

Sustainable Systems Laboratory

good data & systems that last

SSL Technical Report Series SSL-TR-2026-002

November 5, 2026

No Perfect Guardrail & No Perfect Guard

On the Decidability of Output Policies for Large Language Models

Ahmed Amer[1]

Loren Kinczel[2]

Darrell D. E. Long[1,3]

[1] *Sustainable Systems Laboratory, Department of Computer Science & Engineering, Santa Clara University*
[2] *Cultivate Investments* [3] *Pentexoire Consulting*

Series home: sustainable-systems-lab.org/reports/

Suggested Citation

Ahmed Amer, Loren Kinczel, and Darrell D. E. Long. *No Perfect Guardrail & No Perfect Guard: On the Decidability of Output Policies for Large Language Models.* SSL Technical Report Series, SSL-TR-2026-002. Sustainable Systems Laboratory, Santa Clara University, November 5, 2026. Preprint v0.1.
sustainable-systems-lab.org/reports/SSL-TR-2026-002/

This report is made available by the Sustainable Systems Laboratory, Santa Clara University. Preprint v0.1.

© 2026 The authors.

AI tools were used in the preparation of this version (v0.1) of the manuscript.

EXECUTIVE SUMMARY

No general promise of perfect guardrails can be fulfilled.
Our responsibility cannot be shed.

Any general assurance of a perfect guardrail is simply an impossibility. Particular policies can be enforced exactly, but that does not support a general promise of perfection. We argue that these limits demand precision in how we talk about the technology and both individual and institutional responsibility for managing its risks. Neither promises of general perfection, nor the impossibility of fulfilling them, can absolve any of us of that responsibility.

What we establish. We prove that, when a language model can produce any finite string, a computable guard can block every forbidden output and pass every permissible one if and only if an algorithm can decide whether an output violates the policy. For policies where no such algorithm exists, better engineering cannot deliver both guarantees, even if the policy is precisely defined and faithfully captures its authors' intent. Blocking all forbidden outputs then requires excluding infinitely many permissible ones. This does not tell us how often errors occur in practice or establish that guardrails are ineffective. It does rule out a general guarantee of perfect enforcement.

Why blocking more does not settle safety. Withholding permissible information may prevent us from recognizing a danger, finding a remedy, or challenging a defective policy. Compliance with guardrails cannot establish that these exclusions are harmless. Nor could even perfect enforcement establish that we defined the right policies and purposes. Simply put, what a system prevents has consequences, not just what it permits. Even a guardrail that prevents every unsafe output could create risk through its exclusions.

What we ask of policymakers. Responsible AI policy benefits neither from doomsaying nor from demands for impossible safeguards. We ask those proposing systems or restrictions to answer three questions:

1. **What, precisely, is being assured?** Specify the proposed rules and circumstances covered, and their justification. Distinguish proved guarantees from measured reliability and untested expectation.
2. **What might the restriction cost us?** Assess potential harms from exclusions alongside harms from allowed outputs. Make uncertainties explicit; realize that refusals alone do not guarantee safety.
3. **Who can question and repair the system?** Identify accountable people and institutions, and give them the information, time, and authority to challenge outcomes, revise policies, or suspend use. A human approval step alone—a token human-in-the-loop—does not constitute responsible oversight.

We argue that promising general perfection is an ethical failure of precision; invoking either that promise or its impossibility to shed responsibility is a moral failure. Good guardrails support responsible judgment. They cannot replace it. We must therefore retain the intellectual and moral humility to recognize flaws in our machines, our rules, our policies, and even ourselves—and the willingness to address those flaws.

No Perfect Guardrail & No Perfect Guard

On the Decidability of Output Policies for Large Language Models

Ahmed Amer[1] Loren Kinczel[2] Darrell D. E. Long[1,3]

[1]Sustainable Systems Laboratory, Department of Computer Science & Engineering, Santa Clara University
[2]Cultivate Investments [3]Pentexoire Consulting

November 5, 2026

Abstract. Expecting guardrails to provide a general guarantee of safety fails to recognize the limits of both computational enforcement and policy specification. We argue that promising such perfection is an ethical failure of precision, and that invoking it to abdicate responsibility is a moral failure. Guardrails can support responsible action, but cannot relieve researchers, developers, policymakers, or users of their obligations to examine what they claim, permit, and prevent.

We establish a precise computational boundary. If every finite string is a possible output of a language model, a computable guard that blocks every forbidden output and passes every permitted one exists if and only if the policy is decidable. If the policy is undecidable, every guard misclassifies infinitely many strings, and every guard that blocks all forbidden outputs excludes infinitely many permitted ones. We apply this characterization to arithmetic truth, program behavior, and harmful instructions to an idealized recipient. We distinguish guards from execution monitors, identify bounded and interceptable cases admitting exact enforcement, and explain why the results establish no positive lower bound on average error.

These distinctions matter to the practical and ethical interpretation of the proof. Exact enforcement, where possible, does not establish that a policy adequately captures safety. Conservative exclusion cannot be presumed harmless: exclusions may withhold warnings, remedies, or challenges needed to recognize the policy's own defects. Human oversight likewise cannot substitute for the exercise of judgment: an apparent controller can become an instrument of purposes they have ceased to examine. Responsible governance therefore requires continued critical challenge and the intellectual and moral humility to recognize possible flaws in our machines, rules, policies, and ourselves. The impossibility of a general assurance of perfection is a condition responsible practice must confront, not a responsibility that another guardrail—or another guard—can remove.

Keywords: decidability; output policies; large language models; guardrails; responsibility

1 Introduction

Any general assurance of a perfect guardrail is simply an impossibility. Particular policies can be enforced exactly, but that does not support a general promise of perfection. Nor does blocking more outputs, by itself, settle the safety question: withholding permissible information may prevent us from recognizing a danger or remedying a policy's defects. We argue that these limits demand precision in how we talk about the technology and both individual and institutional responsibility for managing its risks. Neither promises of perfection, nor the impossibility of fulfilling them, can discharge that responsibility for any of us.

Language models are rarely deployed alone. Their outputs pass through guardrails, which may include lists of banned phrases, classifiers trained to recognize harmful content, a second

model asked to judge whether an output is acceptable, and, for models that call tools or run code, a monitor that inspects each action and can stop the run. Guardrails make mistakes in both directions. Forbidden outputs get through, often after a user rephrases a prompt for that purpose, and permitted outputs get blocked. Anyone who builds, buys, regulates, or relies on these systems needs to know whether those mistakes are defects that better engineering will fix or whether, for some policies, no guardrail can avoid them.

Guardrails are asked to prevent forbidden outputs while preserving what a language model may legitimately provide. These are distinct demands. A guard that blocks everything satisfies the first; a guard that passes everything satisfies the second. The question is when a computable guard can satisfy both, and what follows when it cannot. We call that combination exactness. We use "guard" in the title in two senses: the computable guard of our formal model, and the person or institution that stands guard over a deployed system; we argue that neither, by its presence alone, discharges responsibility for what a guardrail enforces.

The answer depends on the policy and on which outputs the generator can produce. If every string is a possible output, an exact guard exists if and only if the set of forbidden outputs is decidable (Theorem 1). Under that assumption, whether a perfect guardrail exists depends only on the policy, and no improvement in how the guard is built can change the answer. A guard that is itself a language model faces the same limit. A policy defined by a regular set of banned phrasings is decidable. Idealized policies that judge an output by its truth, by what it does when run, or by what it would lead an exact interpreter of instructions to do are undecidable (Section 3). For those policies, when every string is a possible output, every guard gives the wrong verdict on infinitely many outputs, and every guard that blocks all forbidden outputs also blocks infinitely many permitted ones.

The results have practical consequences, which we develop in later sections. For an undecidable policy, the trade-off between blocking too much and passing too much cannot be engineered away. Whoever sets the policy has to choose where that trade-off falls and should say so openly (Sections 5.1 and 5.2). A requirement that a system never output X, where X is defined by meaning or effect, cannot be met exactly by inspecting outputs. It is better written as a tolerated error rate, under a stated distribution of use and a stated adversary (Sections 2.4 and 5.2). Where exact enforcement matters, it may be available elsewhere in the system. A monitor that observes every action of a running program, and can stop each one, exactly enforces any policy that names a non-empty set of harmful actions (Sections 3.5 and 4.2). That favors controlling an agent's actions at the boundary of the system over judging, in advance, the text that requests them. Finally, a policy becomes a well-defined set only once someone specifies it, and how it is specified determines whether it can be enforced exactly (Section 5.1). None of this shows that guardrails are ineffective. The results give no positive lower bound on average error, and a guardrail can still be accurate enough to be useful in ordinary use (Section 2.4).

We bring guards, monitors, and several semantic policies into a common account of this boundary. We also identify cases in which exactness remains possible, including bounded outputs and fully observed, interceptable actions. Those distinctions matter to policy: a guarantee of compliance with a specified rule does not by itself establish the safety of the wider activity, including the consequences of information withheld. Our ethical argument concerns that further responsibility; we do not treat it as an additional mathematical consequence.

We have a proof to offer, but as researchers offering modest contributions to science, engineering, and policy, we must answer not just for the shape of our proof but for how precisely we state what it warrants and implies. We must consider what follows from the proof, what cannot follow from it, and what *should* not follow. Its conclusions can inform the design, deployment, and governance of computational systems, and we have a responsibility that they do so accurately.

