Factor Automata of Automata and Applications

Mehryar Mohri[1,2], Pedro Moreno[2], Eugene Weinstein[1,2]
mohri@cs.nyu.edu, pedro@google.com, eugenew@cs.nyu.edu

[1] Courant Institute of Mathematical Sciences
[2] Google Inc.

Introduction

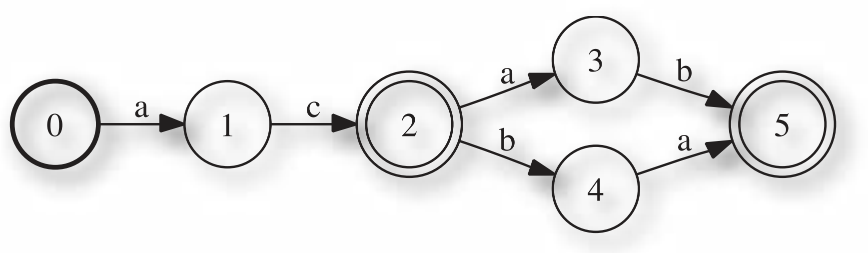
- Objective: construct full index for a large set of strings
  - We want to efficiently search for factors (subwords)
- Deterministic minimal factor automaton is a good option
  - Optimal lookup speed (linear in size of query)
- Set of strings might be given as an automaton
  - Smaller representation
  - Might be produced by another application
- Hence, consider factor automata of automata

Past Work

- Factor automaton of a string x has at most $2|x|-2$ states, and $3|x|-4$ transitions [Crochemore '85; Blumer et al. '86]
  - Can be constructed by a linear-time online algorithm
- Size bounds for a set of strings U has also previously been studied [Blumer et al. '87]
  - If $||U||$ is the sum of the lengths of all the strings in U
  - Factor automaton of U has at most $2||U||-1$ states and $3||U||-3$ transitions
  - We prove a substantially better bound here

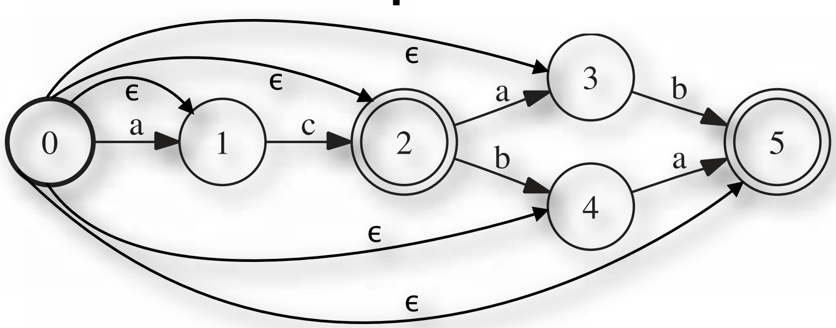
Suffix & Factor Automata

- We start out with an automaton A recognizing strings in U
- Let $S(A)$ and $F(A)$ be the deterministic minimal automata recognizing the suffixes and factors of A , respectively
- To construct $S(A)$ make each state of A initial (by adding epsilons), determinize, minimize
- To construct $F(A)$ make each state of $S(A)$ final, minimize
- Consequence: $|F(A)| \leq |S(A)|$



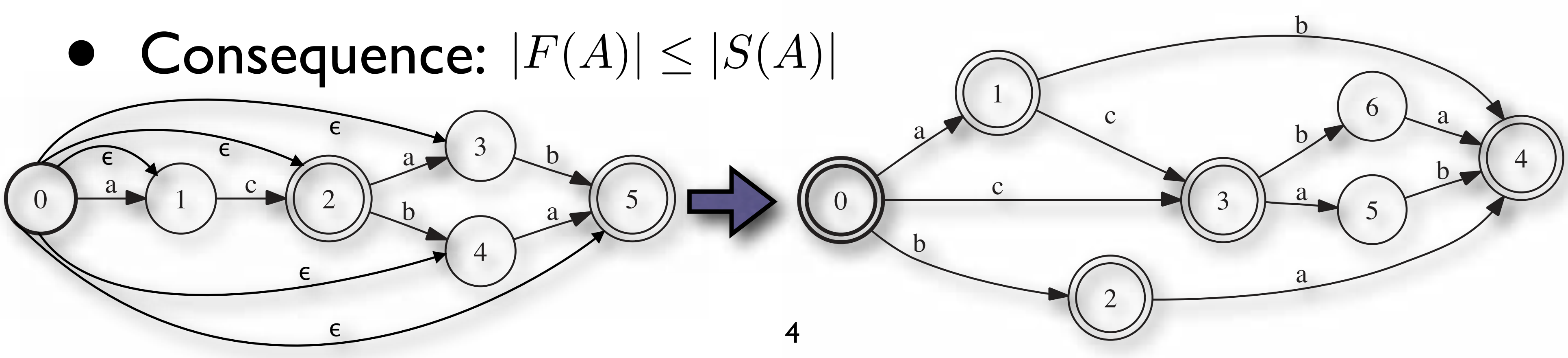
Suffix & Factor Automata

- We start out with an automaton A recognizing strings in U
- Let $S(A)$ and $F(A)$ be the deterministic minimal automata recognizing the suffixes and factors of A, respectively
- To construct $S(A)$ make each state of A initial (by adding epsilons), determinize, minimize
- To construct $F(A)$ make each state of $S(A)$ final, minimize
- Consequence: $|F(A)| \leq |S(A)|$



