CIS 6200: Emerging Theory of LLMs **Fall 2026**

Topic 1A: Generation from Positive Examples

Last updated: September 1, 2026 *Surbhi Goel*

Acknowledgements: *These notes draw on Chomsky [1], Gold [2], Kleinberg and Mullainathan [4], the COLT 2025 language-generation tutorial [5], Charikar and Pabbaraju [6], and Kalavasis, Mehrotra, and Velegkas [7].*

AI Disclosure: *These notes were drafted and revised with the assistance of AI tools. I have reviewed the content and take responsibility for it. If you find an error, please let me know.*

In previous lectures, we described a language model as a probability distribution over complete responses. Here we set the probabilities aside and ask a more basic question. Suppose the learner sees only *positive examples*, meaning outputs known to be valid. A useful generator must go beyond repeating these examples and produce new sentences, proofs, programs, or responses. The difficulty is that an unseen output may be invalid, or it may be valid and simply not have appeared yet.

Gold formalized learning from positive examples in 1967, motivated in part by a child learning to speak [2]. A child hears utterances from a language without receiving explicit grammatical rules or labeled counterexamples. Gold represented a language as a set of strings and asked whether a learner could eventually *identify* that set from a stream of its members.

> I wish to construct a precise model for the intuitive notion "able to speak a language" in order to be able to investigate theoretically how it can be achieved artificially. Since we cannot explicitly write down the rules of English which we require one to know before we say he can "speak English," an artificial intelligence which is designed to speak English will have to learn its rules from implicit information. That is, its information will consist of examples of the use of English and/or of an informant who can state whether a given usage satisfies certain rules of English, but cannot state these rules explicitly.

Gold's original motivation for formalizing language learning (1967).

Identification is demanding because a child may learn to speak grammatical English without characterizing the complete set of grammatical sentences. Gold showed that some broad language families cannot be identified from positive examples alone.[1]

Kleinberg and Mullainathan [4] recently separated generation from identification. Rather than recover the whole language, the learner only has to produce a new valid example. They asked: *Can a learner produce a new valid example after seeing only positive examples?*

[1] His result applies, for example, to the family of all regular languages, whose members are the languages recognized by finite-state machines. Angluin later characterized which families can be identified from positive data [3].

1 The Positive-Data Game

We study this question as a game between an adversary and a learner. In each round, the adversary presents an example and the learner replies, as shown in Figure 1. We begin by describing the objects they exchange, then state the rules and what it means for the learner to win.

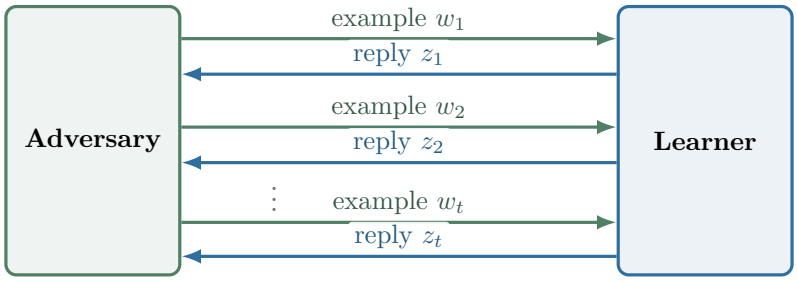

Figure 1: The adversary reveals one valid example, and the learner replies. The same two moves repeat each round.

Every example and reply is a finite sequence of tokens from a fixed finite vocabulary. There are infinitely many such strings because their length is unbounded, but we can list them all by first sorting them by length and then ordering the strings of each length. Thus the set of finite strings is countable. We denote this universe by $\mathcal{U}$. Programs, proofs, and natural language responses all fit the model once we write them as finite text. A *language* is a set $L \subseteq \mathcal{U}$. It may contain infinitely many outputs even though each output is a finite string. An output is valid for L when it belongs to L.

The players and the target. Both players know a list of possible target languages. We index this list as

$$\mathcal{F} = (L_1, L_2, \ldots)$$

where each $L_i \subseteq \mathcal{U}$ is one possible set of valid outputs. The list may be infinite. Calling $\mathcal{F}$ countable means exactly that its candidates can be arranged in such a list. We keep this order fixed. In particular, this includes families whose candidates have finite descriptions, such as grammars or programs, because all finite descriptions can be listed.

Before the first round, the adversary secretly chooses one candidate $L^\star \in \mathcal{F}$. We call $L^\star$ the *target language*. The learner knows the family $\mathcal{F}$, but it does not know which candidate was chosen and cannot query membership in $L^\star$.