Basis and related work. An earlier version of the mathematical argument appears in Long's "No Perfect Guardrail" [1]. The results are consequences of the undecidability of the halting problem [2], Gödel's first incompleteness theorem [3], and Rice's theorem [4]. We state what these theorems imply for guardrails on the outputs of a language model, under what assumptions, and what they do not imply. Taken one at a time, several of the results are close to known ones; we offer a characterization rather than a single undecidability result. For a full generator, an exact guard exists exactly when the policy is decidable, and if the policy is undecidable every sound guard excludes infinitely many permitted outputs. We apply it to truth as well as to programs, judge programs by the actions they perform rather than the results they return, distinguish guards from monitors, and try to be precise about where exactness remains possible. Glukhov et al. [5] apply Rice's theorem to the censorship of outputs that describe programs and conclude that the set of permissible outputs is undecidable, which gives the necessity half of Theorem 1(a) for one class of policies. They also show that censorship fails for outputs of bounded length when the recipient can invert an encoding. This is consistent with Proposition A1 because their policy depends on what the recipient can decode, while ours is a fixed set of strings. Section 3.4 treats the recipient separately. Their compositional Mosaic attacks, which combine several permitted outputs, are outside our analysis. Hamlen et al. [6] show that a property of programs is enforceable by static analysis if and only if it is decidable, and Theorem 1(a) is the corresponding statement for guards that see the prompt. Our treatment of monitors follows Schneider [7], Viswanathan [8], who required that the test a monitor applies be computable, and Hamlen et al. [6]. Monitors that may suppress or insert actions enforce a larger class of policies [9]. Cohen [10] showed that no algorithm decides whether a program is a virus, and Adleman [11] formalized the result. Alfonseca et al. [12] reduce the halting problem to the question of whether a machine will harm humans, the reduction behind Corollaries 1 and 2. Melo et al. [13] apply Rice's theorem to the question of whether a model is aligned and propose a judge that masks harmful outputs, and Theorem 1(a) states when such a judge can be exact. Xu et al. [14] show that a computable model must err against some computable ground truth, and Kalai and Vempala [15] give a statistical lower bound on the rate of hallucination. Ball et al. [16] show, under cryptographic assumptions, that efficient filters need not exist even where exact ones do, which bears on the question of cost that Section 2.4 leaves open. Theorem A2(b) is an instance of the result of de Leeuw et al. [17] that machines with random choices and bounded error decide only decidable sets.

Our ethical analysis also examines the safety consequences of conservative exclusion (attempting to guarantee safety by being more cautious and excluding more potential dangers). Hamlen et al. [6] recognize that conservative enforcement may reject policy-compliant programs, and Glukhov et al. [5] discuss restrictions that sacrifice usefulness to secure enforceable constraints. We ask what follows when the excluded possibilities may themselves be needed for safety. The censorship of permissible outputs may unforeseeably withhold the means to recognize a danger, or to remedy a

policy's defects. The justification of a safety claim, and responsibility for its realization, would therefore extend beyond mere issues of enforceability to the adequacy of the policy and the consequences of any exclusions enforced by its implementation.

Since no guard can enforce an undecidable policy exactly, such a policy should be written on the assumption that whatever enforces it will make errors. A well-written policy says which errors it tolerates, at what rate, under what use and against which adversary, and who reviews the cases it gets wrong. That recommendation is normative. The theorems show that the errors exist, and they leave the choice of which errors to accept to the people who write the policy.

2 General Results and Their Scope

We introduce the formal model and state the general results for guards, which examine outputs before use, and monitors, which examine actions during execution. We give the proof of the central guard theorem here; supporting facts, further results, and proofs omitted or sketched in the body appear in the Appendix. We then explain the scope and limitations of these results. In Section 3, we apply them to particular policies.

2.1 Model

Notation. Σ is a finite alphabet with at least two symbols, Σ^* is the set of finite strings over Σ, ε is the empty string, $|w|$ is the length of the string w, and $\Sigma^{\leq n}$ is the set of strings of length at most n. A *language* is a subset of Σ^*. The *complement* of a language B is $\overline{B} = \Sigma^* \setminus B$, and the *symmetric difference* of languages B and C is $B \bigtriangleup C = (B \setminus C) \cup (C \setminus B)$. We write $\mathbb{N} = \{0, 1, 2, \dots\}$.

We assume the definition of a Turing machine [18], and fix a standard coding under which every string e is the code of a machine M_e [19]. We write $M_e(x)\downarrow$ if M_e halts on input x, and $M_e(x)\uparrow$ if it does not. A function is *computable* if some Turing machine halts on every input x with output $f(x)$; arguments and values that are not strings are coded as strings. A language B is *decidable* if the function that is 1 on B and 0 on $\overline{B}$ is computable, and *undecidable* otherwise. It is *recursively enumerable* if it is the set of inputs on which some Turing machine halts. A language B is *many-one reducible* to a language C, written $B \leq_{\mathrm{m}} C$, if there is a computable function f such that $x \in B$ if and only if $f(x) \in C$, for every x. The *halting set* is $K = \{e : M_e(e)\downarrow\}$.

Definition 1 (Generator, policy, guard)**.** A *generator* is a computable function $G : \Sigma^* \times \Sigma^* \to \Sigma^*$. The string $G(u, r)$ is the output produced in response to the *prompt* u when the random choices of the generator are determined by the *seed* r. The set of possible outputs on prompt u is $\mathrm{Out}_G(u) = \{G(u, r) : r \in \Sigma^*\}$, and Out_G is the union of these sets over all prompts. The generator is *full* if $\mathrm{Out}_G = \Sigma^*$.

A *policy* is a language B, the set of *forbidden* outputs. Outputs in $\overline{B}$ are *permitted*. A *guard* is a computable function $F : \Sigma^* \times \Sigma^* \to \{0, 1\}$. It *blocks* the output w on prompt u if $F(u, w) = 1$ and *passes* it if $F(u, w) = 0$. With respect to G and B, a guard is *sound* if it blocks every forbidden output of G on every prompt, *permissive* if it passes every permitted output of G on every prompt, and *exact* if it is both.

A guard is the formal counterpart of a guardrail, and an exact guard is a perfect one. The guard that blocks every output is sound, and the guard that passes every output is permissive. The question is whether one guard can be both. We take each policy as a given set; how a policy is specified is a

separate question. A generator models a generation procedure that terminates on every prompt and seed. A guard may itself be a language model.

2.2 Guards

Theorem 1. *Let G be a full generator and B a policy.*

(a) *An exact guard exists if and only if B is decidable.*

(b) *If B is undecidable, every guard gives the wrong verdict on infinitely many outputs; every sound guard blocks infinitely many permitted outputs; and every permissive guard passes infinitely many forbidden outputs.*

Proof. Since $\mathrm{Out}_G = \Sigma^*$, the following search halts for every string w: list the pairs (u, r) in a fixed order and stop at the first with $G(u, r) = w$. Let $\pi(w)$ be the prompt of that pair. Then π is computable and $w \in \mathrm{Out}_G(\pi(w))$. For a guard F let $D_F = \{w : F(\pi(w), w) = 0\}$, which is decidable. If F is sound then $D_F \subseteq \overline{B}$, and if F is permissive then $\overline{B} \subseteq D_F$.

(a) If B is decidable, the guard that blocks w if and only if $w \in B$ is exact. If F is exact then $D_F = \overline{B}$, so B is decidable.

(b) Let $E = D_F \bigtriangleup \overline{B}$, so that $\overline{B} = D_F \bigtriangleup E$. If E were finite it would be decidable, and then $\overline{B}$ would be decidable. So E is infinite. On the prompt $\pi(w)$, each $w \in E$ is either forbidden and passed, or permitted and blocked. If F is sound then $E = \overline{B} \setminus D_F$, and if F is permissive then $E = D_F \cap B$.

□

The theorem has the following limits. To conclude that no exact guard exists, the generator need not be full: it suffices that K is many-one reducible to B or to $\overline{B}$ by a function whose every value the generator can emit. Part (b) counts strings and does not bound an average error (Section 2.4).

Three further consequences are stated and proved as Theorem A2 in Section 4 of the Appendix: if B is infinite, no algorithm decides from the code of a guard whether the guard is sound; if B is undecidable, every randomised guard gives the wrong verdict with probability at least $\frac{1}{2}$ on some output; and if $\overline{B}$ is not recursively enumerable, every generator that emits only permitted strings fails to emit infinitely many of them.

2.3 Monitors

A guard examines a program before it is run. A monitor examines the actions of a program while it runs, and can stop the program.

An output that is run as a program has effects while it runs, and not only a final result. Fix a finite non-empty alphabet A of *actions*, such as commands to a device, calls to an operating system, and symbols written to a file. Let A^* be the set of finite sequences of actions, A^ω the set of infinite sequences, and $A^{\leq\omega} = A^* \cup A^\omega$.

Definition 2 (Trace policy, monitor)**.** A *trace policy* is a set $\mathrm{Bad} \subseteq A^*$ that does not contain the empty sequence and that contains sz whenever it contains s. A finite trace is *forbidden* if it is in Bad and *permitted* otherwise. A *monitor* is a computable function $\mu : A^* \to \{0, 1\}$. When a program run under the monitor has performed the sequence s and is about to perform the action a, the program

is stopped if $\mu(sa) = 1$, and a is performed if $\mu(sa) = 0$. A monitor is *sound* if $\mu(sa) = 1$ whenever s is permitted and sa is forbidden, *permissive* if $\mu(s) = 0$ whenever s is permitted, and *exact* if it is both.

Theorem 2. *An exact monitor exists if and only if* Bad *is decidable. If* Bad *is undecidable, then under every sound monitor there are infinitely many permitted finite traces that cannot be performed in full, and under every permissive monitor there are infinitely many forbidden finite traces that can.*

The proof, in the Appendix, follows that of Theorem 1: the finite traces that a monitor allows to be performed in full form a decidable set, and if the monitor is exact this set is the complement of Bad.

A trace policy corresponds to a safety property in the sense of Schneider [7], who showed that a monitor that can only stop a program enforces no other kind of policy. The condition that Bad be decidable corresponds to the condition that Hamlen et al. [6] place on the test that a monitor applies to finite sequences of events.

2.4 Scope of the general result

Bounded outputs. A language model that stops at a limit of length is a generator, and it is not full. If the outputs of a generator are in $\Sigma^{\leq n}$, the set of its forbidden outputs is finite, so it is decidable, and an exact guard exists. Theorem 1 does not apply. For an undecidable policy B, each language $B \cap \Sigma^{\leq n}$ is decidable, but no algorithm produces from n a machine that decides it (Proposition A1 in Section 4 of the Appendix).

The proposition does not say that a particular bounded system lacks an exact guard, or that its exact guard is costly. Fullness is an additional assumption about which outputs the generator can produce; allowing unlimited memory does not establish it.

Average error. Theorem 1(b) counts strings. It gives no positive lower bound on the average error of a guard under a probability distribution on outputs. Theorem A2(b) is a different statement: it bounds the error of a randomised guard on one particular output. Let P be a probability distribution on Σ^* and let $\delta > 0$. There is a finite set U of strings with $P(U) > 1 - \delta$. The guard that passes the permitted members of U and blocks every other string is sound, and the probability that it blocks a permitted output drawn from P is less than δ. This holds for every policy. If the policy is undecidable, no single algorithm produces such a guard from every finite set U, since applied to $U = \{w\}$ it would determine whether w is permitted.

