Safe Language Generation in the Limit

Anonymous ACL submission

Abstract

Recent results in learning a language in the limit have shown that, although language identification is impossible, language generation is tractable. As this foundational area expands, we need to consider the implications of language generation in real-world settings.

This work offers the first theoretical treatment of *safe* language generation. Building on the computational paradigm of learning in the limit, we formalize the tasks of safe language identification and generation. We prove that under this model, safe language identification is impossible, and that safe language generation is at least as hard as (vanilla) language identification, which is also impossible. Last, we discuss several intractable and tractable cases.

1 Introduction

As large language models (LLMs) are increasingly deployed in high-stakes settings, safety failures can lead to significant social, economic, and security harms. These alarming failures have been reported in news articles, lawsuits, and academic works.

Multiple media reports and lawsuits allege that prolonged, emotionally intimate interactions with LLMs reinforced suicidal ideation (Duffy, 2025) and delusional thinking (Jargon and Schechner, 2025) rather than de-escalating crises, with claims that the system validated harmful thoughts (Brant, 2025), discouraged seeking human support, or failed to redirect users to appropriate crisis resources. These allegations include wrongful-death cases involving adolescents and young adults (O'Brien and Ortutay, 2025) and a separate case involving the amplification of conspiratorial beliefs linked to a murder-suicide (Lourosa-Ricardo, 2025). Thus, unsafe LLM behavior is not merely hypothetical but does arise in practice.

Academic and industrial research efforts have largely, if not entirely, focused on *heuristic* approaches on benchmarks and mitigation. There is no dearth of LLM safety benchmarks focusing on various safety aspects, such as truthfulness (Lin et al., 2022), toxicity (Hartvigsen et al., 2022), unethical scenarios (Shen et al., 2024), or generally adversarial dialogues (Ganguli et al., 2022). Another research direction focuses on mitigation strategies ranging from pre-training data filtering to unlearning of undesirable behaviors, and from direct prompting to post-training for safety.[1]

Beyond empirical research, however, what is desperately needed is research to understand the foundational computational task of LLM safety, particularly as LLMs gain greater agency and are applied across diverse languages and contexts.

Our Contributions. Our work offers the first foundational treatment of *safe* language generation, offering the following contributions:

1. We introduce a generic and versatile abstraction that captures the task of *safe language generation in the limit* (§4) under the learning paradigm of Gold (1967). In our new model, the learner sees examples from the desired language as well as examples from the harmful language, and the goal is to produce unseen examples from the (safe) set difference between the two languages.
2. As a first step, we introduce the task of *safe language identification in the limit*, where the goal of the learner is not to produce an unseen example but rather pinpoint the safe language among a collection of languages. Our analysis proves that safe language identification under our proposed model is impossible (§3).
3. We then prove, via a reduction, that safe language generation *is at least as hard* as traditional language identification in the limit (§5), and discuss several *intractable and tractable cases* of safe language generation (§6).

[1] We direct the interested reader to the plethora of relevant surveys that synthesize recent works on benchmarks and mitigation strategies (Liu et al., 2025; Li and Fung, 2025; Yong et al., 2025; Anthropic, 2025, *inter alia*).

The implications of our work are rather surprising. Seminal results from computational learning theory have shown that language generation in the limit is, in a formal sense, fundamentally easier than broader learning tasks such as language identification. In fact, (vanilla) language generation *is possible* even against an adversary that presents positive training examples in a worst-case manner. In contrast, our work shows for the first time that *safe* generation is a fundamentally harder task. We note that our work does not aim to invalidate empirical research on LLM safety; rather, it showcases the fundamental complexity of the task, which may inform practical decisions on safety trade-offs.

2 Related Work

Language *Identification* in the Limit. Gold's foundational paper (Gold, 1967) introduced language identification in the limit: there is a countable domain $\mathcal{X}$ and a countable collection of candidate languages $\mathcal{L} = \{L_1, L_2, \ldots\}$, one of which is the unknown target language K. From an enumeration of positive examples of the true language K, the learner A must output in every round an exact description of K (often an index in $\mathcal{L}$). Crucially, the learner does not get feedback on the correctness of their output. Gold showed that identification from positive data alone is impossible in general. Angluin (1980) gave a characterization of when identification from positive data *is* possible, via finite "telltale" sets that allow the learner to rule out competing hypotheses, and Angluin (1988) studied identification when examples are drawn stochastically. Together, these results highlight a fundamental obstacle: for any finite prefix of the enumeration, there may exist multiple distinct languages that remain consistent in the limit.

Language *Generation* in the Limit. Kleinberg and Mullainathan (2024) introduced language generation in the limit, replacing identification by a weaker goal: after some finite time, the learner must generate previously *unseen* elements of K. Their main (constructive) theorem shows that for every countable collection $\mathcal{L}$ over a countable domain, there exists an algorithm that generates in the limit from every $K \in \mathcal{L}$. In particular, this establishes a *separation* between identification and generation: generation can be universally achievable even when identification is not. Details of the KM algorithm are available in Appendix A.

On Refined Notions of Success. After establishing that unseen-example generation is tractable, the natural question becomes which additional requirements preserve this tractability and which do not. Raman et al. (2025) take a more learning theory-driven approach and provide a characterization of hypothesis classes that are uniformly and non-uniformly generatable. In parallel, the line of work by Kalavasis et al. (2025, 2024); Charikar and Pabbaraju (2025) characterizes the tension between consistency and expressiveness, which is typically seen in practice as the trade-off between hallucination and mode collapse. Kleinberg and Wei (2025a,b) propose density measures as a quantitative proxy for breadth and study how such guarantees behave under partial enumeration. Hanneke et al. (2025) showed that it is possible to have collections L_1 and L_2 that are generatable individually, while their union $L_1 \cup L_2$ is not. Arenas et al. (2025) emphasizes that despite theoretical guarantees that language generation is possible from positive examples alone, simple formal language classes may require samples beyond any computable bound to generate correctly. Another line of work keeps the basic generation goal but varies the *information model*. Raman and Raman (2025) study generation when the observed examples contain a bounded amount of adversarial corruption. Bai et al. (2026) unify noise, loss, and feedback within a single framework. Mehrotra et al. (2025) extends robustness to regimes of unbounded contamination.

Why Is This Work Different. Across these variants, there is still a single target language K, and an output is "wrong" only because it falls outside K. Our work introduces an additional constraint, motivated by safety concerns in LLM deployment: we model allowable outputs as the set difference $K \setminus H$, where H is a *harmful language* revealed over time along with the true language K. The key distinction is that the learner needs to generate while avoiding strings in H, which can overlap with K, i.e., a string can be consistent with the target language while still being disallowed.

3 Safe Language Identification is Impossible

Before considering safely *generating* in the limit, we start with a more fundamental question: *Can we* identify *the safe language (i.e., $K \setminus H$) in the limit?* Given the negative result from (Gold, 1967), which shows that language identification from positive

examples alone is impossible, the question above appears hopeless. On a second thought, one can see the strings from $\mathcal{X}$ that are labeled as members of H as *negative examples* with respect to the safe language $K \setminus H$. Specifically, strings that will only be labeled with 0 are strings that are in H but not in K (thus, they are not in $K \setminus H$ either), while strings that are labeled as both 0 and 1 are in $K \cap H$ (thus, they are not in $K \setminus H$).

It is plausible that the above new perspective on safe language identification may lead to tractability, as the inclusion of negative examples has been shown to transform problems that are intractable with only positive examples into ones that are solvable with both positive and negative examples (Gold, 1967). We note here that the sufficiency and impossibility theorems of Angluin (1980) on language identification in the limit *do not carry over verbatim* in our setting. This is because when a string from K is revealed by the adversary, it does not immediately serve as a positive example, since it may later be revealed to belong in H.

Definition 1 (**Safe Identification** $\mathcal{SI}$)**.** *Fix some K from the language collection $\mathcal{L}_K$ and some H from the language collection $\mathcal{L}_H$. Add language $K \setminus H$ to a random location of $\mathcal{L}_K$. In each step t, the algorithm A observes a labeled set S_t such that each string is labeled 1 if it is a member of K and 0 if it is a member of H. At each timestep, A outputs an index i of $\mathcal{L}_K$. We say that algorithm A* ***safely identifies from K given a harmful H in the limit*** *if there is some time $t' \in \mathbb{N}$ such that, for all $t > t'$ the algorithm's guess at step t is i for which $L_i = K \setminus H$ with respect to $\mathcal{L}_K$.*

Theorem 3.1. *For any safe language identification algorithm $A^{\mathcal{SI}}$, there exists a countable $\mathcal{X}$, a pair of language collections $\mathcal{L}_K$ and $\mathcal{L}_H$, and a choice of K and H, such that $A^{\mathcal{SI}}$ can not safely identify from K given a harmful H in the limit.*

Proof Sketch. For the full proof see Appendix B. In this proof, we use a standard diagonalization argument (Karbasi et al., 2025; Hanneke et al., 2025; Kleinberg and Mullainathan, 2024). We construct two carefully designed families of languages: one $\mathcal{L}_K$ containing the true language and the other $\mathcal{L}_H$ containing the harmful language. These two families are built so that no learning algorithm can reliably identify the safe language in the limit. The key idea is to define these languages using simple arithmetic patterns over the integers (e.g., odds, evens, and ranges of negative numbers), so that they overlap in a structured yet "deceptive" way. An adversary then reveals, via a careful enumeration, examples incrementally, always in a way that remains consistent with multiple possible choices of the true and harmful languages. Because the learner sees only finite data at any time, it can never rule out all but one hypothesis; any guess it makes is later contradicted by a chosen new example.

The core of the argument is an adaptive construction. Whenever the algorithm appears to have converged on a particular hypothesis for the identity of $K \setminus H$, the adversary reveals a new labeled example that invalidates it, while still remaining consistent with some alternative pair of languages from collections $\mathcal{L}_K$ and $\mathcal{L}_H$. This process can continue indefinitely. Either the algorithm never settles on the correct safe language, or it is forced to revise its hypothesis infinitely often. In both cases, identification in the limit fails, even though the safe language itself belongs to $\mathcal{L}_K$ and harmful examples are explicitly labeled. □

The main reason examples from H do not make safe language identification easier (despite being negative examples) is that the set of strings observed by $A^{\mathcal{SI}}$ does not necessarily include all possible negative examples. That is, it is possible to have strings that are neither in $K \setminus H$ nor in H. Overall, Gold (1967) showed that inference from positive data is strictly less powerful than inference from positive and negative data; the above theorem shows that with just a subset of negative examples (as opposed to exhaustive enumeration), the inferential power of the learner does not increase.

4 The Safe Generation Problem

Let $\mathcal{L}_K = \{L_1, L_2, L_3, \ldots\}$ be a countable collection of candidate *true* languages, each an infinite subset of a countable universe $\mathcal{X}$ of strings. Let $\mathcal{L}_H = \{L_1, L_2, L_3, \ldots\}$ be a countable collection of infinite candidate *harmful* languages, also subsets of $\mathcal{X}$. Both $\mathcal{L}_K$ and $\mathcal{L}_H$ are allowed repetitions, i.e., multisets. An adversary and an algorithm (or else the learner) A play the following game: The adversary secretly selects a true language $K \in \mathcal{L}_K$ and a harmful language $H \in \mathcal{L}_H$. At each timestep t, it presents a string from either K or H, i.e., a labeled string (w_t, y_t), where label is $y_t = 1$ if $w_t \in K$ and $y_t = 0$ if $w_t \in H$. In each round, the adversary reveals a string from either the true or the harmful language. We call $K \setminus H$ the *safe* language and $\perp$ the bottom string.