The rounds. In round t, the adversary first reveals an example $w_t \in L^\star$. The learner sees the examples $w_1, \ldots, w_t$ revealed so far and replies with an output z_t. It receives no feedback about whether z_t belongs to $L^\star$.

The adversary controls the order of the examples, but it must eventually reveal every member of the target. Thus each w_t belongs to $L^\star$, and every $w \in L^\star$ appears at some finite time.

We call a sequence satisfying both conditions a *positive text* for $L^\star$. The adversary may repeat examples. We write

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

for the distinct examples observed by the end of round t.

Requiring every target output to appear eventually prevents the adversary from hiding almost all information forever. Suppose one target contains 0 and the positive even integers, while another

contains 0 and the positive odd integers. If the adversary could repeat 0 forever, the learner could never tell which target was chosen. Every unseen integer would be invalid for one of the two targets, so no unseen reply could be guaranteed to be valid.

Even with this rule, the adversary has considerable freedom. It may repeat familiar examples and delay any particular member of $L^\star$ for an arbitrarily long time. Seeing w tells the learner that $w \in L^\star$. Not seeing w by time t tells the learner nothing. The adversary never supplies an invalid example, but it may choose the target and its presentation order to make the learner's task as difficult as possible.

Winning the game. Because informative examples can be delayed, we do not require the learner to succeed immediately. We allow finitely many early mistakes. After some finite time, every reply must be valid and must not have appeared in the input stream.

Definition 1.1 (Generation in the limit, Kleinberg–Mullainathan [4])**.** A learner *generates $L^\star$ in the limit* if, for every positive text for $L^\star$, there is a time T such that

$$z_t \in L^\star \setminus S_t \qquad \text{for every } t \geq T.$$

The family $\mathcal{F}$ is *generatable in the limit* if one learner succeeds for every $L^\star \in \mathcal{F}$.

One learner must work for every target and every positive text for that target. The finite time T may depend on both the target and the order in which its examples appear, and the learner need not know when T has arrived.

When we call a generated output *new*, we mean that it is absent from the observed set S_t. It may still repeat an earlier generator output.

Why every candidate is infinite. We restrict attention to families in which every L_i is infinite. If the target $L^\star$ were finite, a positive text would eventually reveal all of it, giving $S_t = L^\star$. At that point $L^\star \setminus S_t$ would be empty, so no new valid output would remain. Each individual output is still a finite string.

Identification. We can use the same adversary and the same input stream to define the classical identification problem. Only the learner's reply changes. Instead of producing one output, it proposes the entire target language.

Definition 1.2 (Identification in the limit, Gold [2])**.** An identifier proposes one candidate from $\mathcal{F}$ after each round. It *identifies $L^\star$ in the limit* if, for every positive text for $L^\star$, there is a time T after which its proposal is $L^\star$ in every round. The family $\mathcal{F}$ is *identifiable in the limit* if one identifier succeeds for every $L^\star \in \mathcal{F}$.

Both criteria permit finitely many early mistakes. Eventually, a generator must produce a valid output outside S_t, whereas an identifier must name the entire target language.

For example, after hearing several grammatical English sentences, generation asks for one new grammatical sentence. Identification asks for an exact description of the entire set of grammatical sentences.

2 Why Generation Can Be Easier Than Identification

We first study a concrete family that separates the two tasks. Its languages are nested sets of integers, where each integer is viewed as the finite string that writes it in decimal notation.

$$A_m = \{-m, -m+1, -m+2, \ldots\} \quad (m = 0, 1, 2, \ldots), \qquad A_\infty = \mathbb{Z}.$$

Thus A_0 contains the nonnegative integers, A_1 also contains -1, and each subsequent language extends one step farther to the left. The symbol ∞ is only a convenient name for one additional candidate. The family is still countable.

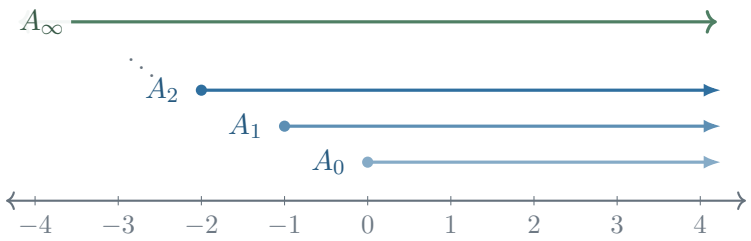

Figure 2: Each dot marks the left endpoint $-m$ of A_m. Thus $A_0 \subset A_1 \subset \cdots \subset A_\infty = \mathbb{Z}$.

