global state graph remains finite. Randomness gives a finite probabilistic transition system with the same boundary.

### 7.2 Bounded Deliberation

A universal machine must be able to spend more time when the simulated program requires more time. Therefore a standalone implementation with a fixed response bound cannot be universal for unbounded computation.

**Proposition 2 (Fixed deliberation).** *Suppose an implementation always produces its final answer after at most  $B$  internal transitions, where  $B$  is independent of the program being simulated and of the requested budget  $T$ . Then the implementation cannot be an exact universal bounded-halting decider for all  $T$ .*

*Proof.* Choose  $T > B + c$ , where  $c$  is the fixed wrapper overhead from Theorem 1. If the implementation exactly decided  $BH_T$  for all program-input pairs within  $B$  transitions, it would decide  $BH_T$  within  $T - c$  transitions. This contradicts Theorem 1.

A feed-forward circuit, fixed-depth inference graph, or controller with a fixed number of update rounds remains a bounded transducer. A larger fixed  $B$  handles more bounded tasks but does not supply an unbounded time parameter.

This test observes only the absence of a fixed deliberation bound. It does not establish a chain of useful dependent computations. A machine can pass the test by repeating one state forever. Resource growth and accessible intermediate data are separate requirements.

### 7.3 Inaccessible Intermediate Data

The next simple observable property is memory retention. A universal machine is programmable so that data produced at one time can affect computation arbitrarily far in the future. A program may discard intermediate symbols, and the model allows it to preserve or reconstruct data when the program requires it.

**Proposition 3 (Finite forgetting).** *Consider a sequential implementation with a fixed finite accessible state space  $S$ , where  $S$  is independent of  $T$ . If, after some fixed horizon  $h$  independent of  $T$ , earlier input or intermediate data can affect future outputs only through the current element of  $S$ , then the implementation cannot realize all computations of a universal machine.*

*Proof.* Choose  $n \geq h$  such that  $2^n > |S|$ . By the pigeonhole principle, two distinct binary strings  $u, v \in \{0, 1\}^n$  leave the implementation in the same accessible state. Give both runs the same continuation  $\#u$ . Their future behavior must be identical because earlier input can affect it only through the current element of  $S$ . Let  $M_{\text{copy}}$  be a Turing machine deciding  $\{w\#w : w \in \{0, 1\}^*\}$ , and choose  $T$  larger than its running time on these two inputs. Then

$$\begin{aligned} BH_T(M_{\text{copy}}, u\#u) &= 1, \\ BH_T(M_{\text{copy}}, v\#u) &= 0. \end{aligned}$$

The implementation gives the same answer on both inputs and therefore fails at least one of these bounded-halting instances. Thus it cannot realize every Turing-machine computation.

Forgetting is often appropriate: many controllers discard old data for stability, privacy, or bounded resource use. What the proposition isolates is that irreversible finite forgetting is an observable obstruction to standalone universality.

The same test applies to intermediate data as well as input data. Suppose a computation derives a lemma, a partial parse, a stack frame, or a simulated tape segment and later overwrites every accessible trace of it. If the implementation has no unbounded store in which a program could choose to save that trace, then later behavior cannot depend on the lost distinction. A universal machine allows the opposite programming pattern: compute a value now, keep it, and use it after an arbitrarily long delay.

## 7.4 An Audit Checklist

The audit separates the mathematical universality of the parametric model from the resources used by a fixed implementation. Four questions guide the audit before a simulation proof.

1. *Boundary.* What exactly is the machine: a neural network, an interpreter process, an operating system with files, or a human-in-the-loop workflow? A component and its surrounding system have different boundaries. A CPU may access extendable memory, and an agent may append to external files (see CSP [Hoare, 1985]). Tape or stack growth extends the resources of a fixed computation model; changes in precision, parameter count, architecture, or exactly addressable context define a growing model family.

2. *Accounting.* Where is the cost of exact search paid? If no resource can grow with  $T$ , the closed system is finite-state (Proposition 1).
3. *Clock.* What allows the machine to continue, and is that continuation part of the model? A component with a hard maximum number of rounds fails Proposition 2.
4. *Retention.* Where can a program place data that must be used later? A fixed cache, context window, or hidden state can summarize old information. Persistent files, logs, stacks, or databases can preserve a growing history when their capacity is unbounded and the program can access them (Proposition 3).