Additionally, through this generation, the adversary commits to an *enumeration* of K and an *enumeration* of H. The adversary can repeat strings in these enumerations, but for every string $w \in K$ and every string $w \in H$, there must be at least one time step t in which it appears. Let S_t denote the set of strings that the adversary has enumerated in steps 1 through t. At the end of each round, A guesses unseen words from the safe language but receives no feedback on whether the guesses are correct.

Definition 2 (Safe Generation $\mathcal{SG}$). *Fix some K from the language collection $\mathcal{L}_K$ and some H from the language collection $\mathcal{L}_H$. In each step $t = 1, 2, \ldots$ the algorithm A observes a labeled set S_t such that each string is labeled 1 if it is a member of K and 0 if it is a member of H. At each timestep, A outputs a string a_t (its guess for an unseen string in $K \setminus H$). We say that algorithm A **safely generates from K given a harmful H in the limit** if (i) in case $K \setminus H$ is countable, there is some time $t' \in \mathbb{N}$ such that for all steps $t > t'$ the algorithm's guess a_t belongs to $(K \setminus H) \setminus S_t$, and (ii) in case $K \setminus H$ is either finite or empty, there is some time $t'' \in \mathbb{N}$ for which for all steps $t > t''$ the algorithm's guess is $\perp$, where $\perp \notin \mathcal{X}$.*

On the Emptiness of Set Difference. In the above definition for safe generation, we expect two distinct behaviors of the algorithm depending on whether the set difference $K \setminus H$ is infinite or not. In case the set difference is infinite, the algorithm is expected to generate (in the limit) strings from the infinite set $K \setminus H$. In case the set difference is finite or empty, it is impossible[2] to generate from $K \setminus H$ in the limit, and the algorithm needs to identify this case by outputting the special symbol $\perp$. This asymmetry is intentional: since safety is a strict requirement in our setting, emitting unsafe strings when no safe output exists would be unacceptable. The symbol $\perp$ explicitly signals the absence of any safe generative behavior.

Why a Direct Application of KM Fails. A naive attempt to apply the KM algorithm for $\mathcal{LG}$ (see Appendix A) directly to $\mathcal{SG}$ fails. To see why, consider substituting generation from the difference with generation solely from $\mathcal{L}_K$, while discarding candidate languages whenever they become inconsistent due to H's examples. That is, whenever a previously shown word $w \in K$ is also enumerated in H, a modified KM algorithm could recompute the set of consistent languages and continue as usual. However, this naive approach breaks a crucial core invariant underlying the correctness arguments of KM. In the KM setting for $\mathcal{LG}$, when the correct language (K) is first considered, it will *remain consistent* for all timesteps thereafter. However, under the $\mathcal{SG}$ setting, any word $w \in K \cap H$ that has only been revealed (so far) as a member of K and not H, renders the correct language ($K \setminus H$) inconsistent with S_t, because $\forall w \in K \cap H : w \notin K \setminus H$. Thus, the correct language $K \setminus H$ is not guaranteed to remain consistent across all possible enumerations.

[2] In this case, the impossibility is due to a lack of enough safe strings to generate in the limit.

Computational Problems	
$\mathcal{LI}$	Language Identification
$\mathcal{LG}$	Language Generation
$\mathcal{SI}$	Safe Language Identification
$\mathcal{SG}$	Safe Language Generation
Setup	
$\mathcal{X}$	Countable universe of strings
$\mathcal{L}, \mathcal{L}_K, \mathcal{L}_H$	Collections of languages
K	True language of $\mathcal{LI}$ or $\mathcal{LG}$
K, H	True, harmful languages for $\mathcal{SI}$ or $\mathcal{SG}$
$K \setminus H$	Safe language of $\mathcal{SI}$ or $\mathcal{SG}$
Algorithm Related	
$A^{\mathcal{P}}$	Algorithm solving problem $\mathcal{P}$
t	timestep t
w_t	word revealed at timestep t
S_t	set of words revealed up to t
$\mathcal{C}_t$	languages consistent with S_t
$\perp$	bottom string

Table 1: Notation summary.

5 Safe Language Generation is *at Least* as Hard as Language Identification

In this section, we will show that safe generation in the limit from a language K given a harmful H is as hard as language identification in the limit. Our reduction assumes an algorithm $A^{\mathcal{SG}}$ that solves the safe generation and uses it to construct an algorithm $A^{\mathcal{SI}}$ that solves any instance of the language identification problem. Thus, if there were an easy way to solve $\mathcal{SG}$, we would have an easy way to solve $\mathcal{LI}$, which is impossible. The complete proof details are in Appendix C, and below we provide a brief sketch with the important details.

Theorem 5.1. *Let $A^{\mathcal{SG}}$ be an algorithm that safely generates from a language K given a harmful language H in the limit for any enumeration of the two languages from collections $\mathcal{L}_K$ and $\mathcal{L}_H$ from domain $\mathcal{X}$. Then, for any collection $\mathcal{L}$ from the domain $\mathcal{X}$, there exists an algorithm $A^{\mathcal{ID}}$ that uses $A^{\mathcal{SG}}$ as a subroutine to identify K in the limit given any enumeration of any $K \in \mathcal{L}$.*

Proof Sketch. At a high level, Algorithm $A^{\mathcal{ID}}$ at time-step t will consider the first t languages from $\mathcal{L}$, and will maintain a list of the languages that are consistent with the set S_t of revealed strings. After some timestep, the true K will be inserted into the set of consistent languages $A^{\mathcal{ID}}$ under consideration (and never removed from it). In the limit, only the correct language K and any superset languages $L_i : K \subseteq L_i$ will be consistent with S_t.

Given such a list of consistent languages,we will take advantage of the asymmetry in the outputs of the safe generation algorithm $A^{\mathcal{SG}}$. In particular, for a pair of languages L_i, L_j consistent with S_t, $A^{\mathcal{ID}}$ will construct an instance of a safe generation problem with effectively one language as the K (with $\mathcal{L}_K = \{L_i, L_i, \ldots\}$) and the other as the harmful H one ($\mathcal{L}_H = \{L_j, L_j, \ldots\}$), with an enumeration of up to t elements; running $A^{\mathcal{SG}}$ on this instance and observing its output allows ordering L_i and L_j when one is a subset of the other.

The guarantee of hypothesized $A^{\mathcal{SG}}$ for safe generation is that there exists some $t \in \mathbb{N}$ after which the returned word $w_t \in K \setminus H$, *if possible*. In cases where safe generation is impossible by construction, the algorithm must generate $\perp$.[3] The key observation is that $\perp$ is returned when $K \subseteq H$, which $A^{\mathcal{ID}}$ uses to identify such cases.[4] Through this mechanism, $A^{\mathcal{ID}}$ can identify any pair of consistent languages that share a strict subset relationship: if $A^{\mathcal{SG}}$ generates $\perp$ on inputs from two languages L_i and (harmful) L_j, then one can assume that $L_i \subseteq L_j$.[5] This will effectively allow $A^{\mathcal{ID}}$ to partially order (with respect to inclusion) the list of consistent languages, and to do so in finite time.

Recall that the issue with original language identification in Gold (1967) was that the algorithm had no way of distinguishing between the correct language K and any language that is its superset. As the enumeration of the target language unfolds, $A^{\mathcal{ID}}$ will not only maintain a list of consistent languages (which will eventually include K) but also partially *order* the list by calling $A^{\mathcal{SG}}$ and observing its outputs. The partial ordering will ensure that if any two consistent languages share a strict subset relationship, then the "smaller" one will definitely appear before the bigger one in the ordered list.

[3]The proof in Appendix C shows that this holds even with relaxed safety requirements, allowing the algorithm to output a random word from $\mathcal{X}$ when safe generation is impossible.

[4]Safe generation is also impossible in the limit in the case of a finite set difference $K \setminus H$, but it falls outside our setup.

[5]Similarly, one can enumerate L_j as the target and L_i as the harmful language to decide if $L_j \subseteq L_i$.

In the limit, any languages that are strict subsets of the true language will be rendered inconsistent by the enumeration of K. Thus, an algorithm $A^{\mathcal{ID}}$ that returns the index of the first language in an *ordered* list constructed as above, will in the limit correctly identify the target language K. □

The complete proof, in Appendix C, handles technical aspects omitted in this proof sketch, such as returning the index of the *first* occurrence of K, or maintaining the ordered list S_t of consistent languages in time linear to the number of examples.

6 Tractable and Intractable Cases for Safe Language Generation

In this section, we analyze three variants of our core safe generation problem and determine whether any of them render safe generation tractable. First, in Section 6.1, we show that even if both languages K and H are identifiable (with respect to Angluin (1980)), the safe generation problem $\mathcal{SG}$ remains impossible in the general case. This impossibility follows from the fact that deciding whether the set difference of two countable sets is empty is undecidable. Next, in Section 6.2, we drop the identifiability assumption on K and H, but instead assume A has access to an oracle that can answer whether the set difference of any two sets is empty or not. We show that access to this oracle alone is still insufficient to make $\mathcal{SG}$ tractable. Trivially, if A has access to such an oracle *and* both languages are identifiable, then the safe language generation problem is tractable. Interestingly, the above two conditions are sufficient but not necessary. In the final variant from Section 6.3, we describe a tractable setting: namely, when the set difference between every pair of languages in $\mathcal{L}_K$ and $\mathcal{L}_H$ is guaranteed to be infinite by construction, while the chosen languages are not necessarily identifiable.

6.1 Identifiability Alone is Not Enough for Safe Generation in the Limit

Condition. (Angluin, 1980) Fix a language collection $\mathcal{L}$. The collection $\mathcal{L}$ is said to satisfy Angluin's condition if for any index i, there is a finite set of strings T_i such that T_i is a subset of $L_i \in \mathcal{L}$, i.e., $T_i \subseteq L_i$, and the following holds:

For all $j \geq 1$, if $T_i \subseteq L_j$, then L_j is not a proper subset of L_i.

A telltale oracle is a primitive that, given index i, outputs an enumeration of the set T_i.

Angluin's Condition for Identifiability. A natural first direction to explore is what happens when

$\mathcal{L}_K$ and $\mathcal{L}_H$ are designed in such a way that *any choice* of K and H is identifiable in the limit. The work of Angluin (1980) established a necessary and sufficient condition for identifiability.

We begin with a decision problem that will anchor the subsequent analysis and show that it is undecidable to safely generate even if A deals with identifiable K and H. Language (in the computability sense) $L_{\texttt{DIFF-EMPTY}}$ contains all the pairs of infinite sets such that their difference is empty.

$$L_{\texttt{DIFF-EMPTY}} = \{(X_1, X_2) \mid X_1, X_2 \subseteq \mathcal{X}, \\ |X_1| = |X_2| = \infty,\ X_1 \setminus X_2 = \emptyset\}.$$

The undecidability of the above language follows from a standard argument; see the Appendix.

Theorem 6.1. *$L_{\texttt{DIFF-EMPTY}}$ is undecidable.*

Suppose, for simplicity, that both K and H are given explicitly to an algorithm A tasked with safe generation. Even in this setting, *A cannot guarantee that it will output, within a finite number of steps, a string w* such that $w \in K$ and $w \notin H$. The reason is that A may be dealing with an instance $(K, H) \in L_{\texttt{DIFF-EMPTY}}$, in which case no string satisfying these properties exists. Because it is undecidable to determine whether a given instance has an empty difference, A cannot determine when to stop seeking a safe generation because none exists; even when explicit descriptions of K and H are provided rather than learned.