2.1 A Generator That Moves to the Right

Every language in this family is closed to the right. Once an integer x is valid, $x + 1$ is also valid. This gives us a safe direction in which to move. After observing a nonempty set S_t, we output

$$z_t = \max S_t + 1.$$

Since S_t is finite and nonempty, its maximum exists. The output is new because it is larger than every observed integer. It is also valid. If the target is A_m, every observed integer is at least $-m$, so $z_t = \max S_t + 1 \geq -m + 1$ and hence $z_t \in A_m$. If the target is $A_\infty = \mathbb{Z}$, every integer is valid.

We have shown that $z_t \in L^\star \setminus S_t$ in every round. The rule succeeds immediately even though it never determines which language the adversary chose.

2.2 Why No Identifier Can Succeed

The difficulty is deciding whether the language has a left endpoint. Suppose the examples seen so far are $0, 1, \ldots, r$, with no negative integer. The target could be A_0, or the negative examples from a larger language might simply be delayed. Positive data alone cannot distinguish these possibilities.

Suppose, for contradiction, that one identifier succeeds on every language in the family. We fix its code, so its guess after any sequence of examples is determined. We use those guesses to construct a positive text for $A_\infty = \mathbb{Z}$ on which the identifier fails. The target remains $A_\infty = \mathbb{Z}$ throughout. Only the examples revealed so far are made to look temporarily like A_m.

At each stage we force the identifier to commit to a finite left endpoint, then reveal one more negative integer and refute that guess. The construction is as follows.

Constructing a difficult positive text
Initialize $n = 0$, the next nonnegative integer that has not appeared. Run stages $m = 0, 1, 2, \ldots$. At the beginning of stage m, the negative integers from $-m$ through -1 have already appeared. This set is empty when $m = 0$. Then:

1. **Force the guess A_m.** Reveal n, increase n by one, and check the identifier's guess. Until it proposes A_m, keep revealing fresh nonnegative integers in this way.
2. **Refute the guess A_m.** Once the identifier proposes A_m, reveal $-(m+1)$ and begin stage $m+1$.

We first verify that every stage ends. If we remained at stage m forever, Step 1 would reveal every nonnegative integer. Together with the negative integers $-1, \ldots, -m$ revealed in earlier stages, the complete stream would be a positive text for $A_m = \{-m, -m+1, \ldots\}$. Our assumed identifier must converge to A_m on this text, so it must eventually propose A_m. At that moment the construction leaves the stage.

Since every stage ends, the construction passes through stages $0, 1, 2, \ldots$. Each stage reveals at least one new nonnegative integer, and the counter n never restarts. We therefore reveal every nonnegative integer. Moving from stage m to stage $m+1$ reveals $-(m+1)$, so we also reveal $-1, -2, \ldots$. The resulting stream contains every integer and is a positive text for $A_\infty = \mathbb{Z}$.

The identifier does not converge on this text. At the end of stage m, it proposes A_m, which is not the target $\mathbb{Z}$. It may correct this proposal later, and the definition allows such temporary mistakes. The problem is that there are infinitely many stages, and their ending times grow without bound. Given any time T, some later stage ends after T, producing another wrong proposal. There is no time after which every proposal equals $\mathbb{Z}$. This contradicts our assumption that the identifier succeeds on every language in the family.

3 Universal Generation for Countable Families

The nested-tail family shows that a countable collection of infinite languages need not be identifiable from positive data. At the same time, its simple right-moving generator suggests a broader possibility: perhaps generation succeeds for every countable family of infinite languages. Kleinberg and Mullainathan prove that it does.

Theorem 3.1 (Universal generation [4])**.** *Every countable family of infinite languages is generatable in the limit.*

The successful strategy is easiest to understand after seeing why two natural approaches fail.

3.1 Two Natural Strategies and Why They Fail

After seeing S_t, the learner can rule out any candidate that omits an observed example: such a candidate cannot be the target. We say that L_i *survives* at time t if it contains every observed example, or equivalently, if $S_t \subseteq L_i$. The target $L^\star$ always survives.

First idea: generate from the common intersection. The nested-tail generator followed this principle: its output $\max S_t + 1$ belonged to every tail that still survived. More generally, any unseen output that belongs to every survivor is safe because the target is one of them.

The common intersection, however, may contain nothing new. Consider a family whose candidates are

$$\mathbb{N} = \{1, 2, \ldots\}, \qquad B_i = \mathbb{N} \setminus \{i\}, \qquad i = 1, 2, \ldots$$

and let the target be $\mathbb{N}$. Thus B_i contains every positive integer except i. Suppose, for example, that $S_t = \{2, 5\}$. For every unseen number j, the candidate B_j still survives and excludes j. No unseen number can therefore belong to every surviving candidate. In fact,

$$\bigcap_{i \notin S_t} B_i = S_t.$$

Although every fixed candidate B_i is eventually eliminated when i appears, there is no single finite time by which all bad candidates have been eliminated. At every finite time, infinitely many integers remain unseen, so infinitely many restrictive candidates still survive.

