SPACE-EFFICIENT LANGUAGE GENERATION IN THE LIMIT

Chirag Pabbaraju

Stanford University

Joint work with



Nicholas Flammarion
EPFL



Hristo Papazov
EPFL



Miltiadis Stouras
EPFL



Ola Svensson
EPFL

LANGUAGE GENERATION IN THE LIMIT

(Kleinberg-Mullainathan'24)

- Known collection $\mathcal{C} = \{L_1, L_2, L_3, \dots\}$
- Adversary chooses some target language $L = L_z$, starts enumerating it in an order of their choosing

$$x_1, x_2, x_3, x_4, x_5, \dots\dots$$

Every $x \in L$ appears at some time, repeats allowed

- At each time step t, algorithm generates a string z_t
- Generates in the limit if beyond some large enough $t^\star$, all strings generated are new and in L_z

chooses L_{70}

	x_1	x_2	x_3	x_4	.....	$x_{t^\star}$	$x_{t^\star+1}$ .....
	z_1	z_2	z_3	z_4	.....	$z_{t^\star}$ new, $\in L_{70}$	$z_{t^\star+1}$ ..... new, $\in L_{70}$

LANGUAGE GENERATION IN THE LIMIT

(Kleinberg-Mullainathan'24)

Theorem (Kleinberg-Mullainathan '24): *Every* countable collection of languages is generatable in the limit! 🤯

The closely related task of **identifying** the target language is intractable even for extremely simple collections… 🤦

COMPUTATIONAL EFFICIENCY

- Focus has primarily been on prescribing a function that maps finite input streams to a string/guess
- Most algorithms in this space remember all the inputs seen so far at each time step
- For generation, this is essentially baked into the requirement of outputting new strings
- But in the limit, this requires unbounded space….

chooses L_{70}

😈	x_1	x_2	x_3	x_4	.....	$x_{t^\star}$	$x_{t^\star+1}$ .....
😇	z_1	z_2	z_3	z_4	.....	$z_{t^\star}$ new, $\in L_{70}$	$z_{t^\star+1}$ ..... new, $\in L_{70}$

COMPUTATIONAL EFFICIENCY

- Focus has primarily been on prescribing a function that maps finite input streams to a string/guess

Can we do language generation in the limit with bounded / efficient utilization of space?

chooses L_{70}

😈	x_1	x_2	x_3	x_4	.....	$x_{t^\star}$	$x_{t^\star+1}$ .....
😇	z_1	z_2	z_3	z_4	.....	$z_{t^\star}$ new, $\in L_{70}$	$z_{t^\star+1}$ new, $\in L_{70}$ .....

SPACE-EFFICIENT GENERATION SETUP

- The input $x_1, x_2, \ldots$ is presented in a **symbol-wise streaming** fashion
- Each x_i is input one symbol at a time; x_i's delineated by special symbol
- After reading each symbol, algorithm performs some computation **under a memory budget**
- After processing each string, algorithm outputs a **representation of the target language**

outputting a new string does not make sense with bounded memory

chooses L_{70}



0101 / 110101 / 00 / 1001 / ..... 001110 / 01100 / .....



M_1 M_2 M_3 M_4 ..... $M_{t^\star}$ $\subseteq L_{70}$ $M_{t^\star+1}$ $\subseteq L_{70}$

SPACE-EFFICIENT GENERATION SETUP

- The input $x_1, x_2, \ldots$ is presented in a **symbol-wise streaming** fashion

What collections can we even consider in this setup?

chooses L_{70}



0101 / 110101 / 00 / 1001 / ..... 001110 / 01100 / .....