## 8 Application to Machine Classes

Table 1 applies the three observable tests: configuration growth with  $T$ , unbounded deliberation, and accessible intermediate data. Passing all three still requires a simulation proof.

In Table 1, “Growth” means that the exact configuration capacity available to the computation can grow with the budget  $T$ . The “Deliberation” column asks whether the system can make a run with  $T$  internal transitions before producing its final answer. It does not require progress or  $T$  useful dependent steps. An  $\varepsilon$ -loop or a fixed-context generation loop passes this column by repeating states. The “Later data” column asks whether intermediate data can be retained and addressed later in the computation. The last column is the conjunction of the three tests.

An  $\varepsilon$ -loop gives an FSM unbounded deliberation by repeating a state, while configuration growth and addressable history remain absent. It changes operational behavior through divergence but adds no new storage; under ordinary acceptance semantics it is summarized by  $\varepsilon$ -closure. Thus neither ordinary nor  $\varepsilon$ -transition FSMs pass all three tests.

An unbounded pushdown stack supplies growth and deliberation but only LIFO access: popped data is unavailable after removal. With two stacks, one can encode the left and right sides of a Turing tape, giving the usual universal construction [Hopcroft et al., 2001].

Unbounded CSP-style multi-agent systems [Hoare, 1985] pass all three tests at the class level. The system boundary permits growth in processes, communication, storage, or time. Persistent messages or stores can retain intermediate data. Two unbounded stack processes coordinated by a finite controller can simulate a Turing machine, with storage and execution time growing as needed. Fixed agents, memories, transcripts, and rounds instead define a finite transition system.

The computational model of the multiagent real-time system in [Krokhalev et al., 2026] resembles CSP. If such a system is deployed with a fixed response deadline that does not increase with the agent population, its fixed computation and communication rates bound the work available per request. Under this deadline, it fails

Table 1. Yes/no application of the three observable tests

| Class                                                                                                | Growth | Deliberation | Later data | All tests |
|------------------------------------------------------------------------------------------------------|--------|--------------|------------|-----------|
| Finite-state machine [Hopcroft et al., 2001]                                                         | No     | No           | No         | No        |
| $\varepsilon$ -transition finite-state machine [Hopcroft et al., 2001]                               | No     | Yes          | No         | No        |
| Pushdown automaton [Hopcroft et al., 2001]                                                           | Yes    | Yes          | No         | No        |
| Two-stack pushdown automaton [Hopcroft et al., 2001]                                                 | Yes    | Yes          | Yes        | Yes       |
| Unbounded CSP-style multi-agent system [Hoare, 1985]                                                 | Yes    | Yes          | Yes        | Yes       |
| Fixed finite-precision Transformer [Vaswani et al., 2017]                                            | No     | No           | No         | No        |
| Fixed-context Transformer with unbounded generation [Wei et al., 2022; Merrill and Sabharwal, 2023a] | No     | Yes          | No         | No        |

the fixed-deliberation test (Proposition 2), despite its distributed architecture.

The Transformer rows separate fixed forward computation from unbounded generation [Vaswani et al., 2017]. Generated intermediate tokens are a computational resource [Wei et al., 2022; Merrill and Sabharwal, 2023a], but unbounded generation changes only deliberation. With fixed precision and context, a deterministic decoder eventually repeats an accessible configuration and loops; stochastic decoding remains finite-state and gives no exact progress guarantee.

Transcript length supplies growth only when the computation can address it exactly. Text outside fixed context is not component memory, and positional information is limited by positional accuracy. Once more positions are required than the implementation can distinguish and address exactly, old data cannot act as arbitrary tape cells. A larger system may retain the transcript externally, but certification then applies to that combined boundary.

## 9 Discussion

### 9.1 Practical Universality

A fixed language model has bounded parameters, precision, context, sampling state, and internal work. Universality results instead assume growth at the model boundary. Removing the generation-length bound does not supply addressable growth: fixed-context generation can repeat states and loop. The growth rule, not the physical finiteness of one artifact, determines which model is being certified.

An external loop may add exactly addressable transcripts, files, tools, program execution, or unbounded scheduler time. Certification then applies to the combined system, provided that later computation can access these resources. An unbounded CSP-style system (Table 1) or a general-purpose interpreter may therefore be universal while each fixed component is not.