Adversaries. A small average error under P does not bound the loss when outputs are chosen by an adversary. Suppose that a use of the generator is made by an adversary with probability α; that the adversary then obtains, with probability σ, a forbidden output that the guard passes; that the loss from such an output is at least λ; and that no loss is negative. Then the expected loss in one use is at least $\alpha\,\sigma\,\lambda$, whatever the average error under P. These quantities are assumed and are not derived from Definition 1, and the bound holds for decidable policies also. If the guard is sound then $\sigma = 0$; that $\sigma = 0$ for a particular adversary does not imply that the guard is sound.

3 SPECIAL CASES

One might hope that the general result is loose in practice: that the policies we actually care about are decidable, so that perfect guards for them exist. We test that hope on policies of increasing depth: what an output says (its form, Section 3.1), whether it is true (Section 3.2), what it does when run as a program (Section 3.3), and what it leads a recipient to do (Section 3.4). The unrestricted semantic policies considered below are undecidable; the policy on form is decidable. Restricted semantic cases can also be decidable, as the qualifications below explain. In Section 3.5, we ask the same question of monitors, distinguishing interception of a specified action from classification of a delivered string under an undecidable policy. We summarise the results in Table 1.

3.1 Form: banned phrasings

Let L be a regular language of banned phrasings, and let B_L be the set of strings in which some member of L occurs as a contiguous part. Then B_L is regular, and a finite automaton decides whether $w \in B_L$ in $|w|$ steps [18]. By Theorem 1(a) an exact guard exists. A finite list of banned phrases is a special case.

Regular languages are Type 3 of the hierarchy of Chomsky [20]. A list of banned words takes no account of context. A regular language can take some: the set of strings in which a given word occurs without a second given word immediately after it is regular. Other dependence on context is not regular. The set of strings over an opening and a closing bracket in which the brackets are correctly nested is context-free (Type 2) and not regular. Every context-free language is decidable [18].

3.2 Truth: false sentences of arithmetic

A *sentence of arithmetic* is a sentence of the first-order language with the symbols 0, S (successor), $+$, $\times$, and $<$. It is *true* if it holds in $\mathbb{N}$. Fix a standard effective Gödel coding of formulas by strings [3]. Let Sent be the set of strings that code sentences, and $\mathrm{Th}(\mathbb{N}) \subseteq \mathrm{Sent}$ the set of strings that code true sentences. Sent is decidable, and the code of the negation of a sentence is computable from the code of the sentence. A policy B is a *falsehood policy* if $B \cap \mathrm{Sent} = \mathrm{Sent} \setminus \mathrm{Th}(\mathbb{N})$, that is, if among the sentences of arithmetic it forbids every false one and no true one.

Theorem 3. *No falsehood policy is decidable, and none is recursively enumerable or has a recursively enumerable complement.*

The proof is given in the Appendix. By Gödel's first incompleteness theorem [3], no machine lists the true sentences of arithmetic, and a machine that listed either the forbidden or the permitted strings of a falsehood policy would yield such a machine.

The theorem concerns a policy that classifies every sentence of arithmetic correctly. It does not say that every policy about false statements is undecidable: truth in the first-order arithmetic of addition alone is decidable [21]. For Theorem 1 to apply it suffices that the generator can emit the code of every sentence. By Theorem A2(c), a generator that emits only true sentences of arithmetic fails to emit infinitely many true sentences.

3.3 Behavior: programs

Definition 3 (Action machine, trace, behavior)**.** An *action machine* is a deterministic Turing machine with an input tape, work tapes, and a write-only action tape over A on which the head

moves only to the right. To write a symbol on the action tape is to perform that action. Fix a standard coding under which every string p is the code of an action machine N_p. We call p a *program*. The *trace* $\mathrm{tr}_p(x)$ is the finite or infinite sequence of actions that N_p performs on input x, in order. It is defined whether or not N_p halts. The *behavior* of p is the function $\beta_p : \Sigma^* \to A^{\leq\omega}$ that maps x to $\mathrm{tr}_p(x)$. The *silent behavior* is the behavior that performs no action on any input. A policy B is *behavioral* if $\beta_p = \beta_{p'}$ implies that p and p' are both in B or both outside it.

This is a simplified trace-based model of programs, following Hamlen et al. [6]. A behavioral policy judges a program by the actions it performs and by nothing else. A policy that forbids programs that perform a given action within a given number of steps is not behavioral, and it is decidable by running the program for that number of steps.

Theorem 4 (Rice's theorem for traces)**.** *Let B be a behavioral policy that is neither empty nor all of Σ^*. If no program with the silent behavior is in B, then $K \leq_{\mathrm{m}} B$. Otherwise $K \leq_{\mathrm{m}} \overline{B}$. In both cases B is undecidable.*

Rice's theorem [4] is the same statement for Turing machines that are judged by the result they return on each input on which they halt. As stated it does not mention actions, and a program can perform a harmful action and never return a result. We therefore state the theorem for traces. The proof, given in the Appendix, is that of Rice's theorem. From e one computes a program that first simulates M_e on input e and then, if the simulation halts, behaves like a fixed program $q \in B$. It has the behavior of q if $e \in K$ and the silent behavior otherwise. The argument applies to traces because the program performs no action while it waits.

Corollary 1. *Let $H \subseteq A$ be a non-empty set of harmful actions, let $x_0 \in \Sigma^*$, and let HARM_H be the set of programs p such that an action in H occurs in $\mathrm{tr}_p(x_0)$. Then HARM_H is recursively enumerable, $K \leq_{\mathrm{m}} \mathrm{HARM}_H$, HARM_H is undecidable, and its complement, the set of safe programs, is not recursively enumerable. The same holds for the set of programs that perform an action in H on some input.*

Proof. HARM_H is behavioral. It contains no program with the silent behavior, and it contains a program that performs an action in H on every input. Apply Theorem 4 and fact (F3) of Section 1 of the Appendix. A machine that runs N_p on x_0 and halts when an action in H is performed halts exactly on the members of HARM_H. For the last statement, the machine runs N_p on all inputs in turn, for increasing numbers of steps. □

Restricted programs. Suitable restrictions on the programs that a generator emits can restore decidability. If every program emitted halts within a number of steps computable from its text, membership in HARM_H is decidable for those programs by running them. A restriction on the format of programs that does not limit what they compute does not have this effect.

3.4 Effect on a recipient: an idealized recipient

An output may do harm through whoever receives it. A *recipient* is a function $\rho : \Sigma^* \to A^{\leq\omega}$, where $\rho(w)$ is the sequence of actions that the recipient performs upon receiving the message w. The recipient is not assumed to be the behavior of any program. For a non-empty set $H \subseteq A$ of harmful

actions, the policy *induced* by ρ is the set $B_H(\rho)$ of messages w such that an action in H occurs in $\rho(w)$. A recipient is *instructable* if there is a computable function ι such that $\rho(\iota(p)) = \mathrm{tr}_p(\varepsilon)$ for every program p. The message $\iota(p)$ instructs the recipient to carry out the program p, and the recipient does so exactly.

Corollary 2. *If ρ is instructable, then $K \leq_{\mathrm{m}} B_H(\rho)$. Hence $B_H(\rho)$ is undecidable and the set of harmless messages is not recursively enumerable. If in addition ρ is the behavior β_c of some program c, then $B_H(\rho)$ is recursively enumerable.*

The proof, in the Appendix, composes the encoding ι with the reduction in the proof of Theorem 4: the message that instructs the recipient to carry out a program leads to a harmful action exactly when that program performs one on the empty input.

An instructable recipient is an interpreter of programs that is exact and unlimited in time and memory. The corollary applies to a person only if there is an encoding ι of instructions that the person carries out in this way. It is not a theorem about persuasion.

3.5 Monitoring a running program

For a monitor, the condition of Theorem 2 is that Bad be decidable. Whether it holds depends on what the monitor observes and on what counts as the violation. There are two cases.

The violation is an action. Let $H \subseteq A$ be a non-empty set of harmful actions, and let Bad be the set of finite traces in which an action in H occurs. This set is decidable. The monitor that stops a program exactly when its next action is in H is exact. By Corollary 1 no guard decides from the text of a program whether the program will perform an action in H. The monitor prevents every such action and stops no program whose trace is permitted. This requires that the harmful actions be specified as a set of symbols, and that the monitor observe every action and be able to stop it.

The violation is the delivery of a string. Suppose that $\Sigma \subseteq A$ and that A contains a symbol ! that is not in Σ. The action ! *delivers* the string of symbols of Σ written since the last action that was not in Σ, or since the start if there was none: it hands that string to a recipient who reads it, or to a machine that runs it. Let B be a policy on delivered strings, and let Bad_B be the set of finite traces in which a string in B has been delivered.

Proposition 1. $B \leq_{\mathrm{m}} \mathrm{Bad}_B$. *Hence if B is undecidable, no monitor is exact with respect to* Bad_B.

Proof. For every string w, the trace $w!$ is in Bad_B if and only if $w \in B$. □

In this case the monitor must decide, before a string is delivered, whether the string is in B. Some harmful programs perform only operations that are individually ordinary, such as writing a file, and the harm is in what is written. If the violation is not the delivery but a later action of whoever receives the string, and that action is observed by a monitor, the first case applies to that action.

3.6 Summary

Table 1 collects the results. If every string is a possible output, an exact guard is a decision procedure for the set of forbidden outputs. There is none for a policy that forbids every false sentence of arithmetic and no true one; for a behavioral policy on programs that forbids some programs and permits others; or for a policy that forbids the messages that lead an instructable recipient to a harmful action. The same holds for a monitor that must decide, before a string is delivered, whether the string is forbidden by such a policy. The results do not exclude exact guards for outputs of bounded length or for suitably restricted programs, or exact monitors for policies on observed actions, and they do not bound average error under an arbitrary probability distribution on outputs.

Table 1: Summary of the results. The general results reduce the existence of an exact guard or monitor to the decidability of the policy that it enforces; the specific cases place particular policies on either side of that boundary.