M_1 M_2 M_3 M_4 ..... $M_{t^\star}$ $\subseteq L_{70}$ $M_{t^\star+1}$ $\subseteq L_{70}$

SPACE-EFFICIENT GENERATION SETUP

- With a memory budget of s bits, algorithm can only be in a bounded number of configurations $\Longrightarrow$ $\exp(s)$ many
- At the very least, algorithm should be able to test membership of a string in a language $\Longrightarrow$ collection = bounded-size regular languages
- With s bits, algorithm can only do $\exp(s)$ many membership tests
- Finite collection ($\exp(s)$) of finite-size ($\exp(s)$) regular languages…

Natural setup: Regular languages accepted by a DFA of size s under $\mathrm{poly}(s)$ space budget

SPACE-EFFICIENT LANGUAGE GENERATION IN THE LIMIT

- Collection $\mathcal{C} = \{L_1, L_2, L_3, \ldots\}$ of regular languages accepted by s-size DFA
- Adversary chooses some target language $L = L_z$, starts enumerating it in an order of their choosing

$$x_1, x_2, x_3, x_4, x_5, \ldots\ldots$$

- At each time step t, algorithm outputs a DFA M_t
- Generates in the limit if beyond some large enough $t^\star$,

$$L(M_t) \subseteq L_z \text{ and } \Delta := |L_z \setminus L(M_t)| < \infty$$

chooses L_{70}



0101 / 110101 / 00 / 1001 / ..... 001110 / 01100 / .....



M_1 M_2 M_3 M_4 ..... $M_{t^\star}$ $\subseteq L_{70}$ $M_{t^\star+1}$ $\subseteq L_{70}$

OUR RESULTS

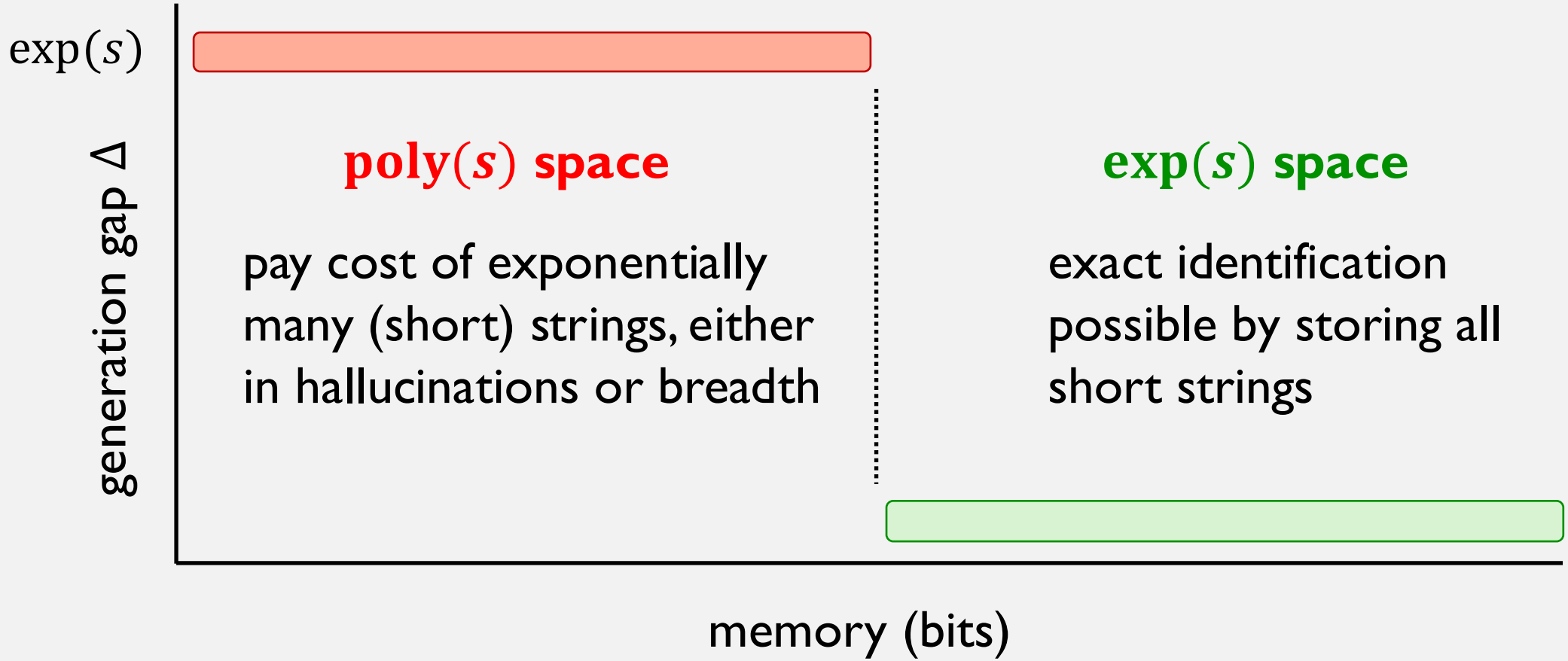
- Assumption: target L_z recognized by s-state DFA, alphabet size $\Theta(1)$
- **Theorem (Upper Bound):** There exists a $\mathrm{poly}(s)$ space algorithm that converges to a DFA M_t which satisfies $L(M_t) \subseteq L_z$ and has generation gap $\Delta \leq O(2^{2s-2})$