Corollary 6.1. *There are identifiable collections $\mathcal{L}_K$ and $\mathcal{L}_H$ such that no algorithm can safely generate in the limit unless* DIFF-EMPTY *is decidable.*

6.2 Access to a Set-Difference Oracle Alone is Not Enough for Safe Generation

In what follows, we assume that the algorithm $A^{\mathcal{SG}}$ is equipped with a set-difference oracle $\mathcal{O}^{SD}(\cdot, \cdot)$, which returns 1 if the difference between two input infinite collections is empty and 0 otherwise. We show that even with access to this oracle, safe generation in the limit is still impossible. Our argument constructs two collections of languages and an adversarial enumeration that diagonalizes against any fixed algorithm. Although the algorithm may appear to generate safely at each finite stage, the adversary repeatedly forces hypothesis revisions. In the limit, the revealed enumerations correspond to a pair of languages with empty difference, for which the only safe output is $\perp$, yet all algorithms can be misled and fail to recognize this setting. Figure 1 illustrates this behavior, where

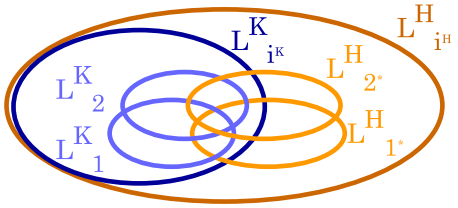

Figure 1: An illustration of collections in which each phase seemingly has infinite safe strings, but the enumerated languages $(L^K_{i^K}, L^H_{i^H})$ have no safe strings.

the adversarial enumeration switches between pairs $(L^K_1, L^H_{1^*}), (L^K_2, L^H_{2^*}), \ldots$, and in case A pivots its hypothesis, the enumeration can be made from $(L^K_{i^K}, L^H_{i^H})$, which has no safe outputs.

Theorem 6.2. *There are collections $\mathcal{L}_K$ and $\mathcal{L}_H$ such that no algorithm can safely generate in the limit even with access to set difference oracle $\mathcal{O}^{SD}$, i.e.,* DIFF-EMPTY*$(\cdot)$ being decidable.*

Proof Sketch. In this proof, we make an adversarial diagonalization argument for two language collections that we construct. Suppose that the following property holds for $\mathcal{L}_K$ and $\mathcal{L}_H$:

> (**Property** P^*) There exists an index i^K and i^H for which $L^K_{i^K} \subseteq L^H_{i^H}$, such that for every pair of finite sets of strings (T^K, T^H) such that $T^K \subset L^K_{i^K}$ and $T^H \subset L^H_{i^H}$ the following holds:
>
> There is a $j \geq 1$ and an associated $j^* \geq 1$, such that **(1)** if $T^K \subseteq L^K_j$, then L^K_j is a proper subset of $L^K_{i^K}$, **(2)** if $T^H \subseteq L^H_{j^*}$, then $L^H_{j^*}$ is a proper subset of $L^H_{i^H}$, and **(3)** $L^K_j \setminus L^H_{j^*}$ has infinite cardinality.

Intuitively, property P^* ensures that although the final true and harmful languages $(L^K_{i^K}, L^H_{i^H})$ have an empty difference, every finite amount of evidence is compatible with some other pair of languages $(L^K_j, L^H_{j^*})$ under which infinitely many safe strings exist– condition (3) of P^*. The algorithm cannot determine, from finite data, whether it is in the "final" scenario in which no safe string exists or in a scenario in which safe generation is possible. Each time the algorithm produces a seemingly safe output, the adversary can extend the enumeration to invalidate the current hypothesis while remaining consistent with all prior observations.

On a more technical level, fix indices i^K and i^H with respect to Property P^*, and let $E^K_{i^K}$ and $E^H_{i^H}$ be any fixed enumerations of $L^K_{i^K}$ and $L^H_{i^H}$, respectively. We describe an adversary that presents prefixes of these enumerations to any arbitrary algorithm $A^{\mathcal{SG}}$. The construction proceeds in phases. At the beginning of phase l, the adversary has revealed finite prefixes of $E^K_{i^K}$ and $E^H_{i^H}$, inducing finite sets T^K and T^H of observed strings. By con-

ditions (1) and (2) from Property P^*, there exist indices i_l and i_l^* such that

$$T^K \subseteq L_{i_l}^K \subsetneq L_{i^K}^K, \qquad T^H \subseteq L_{i_l^*}^H \subsetneq L_{i^H}^H,$$

and such that the difference $L_{i_l}^K \setminus L_{i_l^*}^H$ is infinite (condition (3)). The adversary then continues the enumeration as if the true and harmful languages were $L_{i_l}^K$ and $L_{i_l^*}^H$. At this point, exactly one of the following two cases must occur.

Case 1. The algorithm $A^{\mathcal{SG}}$ never outputs a string in $L_{i_l}^K \setminus L_{i_l^*}^H$. Since this set is infinite, failure of $A^{\mathcal{SG}}$ to produce such a string implies that $A^{\mathcal{SG}}$ cannot safely generate for the pair $(L_{i_l}^K, L_{i_l^*}^H)$, in which case the adversary adapts and sticks to this enumeration, and the impossibility follows.

Case 2. The algorithm $A^{\mathcal{SG}}$ does output at least one string in $L_{i_l}^K \setminus L_{i_l^*}^H$. In this case, the adversary changes its enumeration strategy by exploiting the fact that $L_{i_l}^K$ is a proper subset of $L_{i^K}^K$. Specifically, the adversary extends the enumeration by revealing a string in $L_{i^K}^K \setminus L_{i_l}^K$, which is guaranteed to exist[6] by Property P^*. This forces the algorithm to revise its hypothesis in order to be consistent with the observations and initiates the next phase of the adversarial enumeration.

If Case 1 ever occurs at some finite phase, the algorithm fails outright. Otherwise, Case 2 occurs at every phase. Since $L_{i^K}^K$ is infinite, the adversary can carry out infinitely many phases, ultimately revealing complete enumerations of $L_{i^K}^K$ and $L_{i^H}^H$. However, in each phase transition, the algorithm produces an output that is *not safe* with respect to the final target pair $(L_{i^K}^K, L_{i^H}^H)$, whose difference is empty. Thus, $A^{\mathcal{SG}}$ produces infinitely many unsafe outputs and fails to safely generate in the limit.

Since the argument applies to any arbitrary algorithm $A^{\mathcal{SG}}$, no algorithm can safely generate in the limit, even when $\texttt{DIFF-EMPTY}(\cdot)$ is decidable. □

Remark 1. If (i) collections $\mathcal{L}_K$ and $\mathcal{L}_H$ are identifiable, and (ii) A has access to a set difference oracle, then A can safely generate in the limit.

6.3 A Tractable Case: Language Collections with Guaranteed Infinite Set Differences

Before going into the details of the proofs below, we establish a few general principles for an algorithm that attempts to safely generate.

First, we mention that, given an enumeration (E_K, E_H) of strings from K and H presented to A by the adversary, the algorithm A can efficiently maintain two lists of hypotheses K and H that are *consistent* with the enumeration. This maintenance proceeds in a manner analogous to the approach of Kleinberg and Mullainathan (2024)– which will be referred to as KM in the following.

Second, recall that, *in the limit*, the only "interesting" languages from $\mathcal{L}_K$ and $\mathcal{L}_H$ are those that will remain consistent with respect to the enumeration of the chosen K, H. Thus, the only languages that A needs to track are those that form a hierarchy under set inclusion[7] with respect to K and H.

Third, we establish that, in order to safely generate from $K \setminus H$, we need to at least be able to generate from K alone. As KM showed, this is possible. Given a list of consistent languages K, in order to generate in the limit from the correct K, a conservative but effective approach is to choose the *smallest* consistent K. In fact, the KM algorithm guarantees exactly that: it maintains a list of all consistent languages $L_i \subseteq \ldots \subseteq L_z \subseteq \ldots L_j$, and chooses the "smallest safe" language (KM calls it (t, m)-critical language), proving that in the limit their algorithm will always generate from either the correct K or from one of its subset languages.

Given that KM establishes that a viable but conservative strategy for generation from K corresponds to selecting the smallest consistent candidate, we pose the following question: What is a conservative choice of candidates when the goal is to generate from $K \setminus H$ instead? We show that much like KM, for generating from K the conservative choice is to go with the "smallest" candidate, but to avoid all consistent harmful candidates, the conservative choice is to go with the *largest* consistent H.

New Setting. First, consider a slightly different setting, where we provide the algorithm with collections $\mathcal{L}_K, \mathcal{L}_H$ between which all pairs of possible $K \in \mathcal{L}_K, H \in \mathcal{L}_H$ have an infinite difference: i.e., $K_i \setminus H_j$ has infinite cardinality $\forall i, j$ with $K_i \in \mathcal{L}_K$ and $H_j \in \mathcal{L}_H$. This additional condition circumvents the decidability issue in the set difference problem. In the following, we present a variation of the definition of safe generation $\mathcal{SG}^\infty$ in which every pair of a true language from $\mathcal{L}_K$ with a harmful language from $\mathcal{L}_H$ has *by construction* an infinite set difference.

Definition 3 (Safe Generation - Infinite Differences $\mathcal{SG}^\infty$). *Fix some K from the language collection $\mathcal{L}_K$ and some H from the language collection*

[6] Recall that L_j^K is a proper subset of $L_{i^K}^K$, thus, there is always a string that is in $L_{i^K}^K$ and not in L_j^K.

[7] Recall that all languages in our collection are infinite.

$\mathcal{L}_H$. *All possible pairs of languages from* $\mathcal{L}_K$ *and* $\mathcal{L}_H$ *are guaranteed to have an enumerable (infinite) difference, such that* $\forall K \in \mathcal{L}_K, \forall H \in \mathcal{L}_H : |K \setminus H| = \infty$. *In each step* $t = 1, 2, 3, \ldots$, *the algorithm* A *observes a labeled set* S_t *such that each string is labeled* 1 *if it is a member of* K *and* 0 *if it is a member of* H. *At each timestep,* A *outputs a string* a_t. *We say that algorithm* A ***safely generates from*** K ***given a harmful*** H ***with infinite set difference in the limit*** *if there is some time* $t' \in \mathbb{N}$ *such that for all steps* $t > t'$ *the algorithm's guess* a_t *belongs to* $(K \setminus H) \setminus S_t$.

Theorem 6.3. *Let* $\mathcal{L}_K, \mathcal{L}_H$ *be language collections, such that* $\forall K \in \mathcal{L}_K, \forall H \in \mathcal{L}_H : |K \setminus H| = \infty$. *There is an algorithm* A *that, for any labeled enumeration of any language* $K \in \mathcal{L}_K$ *and* $H \in \mathcal{L}_H$, A *safely generates from* K *given a harmful* H *with infinite set difference in the limit.*

Proof. The proposed A essentially runs two instances of the KM generation, one for "choosing" a good candidate for K and one for H. As the arbitrarily chosen enumerations proceed (through an approach similar to KM), the constructed algorithm A maintains an ordered list of languages from $\mathcal{L}_K$ consistent with the enumeration E_K, and always chooses the "smallest" as its guess for the correct K (denote its guess by K_c). In a similar manner, A maintains an ordered list of languages from $\mathcal{L}_H$ consistent with the enumeration E_H, but now always choose the "largest" as its guess for the correct H (denote its guess by H_c).[8] We know that after some t, through the KM algorithm, the selected language K_c will either be a subset of or the correct K. We also know that after some t', A will consider the correct language H, and from this point on, it will remain within our set of consistent Hs. Now, we argue that for any $t^* > \max(t, t')$, the algorithm will safely generate. After t^*, we know that (a) algorithm A will always choose some $K_c \subseteq K$, and (b) we always choose some $H_c \supseteq H$. Importantly, from the setting, we also know that their difference $K_c \setminus H_c$ will be infinite, and thus A can, in finite time, find a string to return from it, which will also belong in $K \setminus H$.[9] Thus, the proposed algorithm A safely generates in the limit. □

[8] Subscript c stands for "chosen". K_c and H_c need not have the same index in their respective collections.

[9] The difference needs to be infinite, not simply non-empty. This guarantees that A can find a string in $(K_c \setminus H_c) \setminus S_t$. If $(K_c \setminus H_c)$ is non-empty but finite, the adversary could first enumerate it, thereby leaving A with no strings to generate.