Mechanism and policy	Decidable?	Consequence	Result
General results			
Guard on a full generator; any policy B	Depends on B	An exact guard exists if and only if B is decidable; otherwise every guard gives the wrong verdict on infinitely many outputs	Thm. 1
Monitor; any trace policy Bad	Depends on Bad	An exact monitor exists if and only if Bad is decidable	Thm. 2
Specific policies			
Banned phrasings (a regular language)	Yes	An exact guard exists	§3.1
False sentences of arithmetic	No; neither the policy nor its complement is r.e. (Gödel)	No exact guard	Thm. 3
Behavioral policy on programs, neither empty nor all of Σ^*	No (Rice's theorem for traces)	No exact guard	Thm. 4
Programs that perform a harmful action	No; the harmful programs are r.e., the safe ones are not	No exact guard	Cor. 1
Messages that lead an instructable recipient to a harmful action	No	No exact guard	Cor. 2
Monitor; the violation is an observed action	Yes	An exact monitor exists	§3.5
Monitor; the violation is the delivery of a string in an undecidable B	No	No exact monitor	Prop. 1
Where exactness is not excluded			
Outputs of bounded length	Yes for each bound, but no algorithm produces the decider from the bound	An exact guard exists for each bound	Prop. A1
Programs that halt within a number of steps computable from their text	Yes	Harmfulness is decided by running them	§3.3
Average error under a distribution P on outputs	Not required	For every $\delta > 0$, some sound guard blocks a permitted output with probability less than δ; for an undecidable policy, no single algorithm produces such guards from every finite set of outputs	§2.4

4 Interpreting the Guarantees

Guardrails are asked to do more than act as a "nice to have" watchdog for a language model. We now consider a technical interpretation of guardrails and what they can be asked to do, and how we should view the guarantees they can provide.

4.1 Control and content

Guardrails are asked to preserve the authority of intended instructions when the material being processed can influence the processing itself. Prompt injection makes this difficulty visible: content supplied for a task can be treated as instructions governing the task. The National Cyber Security Centre distinguishes this problem from injection attacks whose instruction–data boundary can be enforced by the execution mechanism [22].

That responsiveness is also a feature of the tool. We ask language models to interpret unfamiliar instructions and arguments, and to develop approaches not specified beforehand. Restricting what may influence their operation can restrict what they help us discover. The comparison with self-modifying code concerns changes in effective instructions and subsequent behavior; it does not require inference to rewrite model weights or executable code. The security literature offers a related caution: return-oriented programming and "weird machines" show that a system can be driven to compute through behaviors outside its intended interpretation [23, 24]. An output need not be written as a program to be run as one, so policies that judge outputs by what they do when run (Section 3.3) can apply to outputs that were never intended as code.

These observations explain the relevance of semantic output policies. They do not establish the assumptions of an undecidability reduction. In particular, an agent that reads documents is not thereby the instructable recipient of Corollary 2: that idealisation faithfully executes every encoded program without limits on time or memory. Susceptibility to prompt injection or human persuasion is a different claim. Empirical work on sycophancy and conversational persuasion gives separate reasons to examine how language models interact with human judgment [25, 26]; it does not establish the recipient idealisation.

4.2 Where guarantees remain possible

Theorem 1 rules out exact discrimination for an undecidable policy on a full generator. It does not rule out sound guards, useful restrictions, or proofs of particular safety properties. A sound guard may block every output. A less restrictive one may admit a verified subset. Under the theorem's assumptions, however, every sound guard excludes infinitely many permitted outputs. Whether those exclusions are acceptable is not settled by soundness.

This boundary is consistent with formal verification. A proof of a particular system, such as a verified operating-system kernel [27], is not a procedure that decides the correctness of every unrestricted program. Finite-state model checking [28], supplied proofs, and restrictions on the programs or properties under consideration can establish specific guarantees. Abstract interpretation can instead provide sound approximations that admit false alarms [29]. Automating code production does not remove the need to establish what such verification covers [30].

The same care is needed when relating the model to deployment. A fixed output-length bound admits an exact guard in principle. For an undecidable policy, no algorithm constructs these

exact guards uniformly from the bound, but this does not show that a particular bounded guard is unavailable or costly. Results on saturated and log-precision transformers demonstrate how computational power depends on architectural and arithmetic assumptions [31, 32]. Idealized models with unbounded context or external memory can support universal computation [33, 34]; this alone does not establish that a generator can emit every string, or the family of strings needed for a particular reduction. Those reachability assumptions must be justified for the claim being made.

Monitoring also changes the question. In Section 3.5, we give an exact monitor for specified actions that are fully observed and interceptable. When the violation instead consists in delivering a string forbidden by an undecidable policy, the classification problem remains. Confinement and side-channel research [35, 36] gives reasons to examine whether the observation and interception assumptions hold; it does not establish that every possible confinement mechanism must fail.

4.3 What evidence must establish

The infinite-error result supplies no positive lower bound on average error under an arbitrary output distribution, nor does it determine the likelihood or cost of a successful attack. Measurements of those quantities can therefore matter greatly to deployment decisions without refuting the theorem. Likewise, probes and representation-level interventions can provide evidence about particular systems [37, 38]. Work on the empirical and conceptual limitations of lie detectors cautions against treating such evidence as a general guarantee [39]. A claim that a detector covers unanticipated forms of deception needs support beyond its observed successes. Rice's theorem cannot supply that conclusion merely from the word "deception": the relevant property and class of computations must first be specified.

The distinction is between establishing a guarantee under stated assumptions and justifying reliance on that guarantee in an application. In the next section, we address that further responsibility, including the consequences of what a safeguard prevents.

5 Ethical and Policy Implications

Our responsibility in presenting an impossibility result includes resisting the temptation to make it settle more than it does. "No exact guard for this policy and output domain" is a mathematical conclusion. "This deployment is safe enough," "this restriction is justified," and "this technology must be prohibited" are judgments requiring further evidence and argument. The proof does not decide them for us.

The ethical obligation is not peculiar to AI. Work on the ethics of belief [40–42] asks how beliefs should be formed, maintained, and acted upon. The modest proposition needed here is that *how one inquires can itself be normatively assessable*. Arriving at a defensible conclusion does not automatically vindicate the process by which one arrived there. Researchers who offer a result for use in engineering and policy must make its assumptions and limits available to those who will rely on it. Those who invoke the result bear a corresponding responsibility to understand what it warrants. In software-intensive science, the grounds for reliance on computational results are themselves a subject of inquiry [43].

That obligation applies to advocates and critics alike. Evidence on motivated reasoning, confir-

mation bias, and selective exposure [44–46] gives reason to scrutinise how congenial conclusions are reached; it does not establish bad faith in any particular actor. Studies of biased assimilation and motivated skepticism show how prior commitments can influence the evaluation of evidence [47, 48]. An impossibility theorem can be recruited to justify a prohibition it does not entail, or dismissed because its consequences are inconvenient. Neither the prospect of benefit nor the fear of harm excuses an unsupported inference. Precision matters because other people may act on the assurance or warning we give them.

5.1 Policy compliance and the consequences of exclusion

The formal model takes a policy as a given set of forbidden outputs. Whether that set adequately captures what ought to be prevented is a further question. AI safety research already recognizes failures of objective specification, including reward hacking and specification gaming [49, 50]. Guaranteed policy compliance is not guaranteed overall safety. A policy can fail to capture what safety requires, and its enforcement can suppress information needed to recognize or respond to danger. For a general-purpose informational tool, these exclusions cannot simply be treated as a loss of usefulness with no bearing on safety. Filtering can suppress the very novelty the tool is intended to enable.

Under the assumptions of Theorem 1, every sound guard for an undecidable policy excludes infinitely many permitted outputs. The theorem counts strings; it does not establish that those strings contain valuable discoveries or safety-critical information. But neither does compliance establish the harmlessness of the exclusions. A research tool might withhold a legitimate hypothesis needed to find a remedy, or a security tool might exclude an unfamiliar approach needed to recognize a vulnerability. These possibilities illustrate why omission belongs in the safety assessment; they are not findings that a particular filter has withheld such information.

It might be argued that, where the stakes are sufficiently high, any possibility of an unsafe outcome would be intolerable. But that concern must also extend to the unforeseeable censorship of permissible outputs: withholding a warning, remedy, or unfamiliar approach may itself be a source of unanticipated danger that the policy does not cover. A demand for safety cannot count the dangers of disclosure while presuming the harmlessness of exclusion. The higher the stakes, the greater our responsibility to examine both. This does not establish that every exclusion is dangerous, or that unrestricted release is safe. It establishes that enforcement of guardrails, no matter how conservative, cannot alone settle the safety question.

Disclosing that filtering occurs does not disclose what has been lost. A user may consent to a restriction without being able to inspect the possibilities it excludes. Such consent does not by itself establish that inaccessible possibilities were expendable, or that a blind spot created in the name of safety is itself safe. Nor does a large volume of useful accepted outputs settle the consequences of what remains unavailable.

The relation to the proof can be stated conditionally. Suppose a proposed guarantee requires a sound guard also to preserve every permitted output whose withholding could cause harm. If one guard must satisfy that requirement regardless of *which* permitted output is indispensable, it must pass every permitted output and is therefore exact. Theorem 1 rules this out for a full generator and an undecidable policy.

This demanding requirement is not necessary for every narrower guarantee. Several outputs

may serve the same need, so a guarantee might require only that *some* output serving each relevant need be passed. A general argument may also cover cases that were not individually foreseen. Such guarantees need not demand flawless prescience, and the theorem does not establish that these weaker requirements are impossible to meet. But the scope of the assurance still needs justification: what needs and circumstances does it cover, and what grounds do we have for treating the exclusions as compatible with safety within that scope?

Conservative exclusion cannot automatically be counted as the harmless side of error. A claim that compliance suffices for safety must explain why the policy and its enforcement are adequate to the circumstances for which that claim is made, including the consequences of withholding information. Presenting an assurance as more comprehensive than its grounds support is an ethical failure of precision; invoking it to abdicate responsibility is a moral failure.

5.2 Making responsibility practicable

The distinction between policymaking and rule enforcement is therefore essential. The policymaker must justify what should be pursued or protected; the rule maker and enforcer translate that purpose into constraints and apply them. Successful enforcement cannot validate the purpose it serves. Likewise, the researcher must establish what a finding warrants, while the user must judge whether a particular application is justified. A research result does not authorise every use, and user approval does not validate the research. These are roles, not fixed identities. Distributing the work must not erase the responsibilities attached to it. The analysis of agency laundering [51] identifies a related wrong: using a technological intermediary to obscure responsibility and obstruct demands for an account.