In particular, $L(M_t)$ captures all strings of length at least $2s-1$ in L_z

- **Corollary:** With $\exp(s)$ space, we can exactly identify L_z
- **Theorem (Lower Bound):** Any algorithm that achieves a generation gap $\Delta < 2^{(1-\varepsilon)s}$ must use $2^{\Omega(\varepsilon s)}$ bits of memory

subexponential space $\Longrightarrow$ exponential generation gap

MEMORY-HALLUCINATION-BREADTH TRADEOFF



SUMMARY

- Polynomial-space generation possible with exponential generation gap 🥳
- With exponential space, can do exact identification
- Exponential generation gap necessary with sub-exponential space 😓
- **Question:** What is the optimal attainable generation gap?
- **Question:** What are the memory tradeoffs for uniform / non-uniform / proper generation?

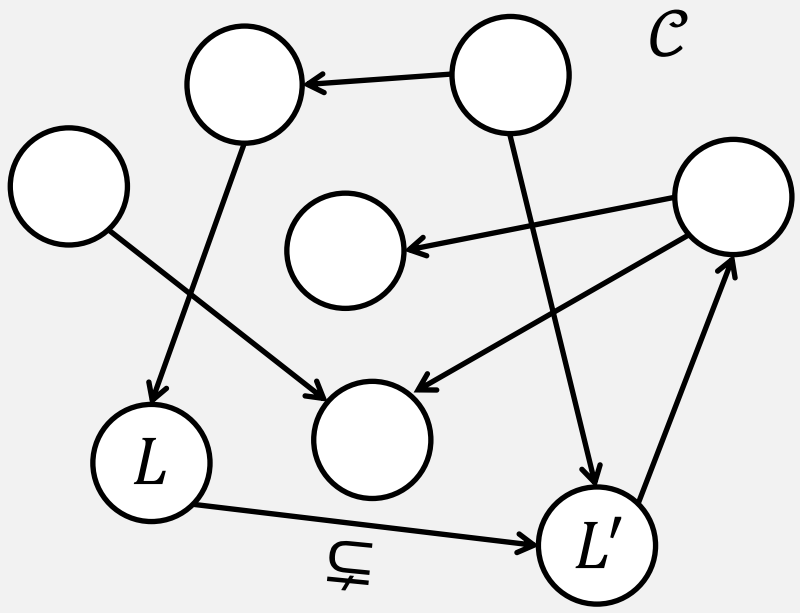
UPPER BOUND

- Angluin's algorithm for identification:

1. Topologically sort the collection according to partial order induced by strict inclusion

$$L_i \prec L_j \Leftrightarrow L_i \subsetneq L_j$$

2. At time t, output the first L_i such that $\{x_1, \ldots, x_t\} \subseteq L_i$



DFS to obtain $L_i \prec L_j \Leftrightarrow L_i \subsetneq L_j$

- **Two issues**

1. Naïve topological sort: $\Omega(|\mathcal{C}|) = \exp(\mathrm{s})$ space
2. Can't afford to remember $\{x_1, \ldots, x_t\}$