6.4 Conservative Generation Fails in $\mathcal{SG}$

Our findings show that the two collections *do not have to be identifiable* if one guarantees that all set differences have infinite cardinality. The tractability proof of $\mathcal{SG}^\infty$ shows that the conservative generation strategy of selecting the smallest candidate K and the largest candidate H is sound.

In the general $\mathcal{SG}$ case, though, this conservative generation strategy can not be used. Consider again an algorithm that maintains two lists of languages $K_1 \subseteq \ldots \subseteq K_i \subseteq \ldots \subseteq K_n$ and $H_1 \subseteq \ldots \subseteq H_j \subseteq \ldots \subseteq H_m$ which are both consistent with the respective enumerations E_K and E_H. For simplicity, assume that A is at a timestep where both the correct K_i and H_j are under consideration and are members of the two lists. The conservative choice for the algorithm is still to select $K_c \leftarrow K_1$ and $H_c \leftarrow H_m$ (i.e., the smallest K and the largest H). Equivalently, this strategy maximizes the portion removed from true K: it selects the smallest subset of K and, in addition, removes the largest portion through (a potentially superset of) H. This gives the smallest remaining difference $K \setminus H$. Without Theorem's 6.3 assumptions about the cardinality of set difference, though, the difference $K_c \setminus H_c$ could be empty (in which case A is tempted to output $\perp$), even if the difference $K \setminus H$ is not empty, i.e., safe generation is possible.

7 Conclusion

Here, we discuss informal practical takeaways from our results. If the class of harmful languages $\mathcal{L}_H$ is overly expansive, the resulting intersection with K may become large (even encompassing the entire K), which is the primary source of intractability. In this sense, an overly generous definition of harmfulness can itself render the task intractable.

The abstraction in this work is deliberately broad: it makes no assumptions about the internal learning strategy of A or the structure of harmful languages in $\mathcal{L}_H$. For example, H could be as simple as *"all strings containing the word* `bomb`*"*, which should be easy to avoid in generation. The generality of our abstraction comes at a cost: it may shift attention away from more constrained settings in which safe generation is tractable.

On the positive side, safe generation is tractable if the harmful part of K (i.e., $K \cap H$) is finite, even if H itself is infinite. One interpretation of our results is that *narrow* definitions of safety are provably achievable in practice.

Limitations

Due to the length requirements, we only include proof sketches in the main paper. The complete proofs are available in the respective appendices.

References

Dana Angluin. 1980. Inductive Inference of Formal Languages from Positive Data. *Information and Control*, 45(2):117–135.

Dana Angluin. 1988. *Identifying Languages from Stochastic Examples*. Yale University. Department of Computer Science.

Anthropic. 2025. Agentic Misalignment: How LLMs Could Be Insider Threats. https://www.anthropic.com/research/agentic-misalignment.

Marcelo Arenas, Pablo Barceló, Luis Cofré, and Alexander Kozachinskiy. 2025. Language Generation: Complexity Barriers and Implications for Learning. *Preprint*, arXiv:2511.05759.

Yannan Bai, Debmalya Panigrahi, and Ian Zhang. 2026. Language Generation in the Limit: Noise, Loss, and Feedback. In *Proceedings of the 2026 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. Conference version (SIAM).

Brian Brant. 2025. Suicidal College Grad's Parents Claim ChatGPT Fed His Despair and They Didn't Realize Until Far Too Late. People Magazine. *Accessed: 01-03-26*.

Moses Charikar and Chirag Pabbaraju. 2025. Exploring Facets of Language Generation in the Limit. In *Proceedings of Thirty Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pages 854–887. PMLR.

Clare Duffy. 2025. Parents of 16-year-old Sue OpenAI, Claiming ChatGPT Advised Teen's Suicide. 6ABC Action News. *Accessed: 01-03-26*.

Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, Andy Jones, Sam Bowman, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Nelson Elhage, Sheer El-Showk, Stanislav Fort, and 17 others. 2022. Red Teaming Language Models to Reduce Harms: Methods, Scaling Behaviors, and Lessons Learned. *Preprint*, arXiv:2209.07858.

E Mark Gold. 1967. Language Identification in the Limit. *Information and Control*, 10(5):447–474.

Steve Hanneke, Amin Karbasi, Anay Mehrotra, and Grigoris Velegkas. 2025. On Union-Closedness of Language Generation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Thomas Hartvigsen, Saadia Gabriel, Hamid Palangi, Maarten Sap, Dipankar Ray, and Ece Kamar. 2022. ToxiGen: A Large-Scale Machine-Generated Dataset for Adversarial and Implicit Hate Speech Detection. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3309–3326, Dublin, Ireland. Association for Computational Linguistics.

Julie Jargon and Sam Schechner. 2025. Seven Lawsuits Allege OpenAI Encouraged Suicide and Harmful Delusions. The Wall Street Journal. *Accessed: 01-03-26*.

Alkis Kalavasis, Anay Mehrotra, and Grigoris Velegkas. 2024. Characterizations of Language Generation With Breadth. *Preprint*, arXiv:2412.18530.

Alkis Kalavasis, Anay Mehrotra, and Grigoris Velegkas. 2025. On the Limits of Language Generation: Trade-Offs between Hallucination and Mode-Collapse. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 1732–1743. ACM.

Amin Karbasi, Omar Montasser, John Sous, and Grigoris Velegkas. 2025. (Im)possibility of Automated Hallucination Detection in Large Language Models. In *ICML 2025 Workshop on Reliable and Responsible Foundation Models*.

Jon Kleinberg and Fan Wei. 2025a. Density Measures for Language Generation. In *FOCS 2025*. FOCS 2025. Full version: arXiv:2504.14370.

Jon Kleinberg and Fan Wei. 2025b. Language Generation and Identification From Partial Enumeration: Tight Density Bounds and Topological Characterizations. *Preprint*, arXiv:2511.05295.

Jon M. Kleinberg and Sendhil Mullainathan. 2024. Language Generation in the Limit. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Miles Q. Li and Benjamin C. M. Fung. 2025. Security Concerns for Large Language Models: A Survey. *Preprint*, arXiv:2505.18889.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring How Models Mimic Human Falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland. Association for Computational Linguistics.

Songyang Liu, Chaozhuo Li, Jiameng Qiu, Xi Zhang, Feiran Huang, Litian Zhang, Yiming Hei, and Philip S Yu. 2025. The Scales of Justitia: A Comprehensive Survey on Safety Evaluation of LLMs. arXiv:2506.11094.

Cristina Lourosa-Ricardo. 2025. A Son Blames ChatGPT for a Murder-Suicide. The Wall Street Journal. *Accessed: 01-03-26*.

Anay Mehrotra, Grigoris Velegkas, Xifan Yu, and Felix Zhou. 2025. Language Generation with Infinite Contamination. *Preprint*, arXiv:2511.07417.

Matt O'Brien and Barbara Ortutay. 2025. Study Says ChatGPT Giving Teens Dangerous Advice on Drugs, Alcohol, and Suicide. PBS News. *Accessed: 01-03-26*.

Ananth Raman and Vinod Raman. 2025. Generation from Noisy Examples. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 51079–51093. PMLR.

Vinod Raman, Jiaxun Li, and Ambuj Tewari. 2025. Generation Through the Lens of Learning Theory. In *Proceedings of Thirty Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pages 4740–4776. PMLR.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2024. "Do Anything Now": Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 1671–1685. ACM.

Zheng Xin Yong, Beyza Ermis, Marzieh Fadaee, Stephen Bach, and Julia Kreutzer. 2025. The State of Multilingual LLM Safety Research: From Measuring The Language Gap To Mitigating It. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 15856–15871, Suzhou, China. Association for Computational Linguistics.

A Description of the KM Algorithm

Intuition and High-level Goal. The purpose of the algorithm of Kleinberg and Mullainathan (2024) in Section 5.1 is to achieve *generation in the limit* without ever identifying the true language. The algorithm observes only positive examples

$$S_t = \{w_1, \ldots, w_t\}$$

from an unknown target language $K = L_z$, drawn from a countable list of candidate languages

$$\mathcal{C} = \{L_1, L_2, L_3, \ldots\}.$$

The algorithm has access to a membership oracle to query whether a string belongs to a candidate language L_i, but it cannot test membership in the unknown target language K directly. Since language identification is provably impossible in this setting, the algorithm instead adopts a *conservative* strategy: it never commits to a single hypothesis for K, but instead generates strings that are guaranteed to lie in *all* plausible hypotheses consistent with the data seen so far.

The Role of (t, m)-critical Languages. Exact set containment between languages cannot be tested in finite time due to the fact that the languages are infinite. To overcome the testing membership problem for an infinite number of strings, KM tests containment on a *restricted part of the vocabulary*; this approach can be viewed as an approximation of the (infinite) containment computation. On a more technical note, the universe of strings is enumerated as

$$U = \{u_1, u_2, u_3, \ldots\},$$

and for (1) each language L_i and (2) cutoff m, the algorithm considers the finite prefix (i.e., the restricted vocabulary)

$$L_i[m] = L_i \cap \{u_1, \ldots, u_m\}.$$

At step t, the algorithm restricts attention to the first t candidate languages $\mathcal{C}_t = \{L_1, \ldots, L_t\}$ and defines a language $L_n \in \mathcal{C}_t$ to be (t, m)-*critical* if it is consistent with the observed sample S_t, and if its finite prefix $L_n[m]$ is contained in the prefix of every consistent language that appears before L_n in $\mathcal{C}_t$. Intuitively, a (t, m)-critical language is the "smallest" consistent language given the current finite evidence on this bounded vocabulary (controlled by m) and bounded subcollection of languages (controlled by t).

Generation Procedure and Convergence. In each step t, the algorithm begins with a cutoff m large enough to include all strings observed so far, and then incrementally increases m. For each value of m, it identifies the highest-indexed (t, m)-critical language $L_{n_t(m)}$ (here, the $n_t(m)$ variable denotes the highest-indexed language among those that are (t, m)-critical.). The algorithm then searches for the first string $u_i \leq m$ such that

$$u_i \in L_{n_t(m)} \setminus S_t.$$

If such a string exists, it is output and the generation of this timestep concludes; otherwise, m is increased and the process repeats. The new m can lead to a different language becoming (t, m)-critical. This search is guaranteed to terminate, i.e., the identity of the highest-indexed (t, m)-critical language can change only finitely many times as m grows, and once it stabilizes, the "infiniteness" of the language's cardinality ensures the existence of a new string. Moreover, after some finite time t^*, the true language L_z becomes (t, m)-critical for all m, implying that every subsequently selected language is either L_z or a subset of L_z. From that point onward, every generated string lies in $K \setminus S_t$. The algorithm thus succeeds by progressively shrinking the space of safe outputs until it is guaranteed to lie entirely within the true language, without ever identifying it.

B Extended Proof for Theorem 3.1

Proof. In this proof, we construct the collections of languages $\mathcal{L}_K$ and $\mathcal{L}_H$ such that no matter how A attempts to identify the safe language, there is always a choice of K and H that they can not identify in the limit.