Keeping a human at the controls is not sufficient. Bainbridge's account of the ironies of automation already showed how automation can leave operators with difficult demands that its design inadequately supports [52]. A person can approve each operation while accepting purposes and conclusions they have ceased to examine. Conversely, substantial delegation can be compatible with responsible reliance when its grounds, scope, and conditions for reconsideration are understood. Research on cognitive offloading and automation bias [53, 54] helps explain why participation alone is an inadequate assurance. Lee et al.'s survey of knowledge workers found an association between confidence in generative AI and reduced self-reported critical-thinking effort [55]; this is evidence about reported practice, not a demonstration of inevitable cognitive decline. The relevant obligation is to exercise judgment appropriate to one's role and capacities, not to reproduce every computation or predict every result. Failure to exercise that judgment can itself be culpable; it does not make responsibility disappear.

Institutions must make this responsibility practicable. An overseer needs enough information, time, and authority to question, refuse, or revise the activity. Assigning a person blame after a failure does not establish that these conditions existed. Developers and deployers remain answerable for the arrangements within which others are asked to rely on their systems.

Elish's "moral crumple zones" and Green's critique of human-oversight policies make this institutional problem explicit [56, 57]. Accounts of meaningful human control connect responsibility to responsiveness to relevant reasons and to the capabilities of the people in the system [58, 59]. Experiments on cognitive forcing also show that interface design can reduce overreliance, with

costs to users' experience and unequal benefits across participants [60]. Responsible oversight therefore requires conditions that can be examined and improved.

A defensible assurance should identify the policy being enforced, the output domain or actions covered, the assumptions about observation and interception, and the evidence for performance beyond a formal guarantee's scope. It should distinguish a proved property from measured reliability, describe known exclusions and uncertainties about their consequences, and say who can respond when those assumptions or consequences are challenged. "Usually works" is an empirical claim to assess, not a substitute for specifying what works and under what conditions.

Policy then has to address the whole application: the harms of use, the harms of restriction or non-use, and the means of detecting and correcting mistakes. This is consistent with a systems approach to safety that examines interactions and organisational arrangements rather than locating every failure in an isolated component [61]. Work on language-model risks and release strategies similarly places deployment in a wider social setting [62, 63]. In practice, this calls for monitoring after deployment, routes for contesting both accepted and refused outputs, and authority to revise or suspend an arrangement when its justification no longer holds. Reviewing exclusions does not require indiscriminate disclosure of dangerous material; it requires ways to examine their consequences without presuming them harmless. The question is: safe from what, for whom, under what circumstances, by what mechanism, with what remaining risk, and who is responsible for responding?

5.3 Humility and revisability

Even exact enforcement of a particular rule leaves someone responsible for choosing it, judging its adequacy, examining its exclusions, and revising it as circumstances change. This obligation survives the cases in which the formal results permit exact guards or monitors. "Guardrails never being enough" is, in this sense, a natural condition to expect and accept. Improving controls remains worthwhile; promising that they can eliminate the need for responsible judgment mistakes their function.

Wiener's warning about the monkey's paw connects literal fulfillment of a wish to the consequences its maker failed to consider. He extends the warning to human institutions, where people can be used as components of a machine rather than treated as responsible human beings [64, pp. 212–213]. The difficulty concerns both the means of carrying out a purpose and our judgment about the purpose itself. We may express what we want imprecisely; we may also come to recognize a flaw in what we sincerely wanted. More faithful execution cannot settle whether that purpose deserves to be pursued.

This matters to the ability to correct policy. A restriction might suppress the warning or unfamiliar argument that would help expose a defect in the restriction itself. The absence of criticism among the outputs it permits would then be poor evidence of its adequacy. This is a possible institutional failure, not a consequence established by the theorem. It gives a further reason to preserve routes of challenge that do not depend solely on what the existing policy allows people to see. Work on epistemic injustice and resistance draws attention to whose testimony is credited and whose interpretations can enter shared inquiry [65, 66]. Providing a formal route for objection is insufficient if those who recognize a defect cannot make themselves heard.

Revisability requires neither accepting every objection nor assuming every change is an im-

provement. It requires that no policy acquire immunity from reconsideration merely because it has been precisely expressed or successfully enforced. A narrower guarantee can be valuable without settling the worth of the whole arrangement. Decisions to deploy, restrict, or refrain must remain answerable to evidence and to those who bear their consequences. Intellectual ability alone does not provide immunity from myside bias [67].

6 CONCLUDING REMARKS

Our formal results establish when exact enforcement is possible within the models we study, and when it is not. Our broader ethical and policy arguments concern the responsibility that remains in either case.

The impossibility of a general assurance of perfect enforcement is itself a reason that responsibility cannot be discharged by invoking guardrails. Promising such perfection is an ethical failure of precision; invoking either that promise or the impossibility of fulfilling it to abdicate responsibility is a moral failure. Responsible AI policy demands neither doomsaying nor demands for the impossible. It demands precision in our discourse, and both individual and institutional responsibility for managing risks that cannot be wished away. Even exact enforcement cannot, by itself, assure the realization of a policy's purposes or establish their adequacy. Continued critical challenge is both an essential element of good policy and a moral imperative.

We must cultivate and maintain the intellectual and moral humility to accept that flaws can exist in our machines, our rules, our policies, or even in our very selves. When we lose that humility and the willingness to repair any part of our overall system, we invite a humbling lesson in the costs of our own blindness. That obligation for precision and humility applies to the claims we make in this paper as well as to the systems we examine. Good guards can build and maintain good guardrails that support and encourage the exercise of responsibility. But their mere presence can neither establish the safety of our overall systems, nor dismiss each individual's agency and responsibility to seek and maintain that safety.

REFERENCES

[1] D. Long, "No Perfect Guardrail," Blog post, Sept. 29 2026, accessed November 19, 2026. [Online]. Available: https://darrelllong.github.io/blog/2026-09-29-no-perfect-guardrail/

[2] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, 1937, received 28 May 1936.

[3] K. Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I," *Monatshefte für Mathematik und Physik*, vol. 38, pp. 173–198, 1931.

[4] H. G. Rice, "Classes of recursively enumerable sets and their decision problems," *Transactions of the American Mathematical Society*, vol. 74, no. 2, pp. 358–366, 1953.

[5] D. Glukhov, I. Shumailov, Y. Gal, N. Papernot, and V. Papyan, "Position: Fundamental limitations of LLM censorship necessitate new approaches," in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning

Research, vol. 235. PMLR, 2024, pp. 15 767–15 787, earlier preprint: "LLM Censorship: A Machine Learning Challenge or a Computer Security Problem?," arXiv:2307.10719, 2023. [Online]. Available: https://proceedings.mlr.press/v235/glukhov24a.html

[6] K. W. Hamlen, G. Morrisett, and F. B. Schneider, "Computability classes for enforcement mechanisms," *ACM Transactions on Programming Languages and Systems*, vol. 28, no. 1, pp. 175–205, 2006.

[7] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security*, vol. 3, no. 1, pp. 30–50, 2000.

[8] M. Viswanathan, "Foundations for the run-time analysis of software systems," Ph.D. dissertation, University of Pennsylvania, 2000.

[9] J. Ligatti, L. Bauer, and D. Walker, "Edit automata: Enforcement mechanisms for run-time security policies," *International Journal of Information Security*, vol. 4, no. 1–2, pp. 2–16, 2005.

[10] F. Cohen, "Computer viruses: Theory and experiments," *Computers & Security*, vol. 6, no. 1, pp. 22–35, 1987.

[11] L. M. Adleman, "An abstract theory of computer viruses," in *Advances in Cryptology — CRYPTO '88*, ser. Lecture Notes in Computer Science, vol. 403. Springer, 1990, pp. 354–374.

[12] M. Alfonseca, M. Cebrian, A. Fernández Anta, L. Coviello, A. Abeliuk, and I. Rahwan, "Superintelligence cannot be contained: Lessons from computability theory," *Journal of Artificial Intelligence Research*, vol. 70, pp. 65–76, 2021, preprint: arXiv:1607.00913, 2016.

[13] G. A. Melo, M. R. O. A. Máximo, N. Y. Soma, and P. A. L. Castro, "Machines that halt resolve the undecidability of artificial intelligence alignment," *Scientific Reports*, vol. 15, p. 15591, 2025, preprint: arXiv:2408.08995, 2024.

[14] Z. Xu, S. Jain, and M. Kankanhalli, "Hallucination is inevitable: An innate limitation of large language models," arXiv:2401.11817, 2024, revised February 13, 2025 (arXiv:2401.11817v2). [Online]. Available: https://arxiv.org/abs/2401.11817

[15] A. T. Kalai and S. S. Vempala, "Calibrated language models must hallucinate," in *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, 2024, pp. 160–171, preprint: arXiv:2311.14648, 2023.

[16] S. Ball, G. Gluch, S. Goldwasser, F. Kreuter, O. Reingold, and G. N. Rothblum, "On the impossibility of separating intelligence from judgment: The computational intractability of filtering for AI alignment," in *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026, earlier preprint: "The Computational Intractability of Filtering for AI Alignment," arXiv:2507.07341, 2025.

[17] K. de Leeuw, E. F. Moore, C. E. Shannon, and N. Shapiro, "Computability by probabilistic machines," in *Automata Studies*, ser. Annals of Mathematics Studies, C. E. Shannon and J. McCarthy, Eds. Princeton University Press, 1956, vol. 34, pp. 183–212.

[18] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, 1979.

[19] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*. New York: McGraw-Hill, 1967.

[20] N. Chomsky, "On certain formal properties of grammars," *Information and Control*, vol. 2, no. 2, pp. 137–167, 1959.

[21] M. Presburger, "Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt," in *Comptes Rendus du I Congrès des Mathématiciens des Pays Slaves, Warszawa 1929*, F. Leja, Ed. Warsaw: Książnica Atlas, 1930, pp. 92–101, 395.

[22] D. Chismon, "Prompt injection is not SQL injection (it may be worse)," National Cyber Security Centre, December 8 2025, accessed October 21, 2026. [Online]. Available: https://www.ncsc.gov.uk/blog-post/prompt-injection-is-not-sql-injection

[23] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, 2007, pp. 552–561.