SPACE-EFFICIENT TOPOLOGICAL SORT

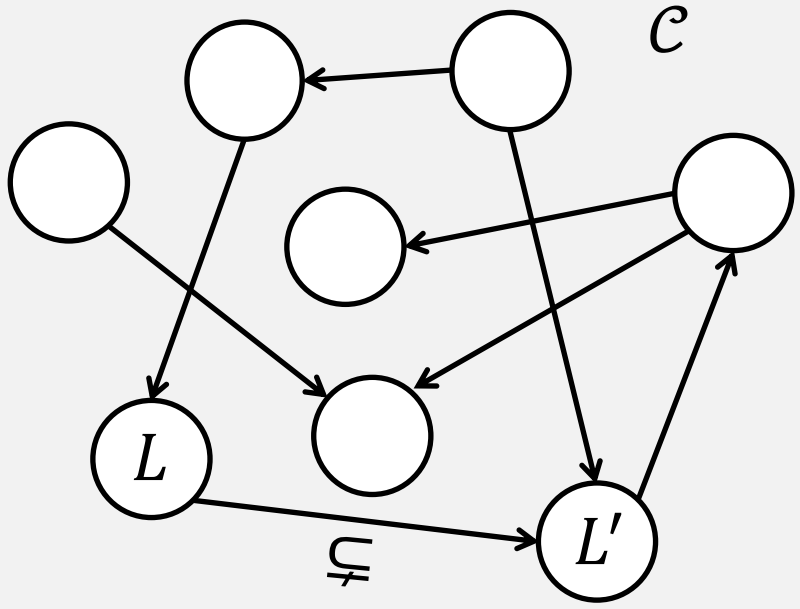
Issue #1: Naïve topological sort $= \exp(s)$ space

- "Rank Iteration Strategy"

$\text{rank}(L_i) = \text{length of largest directed path ending at } L_i$

- If $L_i \prec L_j$, then $\text{rank}(L_j) \geq \text{rank}(L_i) + 1$
- **Goal:** List down L_i's in increasing order of rank!

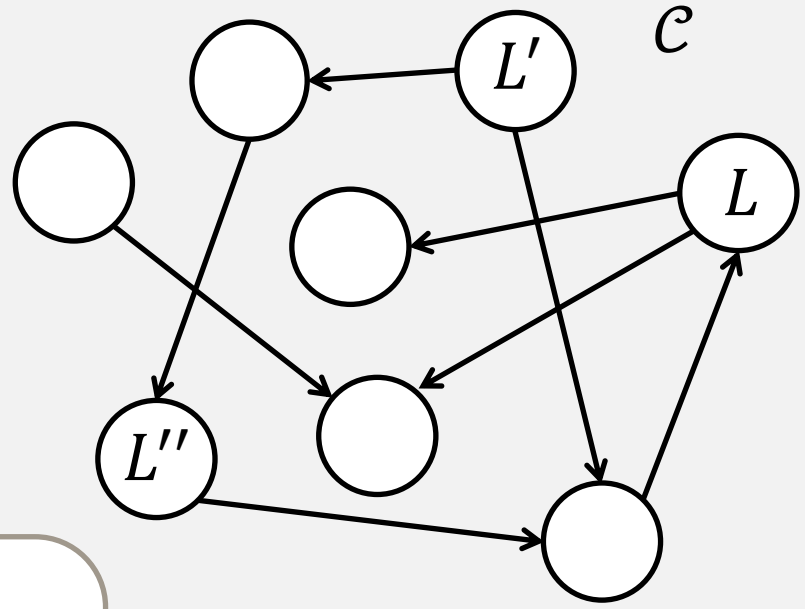
For $\ell = 0, 1, 2, \ldots, |\mathcal{C}|$:
  For language L in collection $\mathcal{C}$:
    If $\text{rank}(L) = \ell$: output L



DFS to obtain $L_i \prec L_j \Leftrightarrow L_i \subsetneq L_j$

SPACE-EFFICIENT TOPOLOGICAL SORT

- **Goal:** List down L_i's in increasing order of rank!