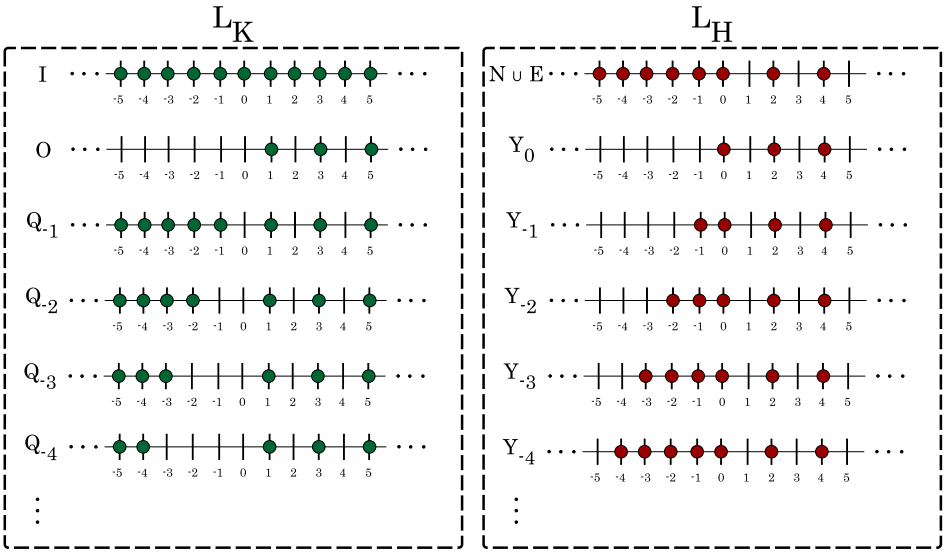

Figure 2: An illustration of the collection of $\mathcal{L}_K$ and $\mathcal{L}_H$ for the constructed instantiation for which there is no timestep in which an algorithm identifies the correct language $K \setminus H$ consistently. Let I denote the set of integers, E the set of even integers, and O the set of odd integers.

Our construction operates over the universe $\mathcal{X} = \mathbb{Z}$ of integers, and the languages will be defined as arithmetic progressions. One can "relabel" each integer with a unique string; thus, the impossibility result is not tied only to languages of integers.

The Languages. To set up the language collections $\mathcal{L}_K$ and $\mathcal{L}_H$, we define several languages that we will need in the proof. Let O be the language of odd positive integers and E be the language of even positive integers:

$$O = \{2i - 1 : i = 1, 2, \ldots\},$$
$$E = \{2i : i = 0, 1, 2, \ldots\}.$$

Next, we define two more languages that are parametrized so as to generate a countable collection of languages. The first language is Y_{-a}, which, on a high level, contains the negative integers from $-a$ all the way to 0 (i.e., $-a \to 0$) as well as all the even positive integers. Formally, it is defined as

$$Y_{-a} = \{-a + i : i = 0, \ldots, a\} \cup E, \text{ for } a \geq 0$$

The second language is Q_{-b} that contains negative integers from $-b$ to $-\infty$ (i.e., $-\infty \to -b$) as well as all the positive odd integers. Formally:

$$Q_{-b} = \{-b - i : i = 0, 1, 2, \ldots\} \cup O, \text{ for } b \geq 1.$$

Finally, we define all integers as $I = \{i : i = \ldots, -2, -1, 0, 1, 2, \ldots\}$ and the negative integers as $N = \{-i : i = 1, 2, \ldots\}$.

The Language Collections. We can now define the language collections for the safe identification instance. An illustration of these collections is provided in Figure 2.

- $\mathcal{L}_H$: The collection $\mathcal{L}_H$ contains language $N \cup E$ as well as all parameterizations of languages Y_{-a} for $a \geq 0$. Thus, it is a countable collection.
- $\mathcal{L}_K$: The collection $\mathcal{L}_K$ is the union of language I, language O, and all the parameterizations of languages Q_{-b} for $b \geq 1$. Thus, it is a countable collection.

Now that we have established the language collections, we will show that for any identification algorithm A there exists (i) a choice of $K \in \mathcal{L}_K$, (ii) a choice of $H \in \mathcal{L}_H$, and (iii) a labeled enumeration E of K and H, such that if $K \setminus H$ is the safe language and the adversary provides the enumeration E to algorithm A, then the algorithm A can not safely identify in the limit.

We will construct the enumeration in phases and adaptively select K and H based on the outputs of A. The proposed construction of E proceeds in multiple and possibly infinite phases.

We define the following two labeled sequences that we will use throughout the proof: $T_1 = \big((0,1), (1,1), (-1,1), (2,1), (-2,1), (3,1), (-3,1), \dots\big)$, where the second element of each pair is the label 1 that signifies that the number is in K; and $T_2 = \big((0,0), (2,0), (4,0), \dots\big)$, where the second element of the pair is the label 0 that signifies that the number is in H. Notice that the elements in T_2 belong to *all* potential harmful languages in $\mathcal{L}_H$ and there are infinite such elements. Finally, we define a sequence T that interleaves the labeled examples from T_1 and T_2, i.e., $T = (T_1(1), T_2(1), T_1(2), T_2(2), T_1(3), T_2(3), \dots)$. Looking ahead, the adversary will use the sequence T to reveal elements to the algorithm A, observe its identification guess, and adapt the final enumeration E accordingly.

Phase 1 of Construction of Enumeration E. In the first phase, we construct the enumeration E by revealing elements as if $K = I$ and $H = Y_0$. Specifically, at any round t, the adversary presents the labeled element $T(t)$ to the algorithm A. The behavior of algorithm A is unknown; thus, it either correctly identifies $K \setminus H$ as Q_{-1} or not.

Next, we consider the following case analysis about A's behavior: (i) either there is some finite step $t_1 \in \mathbb{N}$ such that the guess of the algorithm A is Q_{-1}, or (ii) there is no $t_1 \in \mathbb{N}$ such that A guesses Q_{-1}. In the latter case, the adversary commits to the choices $K = I$ and $H = Y_0$ and enumeration $E = T$ and is guaranteed that at no point in time the algorithm A identifies $K \setminus H$ correctly, which concludes the proof. In the former case, the adversary adapts the enumeration by revealing in step $t_1 + 1$ the harmful labeled example $(-1, 0)$ to the algorithm. Given this new labeled pair, the previous inference of A in which H was hypothesized to be Y_0 is clearly wrong since $-1 \notin Y_0$. Next, the construction transitions to Phase 2.

Phase 2 of Construction of E. Specifically, at each round $t > t_1 + 1$, the adversary enumerates the labeled element $T(t-1)$ to the algorithm A, i.e., continuing the enumeration of elements from the sequence T starting from where phase 1 ended.

Next, we consider two cases: (i) either there is some finite step $t_2 > t_1 + 1$ such that the guess of the algorithm A for $K \setminus H$ is Q_{-2}, or (ii) there is no $t_2 \in \mathbb{N}$ such that A guesses Q_{-2}. In the latter case, the adversary commits to the choices $K = I$ and $H = Y_{-1}$ and enumeration $E = (T(1 : t_1), (-1, 0), T(t_1 + 1 : \infty))$ and it is guaranteed that at no point in time the algorithm A guesses $K \setminus H$ correctly, which concludes the proof. In the former case, the adversary adapts the enumeration by revealing in step $t_2 + 1$, the harmful labeled example $(-2, 0)$ to the algorithm. Given this new labeled pair, the previous inference of A in which H was Y_{-1} is clearly wrong since $-2 \notin Y_{-1}$. The construction then progresses to Phase 3.

Inductive Argument. Generalizing the strategy of the first two phases, the adversary continues building the target enumeration inductively. In case there is any phase $l \in \mathbb{N}$ in which the case (ii) of the case analysis is true, then the impossibility result holds for this algorithm A, and the proof is over. In case there is no such phase $l \in \mathbb{N}$, then the adversary constructed an enumeration E of labeled examples for the languages $K = I$ and $H = N \cup E$. We highlight here that $K \setminus H = I \setminus (N \cup E) = O$ is in $\mathcal{L}_K$, therefore both choices of the adversary come from the appropriate language collection, and the safe language is in $\mathcal{L}_K$. In this case, the algorithm A guesses incorrectly for infinitely many steps $t \in \mathbb{N}$. Finally, the enumeration E for this case is built inductively over all phases, i.e.,

$$\begin{aligned} E = \big(&T(1 : t_1), (-1, 0), T(t_1 + 1 : t_2), (-2, 0), \\ &T(t_2 + 1 : t_3), (-3, 0), \dots\big) \end{aligned}$$

□

Discussion. This result implies that even when safe language identification has access to harmful words from the target language, their presence alone is insufficient to guarantee consistent and correct identification. Or to put it differently, a subset of negative examples (words that belong to both K and H) is just as uninformative for identification as having no negative examples at all.

C Extended Proof for Theorem 5.1

In this appendix, we will show that safe generation in the limit from a language K given a harmful H is as hard as language identification in Angluin's setting.

We will even do so in a slightly modified, easier setting than our $\mathcal{SG}$ setting. Recall $\mathcal{SG}$ has strict requirements about the behavior of the algorithm when SG is impossible, requiring that the algorithm outputs $\perp$. In this "relaxed" $\mathcal{SG}^R$ setting, the algorithm can instead default to simply outputting any string from $\mathcal{X}$ without explicitly marking the impossibility of safe generation:

Definition 4 (Safe Generation - Relaxed $\mathcal{SG}^R$). *Fix some K from the language collection $\mathcal{L}_K = \{L_1, L_2, L_3, \ldots\}$ and some H from the language collection $\mathcal{L}_H = \{L_1, L_2, L_3, \ldots\}$. In each step $t = 1, 2, 3, \ldots$, the algorithm A observes a labeled set S_t such that each string is labeled 1 if it is a member of K and 0 if it is a member of H. At each timestep, A outputs a string a_t (its guess for an unseen string in $K \setminus H$). We say that algorithm A **safely generates from** K **given a harmful** H **in the limit** if there is some time $t' \in \mathbb{N}$ such that (i) in case $K \setminus H$ is countable, then for all steps $t > t'$ the algorithm's guess a_t belongs to $(K \setminus H) \setminus S_t$, and (ii) in case $K \setminus H$ is either finite or empty, then for all steps t the algorithm's guess a_t is in $\mathcal{X}$.*

Safe Generation $\Longrightarrow$ Language Identification. Our reduction assumes an algorithm $A^{\mathcal{SG}}$ that solves the safe generation and uses it to construct an algorithm $A^{\mathcal{SI}}$ that solves any instance of the language identification problem. Thus, if there were an easy way to solve $\mathcal{SG}$, we would have an easy way to solve $\mathcal{LI}$, which is impossible. We restate Theorem 5.1 again below:

Theorem C.1 (Theorem 5.1). *Let $A^{\mathcal{SG}}$ be an algorithm that safely generates from a language K given a harmful language H in the limit for any enumeration of the two languages from collections $\mathcal{L}_K$ and $\mathcal{L}_H$ from domain $\mathcal{X}$. Then, for any collection $\mathcal{L}$ from the domain $\mathcal{X}$, there exists an algorithm $A^{\mathcal{ID}}$ that uses $A^{\mathcal{SG}}$ as a subroutine to identify K in the limit given any enumeration of any $K \in \mathcal{L}$.*

Proof Outline. At a high level, Algorithm $A^{\mathcal{ID}}$ at time-step t will consider the first t languages from $\mathcal{L}$, and will maintain a list of the languages that are consistent with the set S_t of already revealed words from K. After some timestep t, the correct true K will be inserted into the set of consistent languages $A^{\mathcal{ID}}$ under consideration (and never removed from it). In the limit, only the correct language K and any superset languages $L_i : K \subseteq L_i$ will be consistent with S_t.

Given such a list of consistent languages,we will take advantage of the asymmetry in the outputs of the safe generation algorithm $A^{\mathcal{SG}}$. In particular, for a pair of languages L_i, L_j consistent with S_t, $A^{\mathcal{ID}}$ will construct an instance of a safe generation problem with effectively one language as the K (with $\mathcal{L}_K = \{L_i, L_i, \ldots\}$) and the other as the harmful H one ($\mathcal{L}_H = \{L_j, L_j, \ldots\}$), with an enumeration of up to t elements; running $A^{\mathcal{SG}}$ on this instance and observing its output allows ordering L_i and L_j when one is a subset of the other.