### 9.2 Limitations of the Reading

The result concerns exact answers for all program codes. Restricted classes, sufficient conditions, “unknown” outputs, distributions, and failure probabilities use different contracts and may permit faster tools. Classical speedup results also change the machine, program,

or contract; here the implementation boundary and exact output remain fixed.

## 10 Conclusion

The engineering question is whether a system with a stated boundary, memory, precision, and clock can return exact answers for all computations up to a budget  $T$ . Bounded halting gives a minimal form of this question. Some adversarial instances require exact simulation through  $T$  transitions. An exact checker must handle these instances within its stated resource contract.

Bounded halting also gives adversarial tests for universality certification. These tests check which resources grow with  $T$ , whether internal deliberation has a fixed bound, and whether intermediate data remain available. Unbounded deliberation alone can be a loop and does not certify useful progress. The resource growth may come from extendable tape or stack in a fixed computation model, from a growing family of models, or from a larger system around a fixed component. A universality claim must state this boundary, the growth rule, the continuation rule, and the storage used by later steps.

## References

- Abramsky, S. and Jung, A. (1994). Domain theory. In Abramsky, S., Gabbay, D. M., and Maibaum, T. S. E., editors, *Handbook of Logic in Computer Science*, vol. 3, pp. 1–168. Oxford University Press, Oxford.
- Biere, A., Cimatti, A., Clarke, E. M., and Zhu, Y. (1999). Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, vol. 1579 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, Springer, pp. 193–207.
- Blum, M. (1967). A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, **14**(2), pp. 322–336.
- Cadar, C. and Sen, K. (2013). Symbolic execution for software testing: Three decades later. *Communications of the ACM*, **56**(2), pp. 82–90.
- Cook, B., Podelski, A., and Rybalchenko, A. (2011). Proving program termination. *Communications of the ACM*, **54**(5), pp. 88–98.

- Cook, S. A. (1973). A hierarchy for nondeterministic time complexity. *Journal of Computer and System Sciences*, **7** (4), pp. 343–353.
- Cousot, P. and Cousot, R. (1977). Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, New York, ACM, pp. 238–252.
- Davis, M. (1958). *Computability and Unsolvability*. McGraw-Hill, New York.
- Hahn, M. (2020). Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, **8**, pp. 156–171.
- Hartmanis, J. and Stearns, R. E. (1965). On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, **117**, pp. 285–306.
- Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs, NJ.
- Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, 2nd edition.
- King, J. C. (1976). Symbolic execution and program testing. *Communications of the ACM*, **19** (7), pp. 385–394.
- Krokhalev, E., Arkhipov, I., Granichin, O., and Tarasova, E. (2026). Consensus algorithms for a multiagent operating system. *Cybernetics and Physics*, **15** (1), pp. 38–55.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc.
- Merrill, W. and Sabharwal, A. (2023a). The expressive power of Transformers with Chain of Thought. arXiv preprint arXiv:2310.07923.
- Merrill, W. and Sabharwal, A. (2023b). The parallelism tradeoff: Limitations of Log-Precision Transformers. *Transactions of the Association for Computational Linguistics*, **11**, pp. 531–545.
- Merrill, W., Sabharwal, A., and Smith, N. A. (2022). Saturated Transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, **10**, pp. 843–856.
- Pérez, J., Barceló, P., and Marinkovic, J. (2021). Attention is Turing-complete. *Journal of Machine Learning Research*, **22** (75), pp. 1–35.
- Robinson, A. (1966). *Non-standard Analysis*. North-Holland, Amsterdam.
- Seiferas, J. I., Fischer, M. J., and Meyer, A. R. (1978). Separating nondeterministic time complexity classes. *Journal of the ACM*, **25** (1), pp. 146–167.
- Tao, T. (2008). *Structure and Randomness: Pages from Year One of a Mathematical Blog*. American Mathematical Society, Providence, RI.
- Trevisan, L. (2008). Notes for lecture 1: U.C. Berkeley CS278 computational complexity. Lecture notes. Available at <https://lucatrevisan.github.io/cs278-08/lecture01.pdf>.
- Turing, A. M. (1937). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, **42** (1), pp. 230–265.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. (2022). Chain-of-Thought prompting elicits reasoning in large language models. arXiv preprint arXiv:2201.11903.
- Weiss, G., Goldberg, Y., and Yahav, E. (2018). On the practical computational power of finite precision RNNs for language recognition. arXiv preprint arXiv:1805.04908.
- Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. (2024). AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Proceedings of the First Conference on Language Modeling*.