def rank(L):
    For $\ell = |\mathcal{C}|,\ |\mathcal{C}|-1,\ \ldots,\ 0$:
        For language L' in collection $\mathcal{C}$:
            If path(L', L, ℓ): return ℓ

def path(L', L, ℓ):
    For language L'' in collection $\mathcal{C}$:
        Check if path$(L',\ L'', \ell/2)$ and path$(L'', L, \ell/2)$
    Base case $L' \prec L$ i.e., $L' \subsetneq L$ can be checked efficiently

Savitch-style "Middle-first" search

$O(\log^2|\mathcal{C}|) = \mathrm{poly}(s)$ space

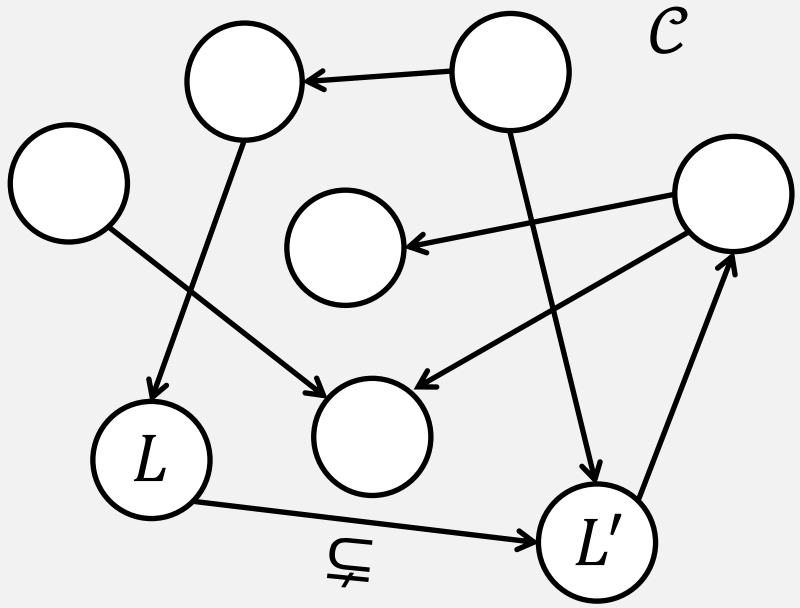
UPPER BOUND

- Angluin's algorithm for identification:

1. Topologically sort the collection according to partial order induced by strict inclusion

$$L_i \prec L_j \Leftrightarrow L_i \subsetneq L_j$$

2. At time t, output the first L_i such that $\{x_1, \ldots, x_t\} \subseteq L_i$



DFS to obtain $L_i \prec L_j \Leftrightarrow L_i \subsetneq L_j$

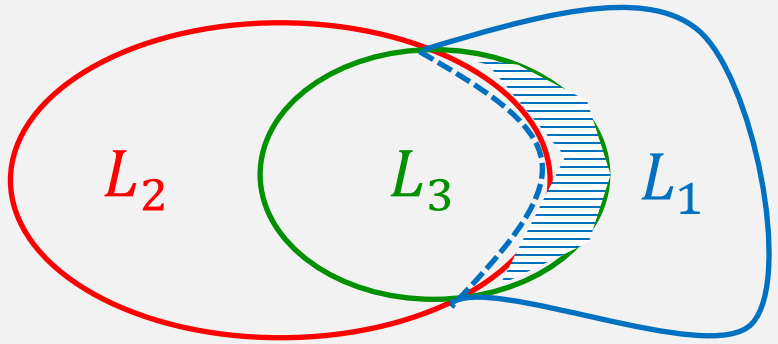
- **Two issues**

1. Naïve topological sort: $\Omega(|\mathcal{C}|) = \exp(\mathrm{s})$ space ✅
2. Can't afford to remember $\{x_1, \ldots, x_t\}$

REMEMBERING HISTORY

Issue #2: Can't afford to remember $\{x_1, \ldots, x_t\}$

- Move on from L_i to L_{i+1} in the topological sort only if **most recent string $x_t \notin L_i$**
- But this can cause stabilizing too early…
- Examples that appeared way back in the past have been forgotten 😕
- It might be the case that the language L^* we converge to satisfies $|L^* \setminus L_z| = \infty$
- But…note that $|L_z \setminus L^*| < \infty$; else will move on