Second idea: follow the first surviving candidate. Order the candidates as $L_1, L_2, \ldots$ and output an unseen member of the first survivor. If a new example falls outside that candidate, eliminate it and move to the next survivor. Any candidate that omits an element of the target is eventually eliminated because the positive text must reveal that element.

The problem is that positive data can never eliminate a candidate that contains the target. Suppose the target is the even numbers and an earlier candidate is all of $\mathbb{N}$. Every observed example belongs to $\mathbb{N}$, so the learner never moves on. Membership in this surviving candidate does not certify validity because its unseen members include odd numbers outside the target.

3.2 The Successful Strategy: Keep Shrinking

The two strategies fail in opposite ways. Intersecting every survivor can be too cautious, while following the first survivor can be too permissive. We combine the useful parts of both ideas. The learner chooses outputs from one survivor at a time, but it does not remain with the first survivor forever. As it moves through the candidate list, it switches only to a later survivor contained in its current choice. The new candidate still contains every observed example, while offering fewer outputs that might be invalid.

At time t, we inspect only $L_1, \ldots, L_t$ and rerun the following scan from scratch. This makes the scan finite in every round, while ensuring that it eventually reaches every fixed candidate. During the scan, C denotes the candidate currently being held. If none of the first t candidates survives, the learner outputs arbitrarily and stops for this round. Otherwise:

1. Start with the first survivor and call it C.
2. Continue from left to right. Whenever a later survivor L_i satisfies $L_i \subseteq C$, replace C by L_i.
3. After the scan ends, output any $z_t \in C \setminus S_t$.

We first show that such a learner exists; we do not claim that this particular scan is efficient. We allow it to test exact containment between candidate languages and to choose an unseen member of one of them. It still cannot ask whether an output belongs to the hidden target.

If $L_k = L^\star$, then from round k onward the scan always finds at least one survivor, namely L_k. The learner can therefore output arbitrarily only in the first $k-1$ rounds. Every replacement moves C to a subset. Since every candidate is infinite and S_t is finite, $C \setminus S_t$ is nonempty whenever the scan finds a survivor.

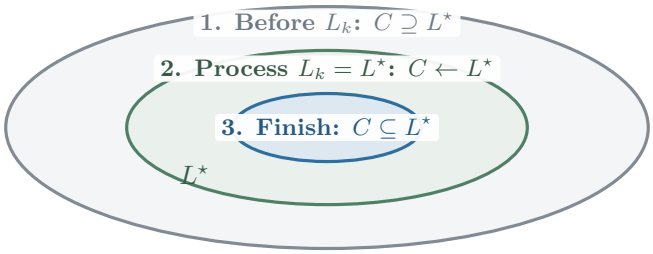

Figure 3: The nested regions show containment during a sufficiently late scan. If the scan holds an earlier survivor before L_k, it contains $L^\star$. Processing L_k makes $C = L^\star$, and every later replacement keeps C inside $L^\star$.

Proof of Theorem 3.1. Fix a target $L^\star$ and any positive text for it. Let k be the least index for which $L_k = L^\star$. We first consider the candidates that appear before the target in the list. If an earlier candidate L_j does not contain $L^\star$, then some target output is missing from L_j. That output eventually appears in the positive text, at which point L_j no longer survives. There are only finitely many indices before k. Therefore, after some time $T \geq k$, every earlier candidate that still survives must contain $L^\star$.

Now consider any round $t \geq T$. The target $L_k = L^\star$ survives. When the scan reaches it, either it is the first survivor, or the current candidate C is an earlier survivor. In the second case, the previous paragraph gives $L^\star \subseteq C$, so the scan replaces C by $L_k = L^\star$. In both cases, the current candidate equals the target after the scan processes L_k.

Every later replacement is a subset of the current candidate. The final C therefore satisfies $C \subseteq L^\star$, as illustrated in Figure 3.

Finally, C is infinite and S_t is finite, so at least one member of C has not appeared in the input. The scan chooses such a member, and its output satisfies

$$z_t \in C \setminus S_t \subseteq L^\star \setminus S_t$$

in every sufficiently late round. □

The proof does not identify $L^\star$. The final candidate C may be a strict subset of the target, and that is enough because every element of C is valid. Identification needs the exact target, whereas generation only needs a candidate contained in the target.