[24] T. Dullien, "Weird machines, exploitability, and provable unexploitability," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 391–403, 2020.

[25] M. Sharma, M. Tong, T. Korbak, D. Duvenaud, A. Askell, S. R. Bowman, N. Cheng, E. Durmus, Z. Hatfield-Dodds, S. R. Johnston, S. Kravec, T. Maxwell, S. McCandlish, K. Ndousse, O. Rausch, N. Schiefer, D. Yan, M. Zhang, and E. Perez, "Towards understanding sycophancy in language models," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024, preprint: arXiv:2310.13548, 2023.

[26] F. Salvi, M. Horta Ribeiro, R. Gallotti, and R. West, "On the conversational persuasiveness of GPT-4," *Nature Human Behaviour*, vol. 9, no. 8, pp. 1645–1653, 2025, preprint: "On the Conversational Persuasiveness of Large Language Models: A Randomized Controlled Trial," arXiv:2403.14380, 2024.

[27] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal verification of an OS kernel," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP)*, 2009, pp. 207–220.

[28] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.

[29] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Conference Record of the 4th*

ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL), 1977, pp. 238–252.

[30] J. Symons and J. K. Horner, "Why there is no general solution to the problem of software verification," *Foundations of Science*, vol. 25, no. 3, pp. 541–557, 2020, published online 3 June 2019.

[31] W. Merrill, A. Sabharwal, and N. A. Smith, "Saturated transformers are constant-depth threshold circuits," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 843–856, 2022.

[32] W. Merrill and A. Sabharwal, "The parallelism tradeoff: Limitations of log-precision transformers," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 531–545, 2023.

[33] J. Pérez, P. Barceló, and J. Marinkovic, "Attention is Turing complete," *Journal of Machine Learning Research*, vol. 22, no. 75, pp. 1–35, 2021.

[34] D. Schuurmans, "Memory augmented large language models are computationally universal," arXiv:2301.04589, 2023.

[35] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[36] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.

[37] S. Marks and M. Tegmark, "The geometry of truth: Emergent linear structure in LLM representations of true/false datasets," in *First Conference on Language Modeling (COLM)*, 2024, preprint: arXiv:2310.06824, 2023. [Online]. Available: https://arxiv.org/abs/2310.06824

[38] A. Zou, L. Phan, S. Chen, J. Campbell, P. Guo, R. Ren, *et al.*, "Representation engineering: A top-down approach to AI transparency," arXiv:2310.01405, 2023.

[39] B. A. Levinstein and D. A. Herrmann, "Still no lie detector for language models: Probing empirical and conceptual roadblocks," *Philosophical Studies*, vol. 182, pp. 1539–1565, 2025, published online 17 February 2024; preprint: arXiv:2307.00175, 2023.

[40] W. K. Clifford, "The ethics of belief," *Contemporary Review*, vol. 29, pp. 289–309, 1877.

[41] R. Feldman, "Epistemic obligations," *Philosophical Perspectives*, vol. 2, pp. 235–256, 1988.

[42] ——, "The ethics of belief," *Philosophy and Phenomenological Research*, vol. 60, no. 3, pp. 667–695, 2000.

[43] J. Symons and J. Horner, "Software intensive science," *Philosophy & Technology*, vol. 27, no. 3, pp. 461–477, 2014.

[44] Z. Kunda, "The case for motivated reasoning," *Psychological Bulletin*, vol. 108, no. 3, pp. 480–498, 1990.

[45] R. S. Nickerson, "Confirmation bias: A ubiquitous phenomenon in many guises," *Review of General Psychology*, vol. 2, no. 2, pp. 175–220, 1998.

[46] W. Hart, D. Albarracín, A. H. Eagly, I. Brechan, M. J. Lindberg, and L. Merrill, "Feeling validated versus being correct: A meta-analysis of selective exposure to information," *Psychological Bulletin*, vol. 135, no. 4, pp. 555–588, 2009.

[47] C. G. Lord, L. Ross, and M. R. Lepper, "Biased assimilation and attitude polarization: The effects of prior theories on subsequently considered evidence," *Journal of Personality and Social Psychology*, vol. 37, no. 11, pp. 2098–2109, 1979.

[48] C. S. Taber and M. Lodge, "Motivated skepticism in the evaluation of political beliefs," *American Journal of Political Science*, vol. 50, no. 3, pp. 755–769, 2006.

[49] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," arXiv:1606.06565, 2016. [Online]. Available: https://arxiv.org/abs/1606.06565

[50] V. Krakovna, J. Uesato, V. Mikulik, M. Rahtz, T. Everitt, R. Kumar, Z. Kenton, J. Leike, and S. Legg, "Specification gaming: The flip side of AI ingenuity," DeepMind research blog, April 21 2020. [Online]. Available: https://deepmind.google/blog/specification-gaming-the-flip-side-of-ai-ingenuity/

[51] A. Rubel, C. Castro, and A. Pham, "Agency laundering and information technologies," *Ethical Theory and Moral Practice*, vol. 22, pp. 1017–1041, 2019. [Online]. Available: https://link.springer.com/article/10.1007/s10677-019-10030-w

[52] L. Bainbridge, "Ironies of automation," *Automatica*, vol. 19, no. 6, pp. 775–779, 1983. [Online]. Available: https://doi.org/10.1016/0005-1098(83)90046-8

[53] E. F. Risko and S. J. Gilbert, "Cognitive offloading," *Trends in Cognitive Sciences*, vol. 20, no. 9, pp. 676–688, 2016.

[54] R. Parasuraman and D. H. Manzey, "Complacency and bias in human use of automation: An attentional integration," *Human Factors*, vol. 52, no. 3, pp. 381–410, 2010.

[55] H.-P. H. Lee, A. Sarkar, L. Tankelevitch, I. Drosos, S. Rintel, R. Banks, and N. Wilson, "The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers," in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2025, pp. 1–22, article 1121.

[56] M. C. Elish, "Moral crumple zones: Cautionary tales in human-robot interaction," *Engaging Science, Technology, and Society*, vol. 5, pp. 40–60, 2019. [Online]. Available: https://estsjournal.org/index.php/ests/article/view/260

[57] B. Green, "The flaws of policies requiring human oversight of government algorithms," *Computer Law & Security Review*, vol. 45, p. 105681, 2022. [Online]. Available: https://doi.org/10.1016/j.clsr.2022.105681

[58] F. Santoni de Sio and J. van den Hoven, "Meaningful human control over autonomous systems: A philosophical account," *Frontiers in Robotics and AI*, vol. 5, p. 15, 2018. [Online]. Available: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2018.00015/full

[59] L. Cavalcante Siebert, M. L. Lupetti, E. Aizenberg, N. Beckers, A. Zgonnikov, H. Veluwenkamp, D. Abbink, E. Giaccardi, G.-J. Houben, C. M. Jonker, J. van den Hoven, D. Forster, and R. L. Lagendijk, "Meaningful human control: Actionable properties for AI system development," *AI and Ethics*, vol. 3, pp. 241–255, 2023, first published online in 2022; cited by the 2023 volume year. [Online]. Available: https://link.springer.com/article/10.1007/s43681-022-00167-3

[60] Z. Buçinca, M. B. Malaya, and K. Z. Gajos, "To trust or to think: Cognitive forcing functions can reduce overreliance on AI in AI-assisted decision-making," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW1, pp. 1–21, 2021, article 188. [Online]. Available: https://doi.org/10.1145/3449287

[61] N. Leveson, "A new accident model for engineering safer systems," *Safety Science*, vol. 42, no. 4, pp. 237–270, 2004. [Online]. Available: https://doi.org/10.1016/S0925-7535(03)00047-X

[62] L. Weidinger, J. Mellor, M. Rauh, C. Griffin, J. Uesato, P.-S. Huang, M. Cheng, M. Glaese, B. Balle, A. Kasirzadeh, Z. Kenton, S. Brown, W. Hawkins, T. Stepleton, C. Biles, A. Birhane, J. Haas, L. Rimell, L. A. Hendricks, W. Isaac, S. Legassick, G. Irving, and I. Gabriel, "Ethical and social risks of harm from language models," arXiv:2112.04359, 2021.

[63] I. Solaiman, M. Brundage, J. Clark, A. Askell, A. Herbert-Voss, J. Wu, A. Radford, G. Krueger, J. W. Kim, S. Kreps, M. McCain, A. Newhouse, J. Blazakis, K. McGuffie, and J. Wang, "Release strategies and the social impacts of language models," OpenAI, Tech. Rep., 2019, arXiv:1908.09203. [Online]. Available: https://arxiv.org/abs/1908.09203

[64] N. Wiener, *The Human Use of Human Beings: Cybernetics and Society*. Boston: Houghton Mifflin, 1950, first edition; especially Chapter XI, pp. 212–213.

[65] M. Fricker, *Epistemic Injustice: Power and the Ethics of Knowing*. Oxford University Press, 2007.

[66] J. Medina, *The Epistemology of Resistance: Gender and Racial Oppression, Epistemic Injustice, and the Social Imagination*. Oxford University Press, 2013.

[67] K. E. Stanovich, R. F. West, and M. E. Toplak, "Myside bias, rational thinking, and intelligence," *Current Directions in Psychological Science*, vol. 22, no. 4, pp. 259–264, 2013.

APPENDIX FULLER PROOF AND FURTHER RESULTS

In this appendix, we present the complete formal development in one place. We repeat the notation, definitions, assumptions, results, and proofs needed to read the argument independently of the main text, including the qualifications on bounded outputs, average error, adversaries, and restricted programs. The central guard theorem and the further guard results are stated separately, as in the main text, but all their proofs are included here.

For comparison with the body, Theorem A1 restates Theorem 1; Theorems A3, A4, and A5 restate Theorems 3, 4, and 2, respectively. The repeated definitions, corollaries, and delivery proposition also retain their substance. Appendix results use an A prefix to distinguish them from the results in the body.

1 Notation and standard facts

Notation. Σ is a finite alphabet with at least two symbols, Σ^* is the set of finite strings over Σ, ε is the empty string, $|w|$ is the length of the string w, and $\Sigma^{\leq n}$ is the set of strings of length at most n. A *language* is a subset of Σ^*. The *complement* of a language B is $\overline{B} = \Sigma^* \setminus B$, and the *symmetric difference* of languages B and C is $B \bigtriangleup C = (B \setminus C) \cup (C \setminus B)$. We write $\mathbb{N} = \{0, 1, 2, \dots\}$.