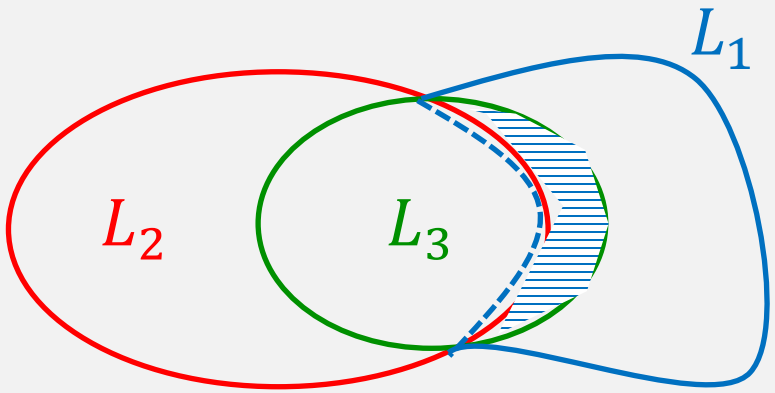
L_1 L_2 L_3
target

Adversary enumerates everything in shaded region first $\Longrightarrow$ stay at L_1

Then adversary enumerates rest of the strings in L_3 $\Longrightarrow$ stay at L_2

REMEMBERING HISTORY

If L, L^* such that $|L \setminus L^*| < \infty$ but $|L^* \setminus L| = \infty$, then the partial order $\subsetneq$ allows L^* to be placed before $L\ldots$ so if L is the target language, infinite hallucinations 😟



L_1 L_2 L_3
target

- Want a partial order that is robust to this "finite missing evidence" problem

$$L \prec L^* \Leftrightarrow |L \setminus L^*| < \infty \text{ and } |L \setminus L^*| < |L^* \setminus L|$$

- **This is partial order, and can also be checked efficiently !!** 🥳

Adversary enumerates everything in shaded region first $\Longrightarrow$ stay at L_1

Then adversary enumerates rest of the strings in $L_3 \Longrightarrow$ stay at L_2

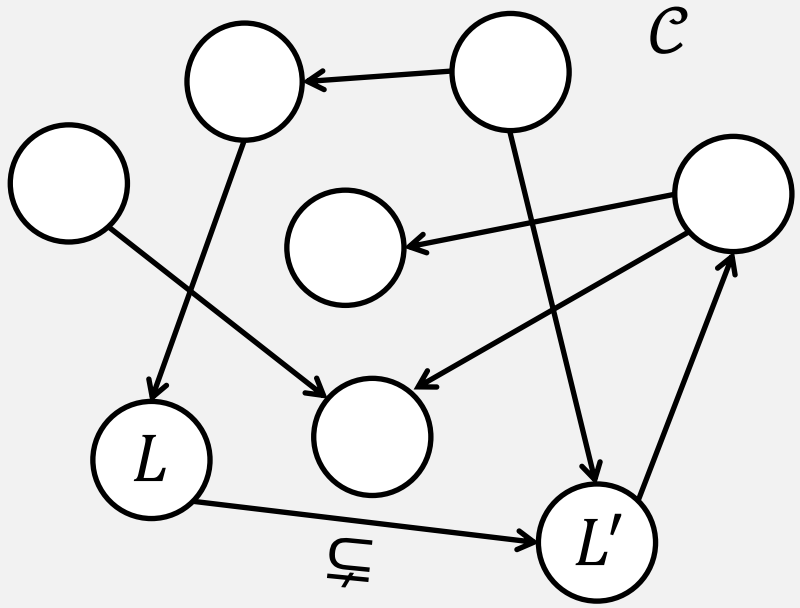
UPPER BOUND

- Angluin's algorithm for identification:

1. Topologically sort the collection according to partial order induced by strict inclusion

$$L_i \prec L_j \Leftrightarrow L_i \subsetneq L_j$$

2. At time t, output the first L_i such that $\{x_1, \ldots, x_t\} \subseteq L_i$



DFS to obtain $L_i \prec L_j \Leftrightarrow L_i \subsetneq L_j$

- **Two issues**

1. Naïve topological sort: $\Omega(|\mathcal{C}|) = \exp(\mathrm{s})$ space ✅
2. Can't afford to remember $\{x_1, \ldots, x_t\}$ ✅

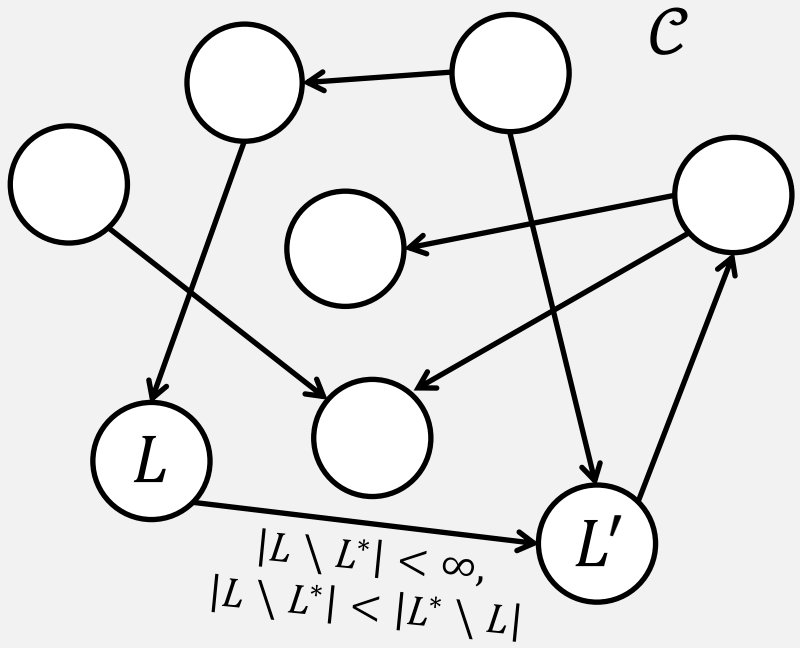
UPPER BOUND

- Our final algorithm:

1. Space-efficiently traverse the collection topologically sorted according to

$$L \prec L^* \Leftrightarrow |L \setminus L^*| < \infty \text{ and } |L \setminus L^*| < |L^* \setminus L|$$

2. At time t, step forward until $x_t \in L^*$



- Converged language L^* satisfies $|L^* \triangle L_z| < \infty$
- But $L^* \triangle L_z$ is an s^2 state DFA! Finite $\Longrightarrow$ no string of size $\geq s^2$

In fact, no string of size $\geq 2s - 1$ (Gawrychowski et al. '11)

3. Output $L^* \cap \{0,1\}^{\geq 2s-1}$ $O(s^2)$-sized DFA

- Identification: remember all strings of size $\leq 2s - 1$

LOWER BOUND

Index Problem



s bits



Alice has a string $x \in \{0,1\}^n$

Bob has an index $k \in [n]$

0	1	0	.....	0

Alice sends an s bit message

LOWER BOUND

Index Problem



s bits



Alice has a string $x \in \{0,1\}^n$

Bob has an index $k \in [n]$

0	1	0	.....	1	.....	0

k

Alice sends an s bit message

Bob's task is to output x_k

Alice must send n bits for Bob to succeed

LOWER BOUND

Universe U = Pairs of the form $(i, 0)$ and $(i, 1)$ for $i \in [n]$

j \ i	1	2	3		n
0	$(1,0)$	$(2,0)$	$(3,0)$	.....	$(n,0)$
1	$(1,1)$	$(2,1)$	$(3,1)$		$(n,1)$

LOWER BOUND

Universe U = Pairs of the form $(i, 0)$ and $(i, 1)$ for $i \in [n]$

j \ i	1	2	3		i		n
0	$(1,0)$	$(2,0)$	$(3,0)$	.....	$(i,0)$	.....	$(n,0)$
1	$(1,1)$	$(2,1)$	$(3,1)$		$(i,1)$		$(n,1)$