Finitely many mistakes versus infinitely many. Both proofs in this lecture involve waiting for examples to arrive, yet the identifier of Section 2 is wrong infinitely often while the generator above is wrong only finitely often. The difference lies in what each learner waits for and how many waits there are. In the proof above, the scan can be misled only by a candidate listed *before* $L_k = L^\star$ that fails to contain $L^\star$. There are at most $k-1$ such candidates, and each is eliminated by a single witness $w \in L^\star \setminus L_j$, a member of the target that the positive text is obliged to reveal at some finite round. The time T is the maximum of these finitely many arrival times together with k, so it is finite, although the adversary can make it as large as it likes by delaying the witnesses, and the learner never knows its value. Candidates listed after $L^\star$ are never waited for at all: once the scan reaches L_k, it moves only to subsets of $L^\star$. The identifier, by contrast, is forced to commit. During stage m of the construction in Section 2, the data are consistent with A_m, and an identifier that never proposed A_m would fail on the positive text for A_m, so it must propose A_m, after which $-(m+1)$ refutes it. This cycle has no last stage, so the refutations occur at times that grow without bound and no finite T lies beyond all of them.

Remark. The proof assumes that the learner can decide whether one candidate language is contained in another. Kleinberg and Mullainathan show that this is unnecessary: the learner only needs to ask whether a string w belongs to a listed candidate L_i, never whether it belongs to the hidden target [4].

4 Validity Does Not Guarantee Coverage

Universal generation guarantees only *safe novelty*: eventually the learner produces a valid output that has not appeared in the data. It does not say that every valid output remains within the learner's reach. The nested-tail rule demonstrates the difference. On the target $A_\infty = \mathbb{Z}$, it always moves right and can ignore every negative integer forever.

Figure 4 contrasts invalid output, safe but narrow generation, and full coverage. Because the generator in this lecture returns only one output in each round, the picture is informal: the blue region represents the intuitive set of outputs the generator might produce after seeing S_t.

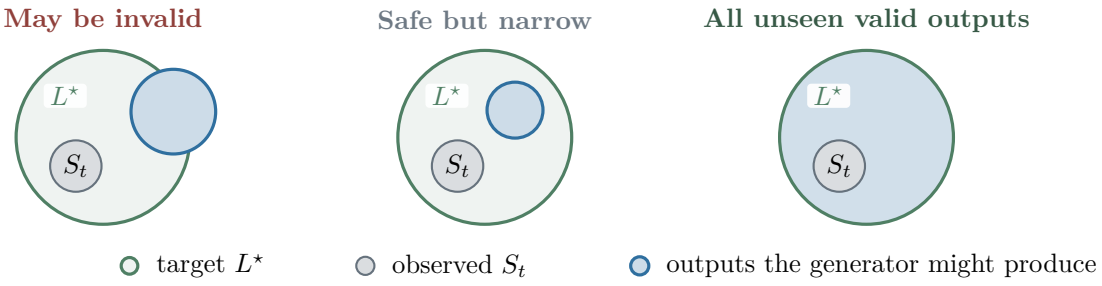

Figure 4: Validity requires the blue region to stay inside the target. Coverage additionally asks how much of the unseen target remains reachable.

Universal generation permits the middle panel: a successful learner may repeatedly choose from only a small subset of the target. Full coverage, shown in the right panel, would require every unseen valid output to remain possible. Together with S_t, those possible outputs would specify the target, tying perfect coverage to identification, which can be impossible [7].

Modern language-model systems are not limited to this positive-data game. Next-token frequencies, post-training feedback, and retrieval provide information absent from the game. Still, a system must generate beyond its observed examples without labels for every plausible unseen response. The theorem shows why some safe novelty can be possible; the middle panel shows why that guarantee alone is insufficient. The next lecture restores probabilities and studies the resulting tension between coverage and hallucination.

References

[1] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[2] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[3] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980. DOI.

[4] J. Kleinberg and S. Mullainathan. Language generation in the limit. In *NeurIPS*, 2024. Paper page.

[5] M. Charikar, A. Mehrotra, C. Pabbaraju, C. Peale, and G. Velegkas. Language generation in the limit. COLT tutorial, 2025. Tutorial website.

[6] M. Charikar and C. Pabbaraju. Exploring facets of language generation in the limit. In *COLT*, pages 854–887, 2025. Paper page.

[7] A. Kalavasis, A. Mehrotra, and G. Velegkas. On the limits of language generation: Trade-offs between hallucination and mode collapse. In *STOC*, 2025. DOI.