We assume the definition of a Turing machine [18], and fix a standard coding under which every string e is the code of a machine M_e [19]. We write $M_e(x)\downarrow$ if M_e halts on input x, and $M_e(x)\uparrow$ if it does not. A function is *computable* if some Turing machine halts on every input x with output $f(x)$; arguments and values that are not strings are coded as strings. A language B is *decidable* if the function that is 1 on B and 0 on $\overline{B}$ is computable, and *undecidable* otherwise. It is *recursively enumerable* if it is the set of inputs on which some Turing machine halts. A language B is *many-one reducible* to a language C, written $B \leq_{\mathrm{m}} C$, if there is a computable function f such that $x \in B$ if and only if $f(x) \in C$, for every x. The *halting set* is $K = \{e : M_e(e)\downarrow\}$. We use three standard facts [2, 19].

(F1) K is recursively enumerable and undecidable.

(F2) A language is decidable if and only if it and its complement are both recursively enumerable.

(F3) If $K \leq_{\mathrm{m}} C$, then C is undecidable and $\overline{C}$ is not recursively enumerable.

2 Generators, policies, and guards

Definition A1 (Generator, policy, guard)**.** A *generator* is a computable function $G : \Sigma^* \times \Sigma^* \to \Sigma^*$. The string $G(u, r)$ is the output produced in response to the *prompt* u when the random choices of the generator are determined by the *seed* r. The set of possible outputs on prompt u is $\mathrm{Out}_G(u) = \{G(u, r) : r \in \Sigma^*\}$, and Out_G is the union of these sets over all prompts. The generator is *full* if $\mathrm{Out}_G = \Sigma^*$.

A *policy* is a language B, the set of *forbidden* outputs. Outputs in $\overline{B}$ are *permitted*. A *guard* is a computable function $F : \Sigma^* \times \Sigma^* \to \{0, 1\}$. It *blocks* the output w on prompt u if $F(u, w) = 1$ and *passes* it if $F(u, w) = 0$. With respect to G and B, a guard is *sound* if it blocks every forbidden output of G on every prompt, *permissive* if it passes every permitted output of G on every prompt, and *exact* if it is both.

A *randomised guard* is a Turing machine that may toss a fair coin at any step and that, on every input (u, w) and every sequence of outcomes of its tosses, halts with output 0 or 1.

A guard is the formal counterpart of a guardrail, and an exact guard is a perfect one. The guard that blocks every output is sound, and the guard that passes every output is permissive. The question is whether one guard can be both. We take each policy as a given set; how a policy is specified is a separate question. A generator models a generation procedure that terminates on every prompt and seed. A guard may itself be a language model.

3 The central guard theorem

Theorem A1. *Let G be a full generator and B a policy.*

(a) *An exact guard exists if and only if B is decidable.*

(b) *If B is undecidable, every guard gives the wrong verdict on infinitely many outputs; every sound guard blocks infinitely many permitted outputs; and every permissive guard passes infinitely many forbidden outputs.*

Proof. Since $\mathrm{Out}_G = \Sigma^*$, the following search halts for every string w: list the pairs (u, r) in a fixed order and stop at the first with $G(u, r) = w$. Let $\pi(w)$ be the prompt of that pair. Then π is computable and $w \in \mathrm{Out}_G(\pi(w))$. For a guard F let $D_F = \{w : F(\pi(w), w) = 0\}$, which is decidable. If F is sound then $D_F \subseteq \overline{B}$, and if F is permissive then $\overline{B} \subseteq D_F$.

(a) If B is decidable, the guard that blocks w if and only if $w \in B$ is exact. If F is exact then $D_F = \overline{B}$, so B is decidable.

(b) Let $E = D_F \triangle \overline{B}$, so that $\overline{B} = D_F \triangle E$. If E were finite it would be decidable, and then $\overline{B}$ would be decidable. So E is infinite. On the prompt $\pi(w)$, each $w \in E$ is either forbidden and passed, or permitted and blocked. If F is sound then $E = \overline{B} \setminus D_F$, and if F is permissive then $E = D_F \cap B$.

□

The theorem has the following limits. To conclude that no exact guard exists, the generator need not be full: it suffices that K is many-one reducible to B or to $\overline{B}$ by a function whose every value the generator can emit. Part (b) counts strings and does not bound an average error (Section 7). Theorem A2(a) does not say that no guard can be proved sound; the guard that blocks every string is sound. Theorem A2(b) requires that the randomised guard halt on every sequence of tosses.

4 Further results on guards

Bounded outputs. A language model that stops at a limit of length is a generator, and it is not full. If the outputs of a generator are in $\Sigma^{\leq n}$, the set of its forbidden outputs is finite, so it is decidable, and an exact guard exists. Theorem A1 does not apply.

Proposition A1. *Let B be an undecidable policy. For every n the language $B \cap \Sigma^{\leq n}$ is decidable. No algorithm produces, from n, the code of a machine that decides $B \cap \Sigma^{\leq n}$.*

Proof. The language is finite. Given such an algorithm, one would decide whether $w \in B$ by producing the machine for $n = |w|$ and running it on w. □

The proposition does not say that a particular bounded system lacks an exact guard, or that its exact guard is costly. Fullness is an additional assumption about which outputs the generator can produce; allowing unlimited memory does not establish it.

Theorem A2. *Let G be a full generator and B a policy.*

(a) *If B is infinite, no algorithm decides, given the code of a machine that computes a guard, whether the guard is sound.*

(b) *If B is undecidable, for every randomised guard there is an output on which it gives the wrong verdict with probability at least $\frac{1}{2}$.*

(c) *If $\overline{B}$ is not recursively enumerable, every generator all of whose outputs are permitted, whether it is full or not, fails to emit infinitely many permitted strings.*

Proof. (a) For each e let F_e be the guard that, on every prompt, passes w if and only if M_e halts on input e within $|w|$ steps. A code of a machine that computes F_e is computable from e. If $e \notin K$, then F_e blocks every string and is sound. If $e \in K$, then M_e halts on e within some number t of steps, and F_e passes every string of length at least t. Since B is infinite, one of these strings is forbidden, and it is an output of G; so F_e is not sound. An algorithm that decided soundness would decide K.

(b) Let R be a randomised guard and fix (u, w). The computations of R on (u, w) form a tree that branches in two at each toss and has no infinite branch. By König's lemma, which states that a tree with finitely many successors at every node and no infinite branch is finite, the tree is finite. So the probability $b(u, w)$ that R outputs 1 on (u, w) is a rational number computable from (u, w). Suppose that R gave the wrong verdict with probability less than $\frac{1}{2}$ on every output. Then $b(u, w) > \frac{1}{2}$ for every forbidden output w on prompt u, and $b(u, w) < \frac{1}{2}$ for every permitted one. The guard that blocks w on prompt u if and only if $b(u, w) > \frac{1}{2}$ would then be exact, contrary to Theorem A1(a).

(c) Let G' be a generator with $\mathrm{Out}_{G'} \subseteq \overline{B}$. The set $\mathrm{Out}_{G'}$ is recursively enumerable: search for a pair (u, r) with $G'(u, r) = w$. If $\overline{B} \setminus \mathrm{Out}_{G'}$ were finite, $\overline{B}$ would be the union of a recursively enumerable language and a finite one, and so would be recursively enumerable. □

Part (a) does not say that no guard can be proved sound; the guard that blocks every string is sound. Part (b) requires that the randomised guard halt on every sequence of tosses.

5 Policies

5.1 Banned phrasings

Let L be a regular language of banned phrasings, and let B_L be the set of strings in which some member of L occurs as a contiguous part. Then B_L is regular, and a finite automaton decides whether $w \in B_L$ in $|w|$ steps [18]. By Theorem A1(a) an exact guard exists. A finite list of banned phrases is a special case.

Regular languages are Type 3 of the hierarchy of Chomsky [20]. A list of banned words takes no account of context. A regular language can take some: the set of strings in which a given word occurs without a second given word immediately after it is regular. Other dependence on context is not regular. The set of strings over an opening and a closing bracket in which the brackets are correctly nested is context-free (Type 2) and not regular. Every context-free language is decidable [18].

5.2 False sentences of arithmetic

A *sentence of arithmetic* is a sentence of the first-order language with the symbols 0, S (successor), $+$, $\times$, and $<$. It is *true* if it holds in $\mathbb{N}$. Fix a standard effective Gödel coding of formulas by strings [3]. Let Sent be the set of strings that code sentences, and $\mathrm{Th}(\mathbb{N}) \subseteq \mathrm{Sent}$ the set of strings that code true sentences. Sent is decidable, and the code of the negation of a sentence is computable from the code of the sentence. A policy B is a *falsehood policy* if $B \cap \mathrm{Sent} = \mathrm{Sent} \setminus \mathrm{Th}(\mathbb{N})$, that is, if among the sentences of arithmetic it forbids every false one and no true one.

Theorem A3. *No falsehood policy is decidable, and none is recursively enumerable or has a recursively enumerable complement.*

Proof. By Gödel's first incompleteness theorem [3], $\mathrm{Th}(\mathbb{N})$ is not recursively enumerable: if it were, the true sentences would be a set of axioms that can be listed by a machine, that is consistent, that includes the axioms of arithmetic, and that settles every sentence, and the theorem states that there is no such set. Let B be a falsehood policy. If $\overline{B}$ were recursively enumerable, so would be $\mathrm{Th}(\mathbb{N}) = \mathrm{Sent} \cap \overline{B}$. If B were recursively enumerable, so would be the set $\mathrm{Sent} \cap B$ of false sentences; then $\mathrm{Th}(\mathbb{N})$ would be recursively enumerable, because a sentence is true if and only if its negation is false. By (F2) a decidable language is recursively enumerable. □

The theorem concerns a policy that classifies every sentence of arithmetic correctly. It does not say that every policy about false statements is undecidable: truth in the first-order arithmetic of addition alone is decidable [21]. For Theorem A1 to apply it suffices that the generator can emit the code of every sentence. By Theorem A2(c), a generator that emits only true sentences of arithmetic fails to emit infinitely many true sentences.