$L_{ij} = U \setminus \{(i,j)\}$, for $i \in [n]$, $j \in \{0,1\}$ $O(\log n)$-size DFA

Collection $\mathcal{C} = \{ L_{ij} : i \in [n],\ j \in \{0,1\} \}$

j \ i	1	2	3		k		n
0	$(1,0)$	$(2,0)$	$(3,0)$	.....	$(i,0)$	.....	$(n,0)$
1	$(1,1)$	$(2,1)$	$(3,1)$		$(i,1)$		$(n,1)$

$L_{ij} = U \setminus \{(i,j)\}$, for $i \in [n]$, $j \in \{0,1\}$



Suppose there is an algorithm that identifies using only $n-1$ bits



Alice has a string $x \in \{0,1\}^n$

Bob has an index $k \in [n]$

0	1	0	.....	1	.....	0

k

Alice feeds algorithm (i, x_i) for all i

j \ i	1	2	3		k		n
0	$(1,0)$	$(2,0)$	$(3,0)$	.....	$(i,0)$	.....	$(n,0)$
1	$(1,1)$	$(2,1)$	$(3,1)$		$(i,1)$		$(n,1)$

$L_{ij} = U \setminus \{(i,j)\}$, for $i \in [n]$, $j \in \{0,1\}$



Suppose there is an algorithm that identifies using only $n-1$ bits

$\longrightarrow$

$n-1$ bits



Alice has a string $x \in \{0,1\}^n$

Bob has an index $k \in [n]$

0	1	0	.....	1	.....	0

k

Alice feeds algorithm (i, x_i) for all i sends memory state to Bob

Bob continues feeding algorithm $(i,0)$ and $(i,1)$ for all $i \neq k$

j \ i	1	2	3		k		n
0	$(1,0)$	$(2,0)$	$(3,0)$	.....	$(i,0)$	.....	$(n,0)$
1	$(1,1)$	$(2,1)$	$(3,1)$		$(i,1)$		$(n,1)$

$L_{ij} = U \setminus \{(i,j)\}$, for $i \in [n]$, $j \in \{0,1\}$



Suppose there is an algorithm that identifies using only $n-1$ bits

$\longrightarrow$

$n-1$ bits



Alice has a string $x \in \{0,1\}^n$

Bob has an index $k \in [n]$

0	1	0	.....	1	.....	0

k

Alice feeds algorithm (i, x_i) for all i sends memory state to Bob

Bob continues feeding algorithm $(i,0)$ and $(i,1)$ for all $i \neq k$

j \ i	1	2	3		k		n
0	$(1,0)$	$(2,0)$	$(3,0)$	.....	$(i,0)$	.....	$(n,0)$
1	$(1,1)$	$(2,1)$	$(3,1)$		$(i,1)$		$(n,1)$

$L_{ij} = U \setminus \{(i,j)\}$, for $i \in [n]$, $j \in \{0,1\}$



Suppose there is an algorithm that identifies using only $n-1$ bits

$\longrightarrow$

$n-1$ bits



Algorithm has been fed $L_{k,1-x_k}$
Bob can tell x_k by algorithm's output

Alice has a string $x \in \{0,1\}^n$

Bob has an index $k \in [n]$

0	1	0	.....	1	.....	0

k

Alice feeds algorithm (i, x_i) for all i sends memory state to Bob

Bob continues feeding algorithm $(i,0)$ and $(i,1)$ for all $i \neq k$

SUMMARY

- Polynomial-space generation possible with exponential generation gap 🥳
- With exponential space, can do exact identification
- Exponential generation gap necessary with sub-exponential space 😓
- See also recent and independent work by Kleinberg, Mehrotra, Saberi, Velegkas '26 on generation with bounded memory!
- **Question:** What is the optimal attainable generation gap?
- **Question:** What are the memory tradeoffs for uniform / non-uniform / proper generation?