The guarantee of hypothesized $A^{\mathcal{SG}}$ for safe generation is that there exists some $t \in \mathbb{N}$ after which the returned word $w_t \in K \setminus H$, *if possible*. In cases where safe generation is impossible by construction, the algorithm must generate $\perp$. The key observation is that $\perp$ is returned when $K \subseteq H$, which $A^{\mathcal{ID}}$ uses to identify such cases.[10] Through this mechanism, $A^{\mathcal{ID}}$ can identify any pair of consistent languages that share a strict subset relationship: if $A^{\mathcal{SG}}$ generates $\perp$ on inputs from two languages L_i and (harmful) L_j, then one can assume that $L_i \subseteq L_j$. Similarly, one can enumerate L_j as the target and L_i as the harmful language to identify if $L_j \subseteq L_i$. This will effectively allow $A^{\mathcal{ID}}$ to partially order (with respect to inclusion) the list of consistent languages, and to do so in finite time.

[10] Safe generation is also impossible in the limit in the case of a finite set difference $K \setminus H$. However, in such cases, either K is also finite and hence not interesting as part of our setup, or K is not a proper subset of H, which means this $L_j = H$ will eventually be rendered inconsistent in the limit by the enumeration of $\mathcal{L}$.

Recall that the issue with language identification in the limit in the original setup was that the algorithm had no way of distinguishing between the correct language K and any language that is its superset. As the enumeration of the target language unfolds, $A^{\mathcal{ID}}$ will not only maintain a list of consistent languages (which will eventually include the target language) but also partially *order* the list by calling $A^{\mathcal{SG}}$ and observing its outputs. The partial ordering will ensure that if any two consistent languages share a strict subset relationship, then the "smaller" one will appear before the larger one in the ordered list.

In the limit, any languages that are strict subsets of the true language will be rendered inconsistent by the enumeration of K. Thus, an algorithm $A^{\mathcal{ID}}$ that returns the index of the first language in an *ordered* list constructed as above, will in the limit correctly identify the target language K.

Proof Details. Algorithm 1 shows the concrete details of how to construct an $A^{\mathcal{ID}}$ (assuming access to an efficient $A^{\mathcal{SG}}$). We discuss its building blocks one by one below:

- Given a $\mathcal{LI}$ problem setting, similar to KM, at each time step t the adversary provides the learner with another word $w_t \in K$, and the learner expands the languages it considers to include another language L_t. All observed words so far make up set S_t.
- We also assume access to a membership oracle, which receives the index i of a language and a word w and returns whether $w \overset{?}{\in} L_i$.
- Only languages consistent with S_t are of interest, which is why we will maintain a list $\mathcal{C}_t$ of consistent languages. The maximum length of the list $\mathcal{C}_t$ is, of course, t. For efficient use of the membership oracle, we will also maintain another list $\mathcal{I}_t$ with the indices of the languages of C_t.
- The crux in our algorithm is that we will also ensure that this list of consistent languages also adheres to the following invariant condition:

$$\begin{aligned}\mathcal{C}_t = [&L_i : S_t \subseteq L_i, \text{ and}\\ &\forall L_m, L_n \text{ if } L_m \subset L_n \text{ then } m < n\\ &\text{ and if } L_m = L_n \text{ and } m < n \text{ then } i_m < i_n]\end{aligned}$$

  that is:
  - $\mathcal{C}_t$ only includes languages consistent with S_t, and
  - for any two languages in $\mathcal{C}_t$, if they have a strict subset relationship (e.g. $L_m \subset L_n$), then the smaller one (in our example L_m) will appear to the left of the bigger one (L_m) in $\mathcal{C}_t$. And if the two languages are exactly the same ($L_m = L_n$), then the one with the smallest index appears to the left.
  - in case of no subset relation between two languages, there are no guarantees about their relative order in $\mathcal{C}_t$.

  This invariant condition will be guaranteed by enforcing this partial ordering of the languages in $\mathcal{C}_t$, through an ORDER() subroutine. This subroutine will read in $\mathcal{C}_t$ and the shared S_t, and will use the safe generation algorithm $A^{\mathcal{SG}}$ along with the membership oracle to ensure the partial ordering.
- The ORDER() routine will check all pairs of languages $L_i, L_j \in \mathcal{C}_t$. To decide the order, it will use the behavior of the safe generation algorithm, along with the membership oracle, to determine whether one language is a strict subset of another. Consider a generic case where we focus on a pair of languages M, N, and assume access to membership oracles $\mathcal{O}^M, \mathcal{O}^N$ for them. Now, let us cover all possible cases with respect to their relationship:
  1. $M \cap N = M$**, i.e.** $M \subset N$**:** In this case, generation from $M \setminus N$ is impossible, but generation from $N \setminus M$ is possible. Thus, running a (presumed correct) $A^{\mathcal{SG}}$ on a labeled enumeration from M, N and asking it to generate from $M \setminus N$ will return a word $w_1 \in \mathcal{X}$, while asking it to generate from $N \setminus M$ (by reversing the assignments for the true and harmful language) will indeed return a word $w_2 \in N \setminus M$.

     Now, we perform the following two checks using the oracles: $w_1 \overset{?}{\in} M$, $w_1 \overset{?}{\in} N$. Now, notice that since we are in the case where $M \subset N$, then it would be impossible for the $A^{\mathcal{SG}}$ algorithm to indeed return a $w_1 \in M$ and $w_1 \notin N$, since $M \setminus N = \emptyset$. We can perform the same check for w_2:

$w_2 \overset{?}{\in} M$ and $w_2 \overset{?}{\in} N$ and similarly decide if generation from $N \setminus M$ is (as it should be in this case) possible. Thus these two checks, under these conditions, will return a "check bitstring" equal to `01`.

2. $M \cap N = N$**, i.e.** $N \subset M$**:** Similarly as above, with reverse properties for the returned words. In this case, the two checks will return a bitstring of `10`.
3. $M \cap N = M = N$**, i.e.** $M = N$**:** In this case, generation from both $M \setminus N$ and $N \setminus M$ is impossible. Now both instances of $A^{\mathcal{SG}}$ will return arbitrary words $w_1, w_2 \in \mathcal{X}$, which will not satisfy either of the checks we will perform. Now the check bitstring will be `00`.
4. **otherwise:** In all other cases, regardless of whether $M \cap N$ is $\emptyset$ or not, both $M \setminus N$ and $N \setminus M$ are non-empty sets, and thus generation from both is possible, which we can indeed confirm with our oracle checks. The bitstring will be `11`.

Remember that the assumption here is that algorithm $A^{\mathcal{SG}}$ runs in finite time, which makes the above-described check feasible. And, importantly, each of the above cases yields a unique check bitstring, which allows us to precisely identify any pair of languages that shares a string subset or an equality relationship.

Note that, to determine whether $M \subseteq N$ for two languages M and N, it suffices to attempt to generate from $w_1 \in M \setminus N$ (which will be impossible if $M \subseteq N$), and then confirm with the membership oracles. This corresponds to cases 1 and 3 above, with a sufficient condition being that this check returns false (bitstring: `0*`).

- Going back to our algorithm 1 to lines 4–6: once a new word w_t is shared, our first order of business is to ensure that all previously consistent languages from $\mathcal{C}_{t-1}$ remain consistent, discarding any inconsistent ones, also discarding their indices from $\mathcal{I}_{t-1}$. This can be easily done in linear time by consulting the membership oracle.
- Next, we check if the new language L_t is consistent with S_t. If not, we do not need to do anything about it. If it is consistent, then we need to add it to our list $\mathcal{C}_t$ (cf. lines 8,9).
- Next, we run the ORDER() routine, thus ensuring that $\mathcal{C}_t$ maintains the invariant condition discussed above.
- Our algorithm now, if needed, inserts L_t into $\mathcal{C}_t$, comparing it against each language already in $\mathcal{C}_t$ in reverse order (from largest to smallest; cf. Alg. 1 line 11). Lines 12–18 obtain the first bit that we need for the checks described above, and lines 19–25 obtain the second bit. Now, we move language L_t to the left of the language we are comparing to (with the swap in line 28) in all cases, except when the language we compare it to is its strict subset, in which case L_t has to remain to its right (cf. lines 26-28). Note that once we encounter a language that is a strict subset of L_t, we do not perform any more checks or swaps (cf. `break` in line 27). As we discuss in our proof below, this is enough to maintain the key property we want for $\mathcal{C}_t$. Note that this process allows us to only run a linear number of comparisons.
- Last, our algorithm returns the index of the first consistent language, i.e., the left-most language in $\mathcal{C}_t$ if $\mathcal{C}_t$ is non-empty, or a random index if no consistent languages are in our list (cf lines 32-34). As discussed in our proof, this will eventually return the index of the correct language K. On a high level, we will prove that there will exist a time after which the list $\mathcal{C}_t$ will only include the correct language(s) and their superset languages, in which case all of the languages will be comparable and thus properly partially ordered.

Proof of Theorem 5.1. We need to show two things. First, that there exists some time t after which our algorithm maintains the invariant condition of $\mathcal{C}_t$. Second, our algorithm will eventually always return the index of the correct language.

Part 1 The first part is straightforward. We only modify $\mathcal{C}_t$ in two ways. First, by removing any languages rendered inconsistent by a new word. One can easily show that removing any number of languages from $\mathcal{C}_t$ does not change the relative order of the remaining languages. Thus, if the property held before the deletions, it will continue to hold after them. Second, we modify $\mathcal{C}_t$ by running the ORDER routine.

Algorithm 1 Language Identification from Safe Generation

Inputs:

- Domain $\mathcal{X}$
- Language collection $\mathcal{L}$
- Enumeration $E = (w_1, w_2, \ldots)$ of K
- Membership oracle for $\mathcal{O}(w, i)$, where $L_i \in \mathcal{L}$. ▷ `Returns True if` $w \in L_i$`, else False.`
- Safe Generation algorithm $A^{\mathcal{SG}}(E', \mathcal{L}_1, \mathcal{L}_2)$, where E' is finite and $\mathcal{L}_1, \mathcal{L}_2$ are language collections.

1: Let $\mathcal{C}_0$ initialize the empty *ordered* list of consistent languages.
2: Define function $\mathcal{I}^{\mathcal{C}\to\mathcal{L}}$ that will map the indices of languages in $\mathcal{C}$ to their indices in $\mathcal{L}$.
3: **for** $t = 1, 2, \ldots$ **do**
4: Let $S_t = \{w_1, \ldots, w_t\}$ be the sequence of observed words from (unknown) K.
5: Create $\mathcal{C}_t$ by scaning $\mathcal{C}_{t-1}$ and removing all languages $L_i \in \mathcal{C}_{t-1}$ for which $w_t \notin L_i$.
▷ `Deletions maintain the relative order of` $\mathcal{C}_{t-1}$`.`
6: **if** language L_t is consistent with S_t **then**
7: Append L_t to $\mathcal{C}_t$.
8: **end if**
9: Run ORDER$(\mathcal{C}_t, S_t, A^{\mathcal{SG}}, \mathcal{I}^{\mathcal{C}\to\mathcal{L}}, \mathcal{O})$.
10: **if** $\mathcal{C}_t$ is not empty **then**
11: **return** $\mathcal{I}^{\mathcal{C}\to\mathcal{L}}(1)$. ▷ Guess the index of the left-most consistent language.
12: **else return** 1
13: **end if**
14: **end for**

There are many ways to implement this partial ordering routine. Algorithm 2 initializes a new empty list $\mathcal{C}_{new}$, and then inserts each language from $\mathcal{C}_t$ into this new list, ensuring that the insertion maintains the invariant condition. By construction, when we insert a language L_i into $\mathcal{C}_{new}$, we insert it to the right-most position of $\mathcal{C}_{new}$, and it moves left through continuous swaps until it encounters a language that is its subset. Assuming a correct safe generation algorithm $A^{\mathcal{SG}}$, lines 6 through 12 implement the checks that we outlined above. We only perform a check for whether the left language is a subset of the right one, in which case the language on the right does not move further left (cf. `break` in line 14). Thus, it will be impossible for any language that we insert in $\mathcal{C}_{new}$ to end up to the left of any language that is its strict subset or its equal. For all languages that L_i might have skipped and now lie to its right, their relative positions among them do not change, and neither do their relative positions with any languages left of L_i that might be strict subsets of any languages to the right of L_i. Thus, if the property (which holds trivially for $n = 1$ languages in $\mathcal{C}_{new}$) held when we have n languages in $\mathcal{C}_{new}$, it will continue to hold after the insertion of language $n + 1$ into $\mathcal{C}_{new}$.