5.3 Programs

An output that is run as a program has effects while it runs, and not only a final result. Fix a finite non-empty alphabet A of *actions*, such as commands to a device, calls to an operating system, and symbols written to a file. Let A^* be the set of finite sequences of actions, A^ω the set of infinite sequences, and $A^{\leq\omega} = A^* \cup A^\omega$.

Definition A2 (Action machine, trace, behavior)**.** An *action machine* is a deterministic Turing machine with an input tape, work tapes, and a write-only action tape over A on which the head moves only to the right. To write a symbol on the action tape is to perform that action. Fix a standard coding under which every string p is the code of an action machine N_p. We call p a *program*. The *trace* $\mathrm{tr}_p(x)$ is the finite or infinite sequence of actions that N_p performs on input x, in order. It is defined whether or not N_p halts. The *behavior* of p is the function $\beta_p : \Sigma^* \to A^{\leq\omega}$ that maps x to $\mathrm{tr}_p(x)$. The *silent behavior* is the behavior that performs no action on any input. A policy B is *behavioral* if $\beta_p = \beta_{p'}$ implies that p and p' are both in B or both outside it.

This is a simplified trace-based model of programs, following Hamlen et al. [6]. A behavioral policy judges a program by the actions it performs and by nothing else. A policy that forbids programs that perform a given action within a given number of steps is not behavioral, and it is decidable by running the program for that number of steps.

Theorem A4 (Rice's theorem for traces)**.** *Let B be a behavioral policy that is neither empty nor all of Σ^*. If no program with the silent behavior is in B, then $K \leq_{\mathrm{m}} B$. Otherwise $K \leq_{\mathrm{m}} \overline{B}$. In both cases B is undecidable.*

Proof. Suppose that no program with the silent behavior is in B, and let $q \in B$. For each e let $f(e)$ be a code of the action machine that, on input x, first simulates M_e on input e, performing no action, and then, if that simulation halts, performs the actions that N_q performs on x. The function f is computable. If $e \in K$ then $\beta_{f(e)} = \beta_q$, so $f(e) \in B$. If $e \notin K$ then $f(e)$ has the silent behavior, so $f(e) \notin B$. Hence $K \leq_{\mathrm{m}} B$. Otherwise every program with the silent behavior is in B; apply the argument to $\overline{B}$, which is behavioral. Undecidability follows by (F3). □

Rice's theorem [4] is the same statement for Turing machines that are judged by the result they return on each input on which they halt. As stated it does not mention actions, and a program can perform a harmful action and never return a result. We therefore state the theorem for traces. The proof is that of Rice's theorem. It applies to traces because the machine with code $f(e)$ performs no action while it waits.

Corollary A1. *Let $H \subseteq A$ be a non-empty set of harmful actions, let $x_0 \in \Sigma^*$, and let HARM_H be the set of programs p such that an action in H occurs in $\mathrm{tr}_p(x_0)$. Then HARM_H is recursively enumerable, $K \leq_{\mathrm{m}} \mathrm{HARM}_H$, HARM_H is undecidable, and its complement, the set of safe programs, is not recursively enumerable. The same holds for the set of programs that perform an action in H on some input.*

Proof. HARM_H is behavioral. It contains no program with the silent behavior, and it contains a program that performs an action in H on every input. Apply Theorem A4 and (F3). A machine that runs N_p on x_0 and halts when an action in H is performed halts exactly on the members of HARM_H. For the last statement, the machine runs N_p on all inputs in turn, for increasing numbers of steps. □

5.4 An idealized recipient

An output may do harm through whoever receives it. A *recipient* is a function $\rho : \Sigma^* \to A^{\leq\omega}$, where $\rho(w)$ is the sequence of actions that the recipient performs upon receiving the message w. The recipient is not assumed to be the behavior of any program. For a non-empty set $H \subseteq A$ of harmful actions, the policy *induced* by ρ is the set $B_H(\rho)$ of messages w such that an action in H occurs in $\rho(w)$. A recipient is *instructable* if there is a computable function ι such that $\rho(\iota(p)) = \mathrm{tr}_p(\varepsilon)$ for every program p. The message $\iota(p)$ instructs the recipient to carry out the program p, and the recipient does so exactly.

Corollary A2. *If ρ is instructable, then $K \leq_{\mathrm{m}} B_H(\rho)$. Hence $B_H(\rho)$ is undecidable and the set of harmless messages is not recursively enumerable. If in addition ρ is the behavior β_c of some program c, then $B_H(\rho)$ is recursively enumerable.*

Proof. Let f be the reduction of K to HARM_H with $x_0 = \varepsilon$ given by Theorem A4. Then $e \in K$ if and only if $\iota(f(e)) \in B_H(\rho)$. Apply (F3). If $\rho = \beta_c$, a machine that runs N_c on w and halts when an action in H is performed halts exactly on the members of $B_H(\rho)$. □

An instructable recipient is an interpreter of programs that is exact and unlimited in time and memory. The corollary applies to a person only if there is an encoding ι of instructions that the person carries out in this way. It is not a theorem about persuasion.

6 Monitors

A guard examines a program before it is run. A monitor examines the actions of a program while it runs, and can stop the program.

Definition A3 (Trace policy, monitor)**.** A *trace policy* is a set $\mathrm{Bad} \subseteq A^*$ that does not contain the empty sequence and that contains sz whenever it contains s. A finite trace is *forbidden* if it is in Bad and *permitted* otherwise. A *monitor* is a computable function $\mu : A^* \to \{0, 1\}$. When a program run under the monitor has performed the sequence s and is about to perform the action a, the program is stopped if $\mu(sa) = 1$, and a is performed if $\mu(sa) = 0$. A monitor is *sound* if $\mu(sa) = 1$ whenever s is permitted and sa is forbidden, *permissive* if $\mu(s) = 0$ whenever s is permitted, and *exact* if it is both.

Theorem A5. *An exact monitor exists if and only if* Bad *is decidable. If* Bad *is undecidable, then under every sound monitor there are infinitely many permitted finite traces that cannot be performed in full, and under every permissive monitor there are infinitely many forbidden finite traces that can.*

Proof. Let D_μ be the set of finite traces s such that μ is 0 on every non-empty initial segment of s. These are the traces that can be performed in full. D_μ is decidable. If μ is sound, no forbidden trace is in D_μ, because μ is 1 on its shortest forbidden initial segment. If μ is permissive, every permitted trace is in D_μ, because all its initial segments are permitted. So if μ is exact, D_μ is the complement of Bad, and Bad is decidable. Conversely, if Bad is decidable, the function that is 1 on Bad and 0 elsewhere is an exact monitor. The rest follows as in Theorem A1(b). □

A trace policy corresponds to a safety property in the sense of Schneider [7], who showed that a monitor that can only stop a program enforces no other kind of policy. The condition that Bad be decidable corresponds to the condition that Hamlen et al. [6] place on the test that a monitor applies to finite sequences of events. Whether it holds depends on what the monitor observes and on what counts as the violation. There are two cases.

The violation is an action. Let $H \subseteq A$ be a non-empty set of harmful actions, and let Bad be the set of finite traces in which an action in H occurs. This set is decidable. The monitor that stops a program exactly when its next action is in H is exact. By Corollary A1 no guard decides from the text of a program whether the program will perform an action in H. The monitor prevents every such action and stops no program whose trace is permitted. This requires that the harmful actions be specified as a set of symbols, and that the monitor observe every action and be able to stop it.

The violation is the delivery of a string. Suppose that $\Sigma \subseteq A$ and that A contains a symbol ! that is not in Σ. The action ! *delivers* the string of symbols of Σ written since the last action that was not in Σ, or since the start if there was none: it hands that string to a recipient who reads it, or to a machine that runs it. Let B be a policy on delivered strings, and let Bad_B be the set of finite traces in which a string in B has been delivered.

Proposition A2. $B \leq_{\mathrm{m}} \mathrm{Bad}_B$. *Hence if B is undecidable, no monitor is exact with respect to* Bad_B.

Proof. For every string w, the trace $w!$ is in Bad_B if and only if $w \in B$. □

In this case the monitor must decide, before a string is delivered, whether the string is in B. Some harmful programs perform only operations that are individually ordinary, such as writing a file, and the harm is in what is written. If the violation is not the delivery but a later action of whoever receives the string, and that action is observed by a monitor, the first case applies to that action.

7 Limits of the results

Average error. Theorem A1(b) counts strings. It gives no positive lower bound on the average error of a guard under a probability distribution on outputs. Theorem A2(b) is a different statement: it bounds the error of a randomised guard on one particular output. Let P be a probability distribution on Σ^* and let $\delta > 0$. There is a finite set U of strings with $P(U) > 1 - \delta$. The guard that passes the permitted members of U and blocks every other string is sound, and the probability that it blocks a permitted output drawn from P is less than δ. This holds for every policy. If the policy is undecidable, no single algorithm produces such a guard from every finite set U, since applied to $U = \{w\}$ it would determine whether w is permitted.

Adversaries. A small average error under P does not bound the loss when outputs are chosen by an adversary. Suppose that a use of the generator is made by an adversary with probability α; that the adversary then obtains, with probability σ, a forbidden output that the guard passes; that the loss from such an output is at least λ; and that no loss is negative. Then the expected loss in one use is at least $\alpha\,\sigma\,\lambda$, whatever the average error under P. These quantities are assumed and are not derived from Definition A1, and the bound holds for decidable policies also. If the guard is sound then $\sigma = 0$; that $\sigma = 0$ for a particular adversary does not imply that the guard is sound.

Restricted programs. Suitable restrictions on the programs that a generator emits can restore decidability. If every program emitted halts within a number of steps computable from its text, membership in HARM_H is decidable for those programs by running them. A restriction on the format of programs that does not limit what they compute does not have this effect.

Summary. If every string is a possible output, an exact guard is a decision procedure for the set of forbidden outputs. There is none for a policy that forbids every false sentence of arithmetic and no true one; for a behavioral policy on programs that forbids some programs and permits others; or for a policy that forbids the messages that lead an instructable recipient to a harmful action. The same holds for a monitor that must decide, before a string is delivered, whether the string is forbidden by such a policy. The results do not exclude exact guards for outputs of bounded length or for suitably restricted programs, or exact monitors for policies on observed actions, and they do not bound average error under an arbitrary probability distribution on outputs.