The final condition for this to work properly is that the safe generation algorithm $A^{\mathcal{SG}}$ is correct (which is addressed in the second part of the proof).

Part 2 Let $z \in \mathbb{N}$ be the smallest index such that $L_z = K$.

Now, by construction, there will exist a time t^* after which the correct language L_z will come into consideration by our algorithm. Obviously, L_z will be consistent and will be inserted into $\mathcal{C}_{t^*}$ at some position following the ORDER routine. Once inserted, L_z will never be removed from $\mathcal{C}_t$.

Again by construction, there will exist a time t^{**} after which the safe generation algorithm $A^{\mathcal{SG}}$ will be correct for all enumerations and language choices from $\mathcal{L}$.[11] This entails that after time t^{**} we are guaranteed that our checks that ensure the partial ordering of the languages in $\mathcal{C}_t$ are correct, i.e., for all $t \geq t^{**}$ the invariant condition for C_t holds.

Now, consider the ordered state of $\mathcal{C}_t$ after some time $t > \max(t^*, t^{**})$:

$$\mathcal{C}_t = \left[L_{j_1}, L_{j_2}, \ldots, L_z, \ldots L_{j_n}\right]$$

Since $A^{\mathcal{SG}}$ is correct, then any consistent language $L_{\leftarrow}$ that ends up to the left of the correct language L_z can only belong in one of the following two categories:

1. $L_{\leftarrow} \subset L_z$.

[11] Note that the safe generation algorithm is always called on the same language collection $\mathcal{L}$.

Algorithm 2 Order Language Collection with Safe Generation

Inputs:

- List of consistent languages $\mathcal{C}_t$
- Enumeration $S_t = (w_1, w_2, \ldots)$
- Safe Generation algorithm $A^{\mathcal{SG}}(E', \mathcal{L}_1, \mathcal{L}_2)$, where E' is finite and $\mathcal{L}_1, \mathcal{L}_2$ are language collections.
- Membership oracle for $\mathcal{O}(w, i)$, where $L_i \in \mathcal{L}$. ▷ `Returns True if` $w \in L_i$, `else False.`
- Function $\mathcal{I}^{\mathcal{C}\to\mathcal{L}}$ that maps indices of languages in $\mathcal{C}_t$ to their indices in $\mathcal{L}$

Let $\mathcal{C}_{new}$ initialize an empty ordered list of consistent languages
▷ Order languages in $\mathcal{C}_t$ in ascending index order
Let $\mathcal{L}^{\text{sorted}} \leftarrow \left[L_c : L_c \in \mathcal{C}_t, \text{ordered by index in } \mathcal{L}\right]$
for all languages $L_c \in \mathcal{L}^{\text{sorted}}$ **do**
    Append L_c to $\mathcal{C}_{new}$.
    Let $n \leftarrow |\mathcal{C}_{new}|$ ▷ `The length of` $\mathcal{C}_{new}$
    for $j = n, n-1, \ldots 2$ **do**
        `** Block where we attempt to generate from` $\mathcal{C}_{new}[j-1] \setminus \mathcal{C}_{new}[j]$ `**`
        ▷ `Create enumeration where` $\mathcal{C}_{new}[j-1]$ `acts as` K^α `and` $\mathcal{C}_{new}[j]$ `acts as` H^α.
        Create labeled enumeration E_t^α sampling t new words from $\mathcal{C}_{new}[j-1]$ and $\mathcal{C}_{new}[j]$.
        ▷ `Run` $A^{\mathcal{SG}}$ `on` E_t^α `to obtain a guessed word (ideally from` $\mathcal{C}_{new}[j-1] \setminus \mathcal{C}_{new}[j]$`).`
        $\alpha \leftarrow A^{\mathcal{SG}}(E_t^\alpha, \mathcal{L}, \mathcal{L})$
        ▷ `Perform checks with oracles:`
        Set $\texttt{isInK}^\alpha \leftarrow \mathcal{O}(\alpha, \mathcal{I}^{\mathcal{C}\to\mathcal{L}}(j-1))$ and $\texttt{isInH}^\alpha \leftarrow \mathcal{O}(\alpha, \mathcal{I}^{\mathcal{C}\to\mathcal{L}}(j))$
        if $\texttt{isInK}^\alpha$`=TRUE and` $\texttt{isInH}^\alpha$`=FALSE` **then**
            ▷ `Safe Generation from` $\mathcal{C}_{new}[j-1] \setminus \mathcal{C}_{new}[j]$ `is possible.`
            Let $\texttt{isInK}^\alpha\texttt{-H}^\alpha \leftarrow$ `TRUE`
        else Let $\texttt{isInK}^\alpha\texttt{-H}^\alpha \leftarrow$ `FALSE`
        end if
        ▷ `Swap if needed`
        if $\neg(\texttt{isInK}^\alpha\texttt{-H}^\alpha)$ **then**
            break ▷ `must not bubble left because` $\mathcal{C}_{new}[j-1] \subseteq \mathcal{C}_{new}[j]$
        else swap$(\mathcal{C}_{new}[j-1], \mathcal{C}_{new}[j])$ ▷ `bubble left in all other cases`
        end if
    end for
end for
return $\mathcal{C}_{new}$

2. $L_\leftarrow \cap L_z \subset L_z$, and $L_\leftarrow \setminus L_z \neq \emptyset$: these are languages that include a subset of the correct language (but not all of it) as well as other strings outside of L_z.

Now for all such languages $L_\leftarrow$ in the above two categories there exists a time t' that they will be considered, and also by construction there will exist a time t'' at which point a word $w_{t''} \in L_z$ and $w_{t''} \notin L_\leftarrow$ will be revealed to render them inconsistent with S_t. Thus, there exists a time t^{***} that is the maximum of all possible t' and t'' for all such languages.

Finally, let's consider the state of $\mathcal{C}_t$ after some time $t > \max(t^*, t^{**}, t^{***})$, which we argue that it will be:

$$\mathcal{C}_t = \left[L_z, \ldots L_{j_n}\right].$$

All languages that may include a strict subset of the correct L_z have been seen, since we are after t^{***}. That is, all languages $L_\leftarrow$ that could have ended up to the left of L_z have been discarded. Thus, for us to consider that there only exists

1. inconsistent languages (which we will never insert into $\mathcal{C}_t$);
2. languages that are repetitions of the correct language; and
3. languages that are strict supersets of the correct language.

Even though any language $L_\rightarrow$ from the last two categories will be consistent and we will insert it into $\mathcal{C}_t$,[12] our algorithm will keep them to the right of L_z, since (a) $L_z \subseteq L_\rightarrow$, and this is exactly the invariant condition that C_t satisfies (the ORDER routine does *not* perform a swap to the left for these languages when it encounters L_z to its left), and (b) for any repetitions of the correct language the one with the smallest index z is guaranteed to be to the left of the others by the ORDER routine.

Thus, our algorithm after some time $t > \max(t^*, t^{**}, t^{***})$ will always return the smallest index z of the correct language. □

[12] In the end, these are the languages that are problematic to choose among in Angluin's setting.

D Proof for Theorem 6.1

Proof. In this elementary proof, we will show that if $L_{\texttt{DIFF-EMPTY}}$ is decidable, then we can build a decider for $L_{\texttt{HALT}}$, which is the language for the halting problem. Suppose for the sake of contradiction that $L_{\texttt{DIFF-EMPTY}}$ is decidable. Then, there must be a Turing machine D that is the decider of $L_{\texttt{DIFF-EMPTY}}$. Next, we build a decider M^* for $L_{\texttt{HALT}}$ by using D. The TM M^* takes as input the description of a TM $< T >$ and an input string s. Let $X_1 = \mathbb{N}$ be a set whose members are the natural numbers. Let B be a set such that if (i) $T(s)$ halts then $X_2 = \mathbb{N}$, otherwise (ii) $X_2 = \mathbb{N} \setminus \{0\}$. That is, the set X_2 contains 0 only if $T(s)$ halts. Thus, the TM M^* feeds the above two sets to the decider of $L_{\texttt{DIFF-EMPTY}}$, namely D, and outputs whatever D outputs. It is easy to see that membership in $L_{\texttt{DIFF-EMPTY}}$ implies that the input TM T halts on s, thus M^* is a decider for $L_{\texttt{HALT}}$ which is a contradiction since $L_{\texttt{HALT}}$ is undecidable. □

E Extended Proof of Theorem 6.2

Proof. In this proof, we make a diagonalization argument for two language collections that we construct. Suppose that the following property holds for $\mathcal{L}_K = \{L_1^K, L_2^K, \ldots\}$ and $\mathcal{L}_H = \{L_1^H, L_2^H, \ldots\}$:

> (**Property** P^*) There exists an index i^K and i^H for which $L_{i^K}^K \subseteq L_{i^H}^H$, such that for every pair of finite sets of strings (T^K, T^H) such that $T^K \subset L_{i^K}^K$ and $T^H \subset L_{i^H}^H$ the following holds:
>
> There exists a $j \geq 1$ and an associated $j^* \geq 1$, such that (1) if $T^K \subseteq L_j^K$, then L_j^K is a proper subset of $L_{i^K}^K$, and (2) if $T^H \subseteq L_{j^*}^H$, then $L_{j^*}^H$ is a proper subset of $L_{i^H}^H$, and (3) $L_j^K \setminus L_{j^*}^H$ has infinite cardinality.

Let $E_{i^K}^K$ be any enumeration of the L_{i^K} and $E_{i^H}^H$ be any enumeration of L_{i^H}. We will show that for any generator G there exists an adversary that enumerates from $E_{i^K}^K$ and $E_{i^H}^H$ such that G cannot safely generate from L_{i^K} in the limit.

Phase 1. Let x_1^K be the first element of the enumeration $E_{i^K}^K$, i.e., $x_1 = E_{i^K}^K(1)$. Set T^K to be $\{x_1^K\}$. Let x_1^H be the first element of the enumeration $E_{i^H}^H$, i.e., $x_1 = E_{i^H}^H(1)$. Set T^H to be $\{x_1^H\}$.

From property P^* we know that there must be an i_1 and an i_1^* such that (1) $T^K \subseteq L_{i_1}^K$ where $L_{i_1}^K$ is a proper subset of $L_{i^K}^K$, and (2) $T^H \subseteq L_{i_1^*}^H$ where $L_{i_1^*}^H$ is a proper subset of $L_{i^H}^H$, and (3) $L_{i_1}^K \setminus L_{i_1^*}^H$ has infinite cardinality.

Subphase 1.A: Consider an enumeration of $L_{i_1}^K$ constructed by traversing the fixed enumeration $E_{i^K}^K$ and using only the elements of $L_{i_1}^K$ that appear in it and transfer them to $E_{i_1}^K$ in the same order. Consider an enumeration of $L_{i_1^*}^H$ constructed by traversing the fixed enumeration $E_{i^H}^H$ and using only the elements of $L_{i_1^*}^H$ that appear in it and transfer them to $E_{i_1^*}^H$ in the same order.

We note that the infinite sequence $E_{i_1}^K$ (resp. $E_{i_1^*}^H$) is indeed an enumeration of $L_{i_1}^K$ (resp. $L_{i_1^*}^H$) since $L_{i_1}^K$ (resp. $L_{i_1^*}^H$) is a proper subset of L_{i^K} (resp. L_{i^H}), therefore, all of its strings appear in $E_{i^K}^K$ (resp. $E_{i^H}^H$). At any round t the adversary presents the string $E_{i_1}^K(t)$ and $E_{i_1^*}^H(t)$ to the generator G.

Consider two cases: (i) either there is some finite $t_1 \in \mathbb{N}$ such that the generator G will generate an element from $L_{i_1}^K \setminus L_{i_1^*}^H$, or (ii) there is no such $t_1 \in \mathbb{N}$. Notice that the adversary can verify whether the algorithm is in case (i), by observing a generated string x such that the membership oracle of $L_{i_1}^K$ says "member" while the membership oracle of H_{j_1} says "non-member". Analogously, the adversary can verify whether the algorithm is in case (ii), by observing no generated string x such that the membership oracle of $L_{i_1}^K$ says "member" and the membership oracle of $L_{i_1^*}^H$ says "non-member".

In case the adversary is dealing with case (ii), then the adversary simply picks $K = L_{i_1}^K$, $H = L_{i_1^*}^H$, and their enumerations $E_{i_1}^K$ and $E_{i_1^*}^H$ and the impossibility follows from the fact that G never generates a string from $L_{i_1}^K \setminus L_{i_1^*}^H$. In case the adversary is dealing with case (i), then let $\hat{x}_1$ be the first string generated by the algorithm for which $\hat{x}_1 \in L_{i_1}^K$ and $\hat{x}_1 \notin L_{i_1^*}^H$, i.e., a safe string under the current setting. At this point, the adversary takes a conservative approach and assumes that the generator has the ability to safely generate from $L_{i_1}^K$ when the harmful language is $L_{i_1^*}^H$ (even if this is not necessarily true). For the remainder of this subphase, the adversary will take a few "housekeeping" actions to prepare for the next phase.

Suppose that string $E_{i_1}^K(t_1)$ appears in position t_1' in $E_{i^K}^K$, i.e., the adversary has traversed up to position t_1' in the original enumeration $E_{i^K}^K$ in order to generate the t_1-th string from the enumeration $E_{i_1}^K$ of this

phase. Notice that t'_1 must be finite, i.e., $t'_1 \neq \infty$. Define as S_1^K the strings from $E_{i_K}^K[1:t'_1]$ that are not in $E_{i_1}^K[1:t_1]$, i.e., all the string from the original enumeration that were bypassed to pretend an enumeration from $L_{i_1}^K$.

An analogous argument is made for the harmful language. Suppose that string $E_{i_1}^H(t_1)$ appears in position t''_1 in $E_{i_H}^H$. Notice that t''_1 must be finite. Define as S_1^H the strings from $E_{i_H}^H[1:t''_1]$ that are not in $E_{i_1}^H[1:t_1]$, i.e., all the string from the original enumeration that were bypassed to pretend an enumeration from $L_{i_1}^H$.

If $S_1^K \neq \emptyset$ or $S_1^H \neq \emptyset$, then we proceed to Subphase 1.B; otherwise, if $S_1^K = \emptyset$ and $S_1^H = \emptyset$, we proceed straight to Subphase 1.C. We denote with $S_{t_1}^K$ (resp. $S_{t_1}^H$) the strings that have been enumerated so far, i.e., $E_{i_1}^K[1:t_1]$ (resp. $E_{i_1}^H[1:t_1]$).

Subphase 1.B (Add Skipped Elements): In this subphase, the adversary enumerates all the leftover strings from S_1^K and S_1^H so that we are guaranteed that G has seen every single string in sequences $E_{i_K}^K[1:t'_1]$ and $E_{i_H}^H[1:t''_1]$. This action guarantees no "left-over" strings in $E_{i_K}^K$ and $E_{i_H}^H$ due to the (unsuccesful) attempt of the adversary to trick G by enumerating as if the choice is $K = L_{i_1}^K$, $H = L_{i_1^*}^H$. Proceed to Subphase 1.C.

Subphase 1.C (Nothing Skipped; Issue element outside $L_{i_1}^K$): Recall that G has seen all elements in $E_{i_K}^K[1:t'_1]$. Now the adversary needs to enumerate a string from the new true language that contradicts the hypothesis $K = L_{i_1}^K$ of generator G.

We first argue that there exists a string y_1 in $E_{i_K}^K[t'_1:\infty]$ that does not belong to $L_{i_1}^K$, such that, when the adversary enumerates it, the generator G will recognize that $K \neq L_{i_1}^K$. From property P^*, we know that $L_{i_1}^K \subsetneq L_{i_K}^K$, therefore there exists a string y_1 in $L_{i_K}^K$ that does not belong to $L_{i_1}^K$. Since $E_{i_K}^K$ contains all strings in $L_{i_1}^K$ and since y_1 has not been enumerated so far, it must be in $E_{i_K}^K[t'_1:\infty]$. Let $t_{y_1} > t'_1$ be the position of $E_{i_K}^K$ that is equal to y_1. As a final step, the adversary enumerates all the strings in $E_{i_K}^K[t'_1+1:t_{y_1}]$ and proceeds to phase 2.

Phase l. For the l-th phase, we will use the fact that P^* holds but this time, the T that we use is $E_{i_K}^K[1:t_{y_l}]$. That is, let $T^K = \{E_{i_K}^K[1:t_{y_l}]\}$ and $T^H = \{E_{i_H}^H[1:t''_l]\}$. Then from P^*, there exists a $i_l \geq 1$ and an associated $i_l^* \geq 1$, such that (1) $T^K \subseteq L_{i_l}^K$ and $L_{i_l}^K$ is a proper subset of $L_{i_K}^K$, and (2) $T^H \subseteq L_{i_l^*}^H$ and $L_{i_l^*}^H$ is a proper subset of $L_{i_H}^H$, and (3) $L_{i_l}^K \setminus L_{i_l^*}^H$ has infinite cardinality. Additionally, notice that $L_{i_l}^K$ is different from all $L_{i_1}^K, \ldots, L_{i_{l-1}}^K$ because it contains string y_l which does not appear in any of these languages.

The arguments after this point are analogous to the ones in phase 1.

Subphase l.A: Consider an enumeration of $L_{i_l}^K$ constructed by traversing the fixed enumeration $E_{i_K}^K$ and using only the elements of $L_{i_l}^K$ that appear in it and transfer them to $E_{i_l}^K$ in the same order. Consider an enumeration of $L_{i_l^*}^H$ constructed by traversing the fixed enumeration $E_{i_H}^H$ and using only the elements of $L_{i_l^*}^H$ that appear in it and transfer them to $E_{i_l^*}^H$ in the same order.

Consider two cases: (i) either there is some finite $t_l \in \mathbb{N}$ such that the generator G will generate an element from $L_{i_l}^K \setminus L_{i_l^*}^H$, or (ii) there is no such $t_l \in \mathbb{N}$. In case the adversary is dealing with case (ii), then the adversary simply picks $K = L_{i_l}^K$, $H = L_{i_l^*}^H$, and their enumerations $E_{i_l}^K$ and $E_{i_l^*}^H$ and the impossibility follows from the fact that G never generates a string from $L_{i_l}^K \setminus L_{i_l^*}^H$. In case the adversary is dealing with case (i), then let $\hat{x}_l$ be the first string generated by the algorithm for which $\hat{x}_l \in L_{i_l}^K$ and $\hat{x}_l \notin L_{i_l^*}^H$, i.e., a safe string under the current setting. At this point, the adversary takes a conservative approach and assumes that the generator has the ability to safely generate from $L_{i_l}^K$ when the harmful language is $L_{i_l^*}^H$ (even if this is not necessarily true).

Suppose that string $E_{i_1}^K(t_l)$ appears in position t'_l in $E_{i_K}^K$. Notice that t'_l must be finite, i.e., $t'_l \neq \infty$. Define as S_l^K the strings from $E_{i_K}^K[1:t'_l]$ that are not in $E_{i_1}^K[1:t_l]$, i.e., all the string from the original enumeration that were bypassed to pretend an enumeration from $L_{i_l}^K$.

An analogous argument is made for the harmful language. Suppose that string $E_{i_l}^H(t_l)$ appears in position t''_l in $E_{i_H}^H$. Notice that t''_l must be finite. Define as S_l^H the strings from $E_{i_H}^H[1:t''_l]$ that are not in $E_{i_1}^H[1:t_l]$, i.e., all the string from the original enumeration that were bypassed to pretend an enumeration

from $L^H_{i_l}$.

If $S^K_l \neq \emptyset$ or $S^H_l \neq \emptyset$, then we proceed to Subphase l.B; otherwise, if $S^K_l = \emptyset$ and $S^H_l = \emptyset$, we proceed straight to Subphase 1.C. We denote with $S^K_{t_l}$ (resp. $S^H_{t_l}$) the strings that have been enumerated so far, i.e., $E^K_{i_l}[1 : t_l]$ (resp. $E^H_{i_l}[1 : t_l]$).

Subphase l.B (Add Skipped Elements): In this subphase, the adversary enumerates all the leftover strings from S^K_l and S^H_l so that we are guaranteed that G has seen every single string in sequences $E^K_{i^K}[1 : t'_l]$ and $E^H_{i^H}[1 : t''_l]$. This action guarantees no "left-over" strings in $E^K_{i^K}$ and $E^H_{i^H}$ due to the (unsuccesful) attempt of the adversary to trick G by enumerating as if the choice is $K = L^K_{i_l}$, $H = L^H_{i^*_l}$. Proceed to Subphase l.C.

Subphase l.C (Nothing Skipped; Issue element outside $L^K_{i_1}$): Recall that G has seen all elements in $E^K_{i^K}[1 : t'_l]$. Now the adversary needs to enumerate a string from the new true language that contradicts the hypothesis $K = L^K_{i_l}$ of generator G.

We first argue that there exists a string y_l in $E^K_{i^K}[t'_l : \infty]$ that does not belong to $L^K_{i_l}$, such that, when the adversary enumerates it, the generator G will recognize that $K \neq L^K_{i_l}$. From property P^*, we know that $L^K_{i_l} \subsetneq L^K_{i^K}$, therefore there exists a string y_l in $L^K_{i^K}$ that does not belong to $L^K_{i_l}$. Since $E^K_{i^K}$ contains all strings in $L^K_{i_l}$ and since y_l has not been enumerated so far, it must be in $E^K_{i^K}[t'_l : \infty]$. Let $t_{y_l} > t'_l$ be the position of $E^K_{i^K}$ that is equal to y_l. As a final step, the adversary enumerates all the strings in $E^K_{i^K}[t'_l + 1 : t_{y_l}]$ and proceeds to phase 2.

Inductive Argument. As explained, we continue the construction of the target enumeration inductively. If there exists some phase $(l + 1)$ such that Case (ii) (in Subphase A) is activated, then the impossibility follows. Now, assume that Case (ii) is not activated for any phase $(l + 1) \in \mathbb{N}$. In that case, the adversary revealed an enumeration of $L^K_{i^K}$ and $L^H_{i^H}$ such that G does not safely generate consistently from $L^K_{i^K} \setminus L^H_{i^H}$ which is empty. To see this, recall that at the time of transitioning to a new phase, the generator generated a string that is not ϵ, which is not a safe generation when $K = L^K_{i^K}$ and $H = L^H_{i^H}$. Since there are infinitely many such phases, there are infinitely many such timesteps so the generator fails. The reason that the induction can proceed for t to infinity is that $L^K_{i^K}$ has infinite cardinality; thus, we can have an infinite number of phases where the algorithm cannot safely generate. Then, in this case, the impossibility follows by setting the target language $K = L^K_{i^K}$ and $H = L^H_{i^H}$ and using the enumeration we have inductively constructed over all phases.

□