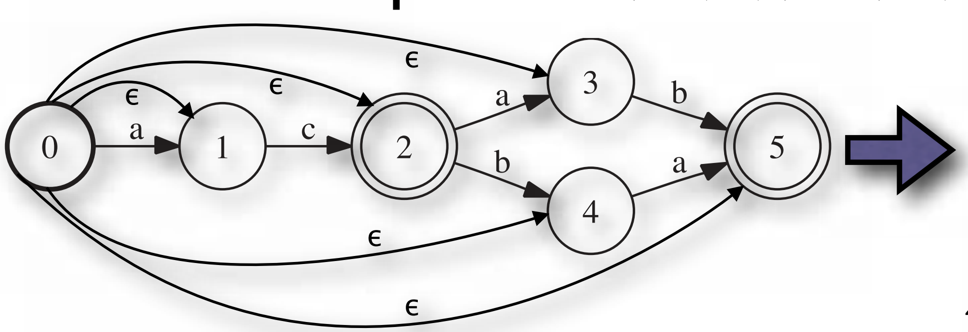
Suffix & Factor Automata

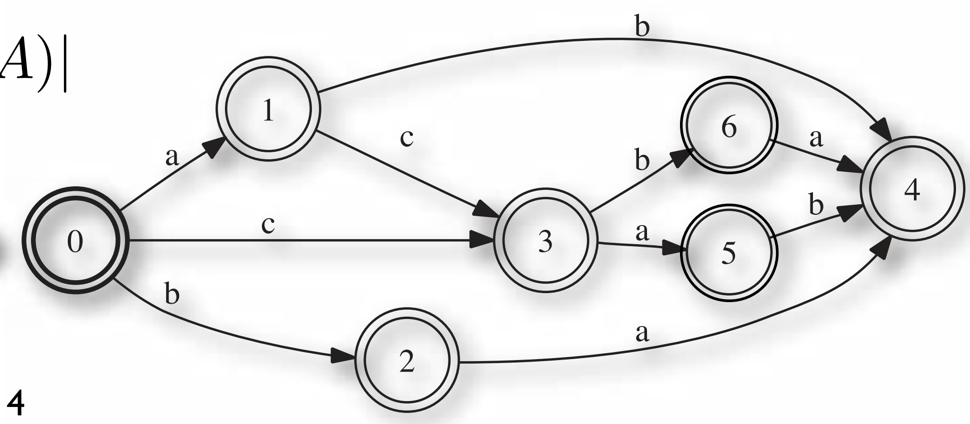
- We start out with an automaton A recognizing strings in U
- Let $S(A)$ and $F(A)$ be the deterministic minimal automata recognizing the suffixes and factors of A, respectively
- To construct $S(A)$ make each state of A initial (by adding epsilons), determinize, minimize
- To construct $F(A)$ make each state of $S(A)$ final, minimize
- Consequence: $|F(A)| \leq |S(A)|$



Suffix & Factor Automata

- We start out with an automaton A recognizing strings in U
- Let $S(A)$ and $F(A)$ be the deterministic minimal automata recognizing the suffixes and factors of A , respectively
- To construct $S(A)$ make each state of A initial (by adding epsilons), determinize, minimize
- To construct $F(A)$ make each state of $S(A)$ final, minimize
- Consequence: $|F(A)| \leq |S(A)|$



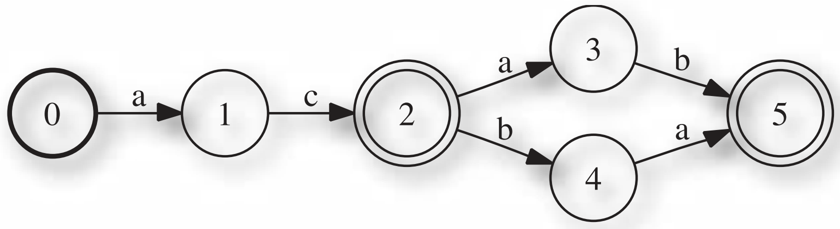


Size Bound: Strategy

- Goal: a bound on $|F(A)|$ in terms of $|A|$
- Work on bounding $|S(A)|$ – consider suffixes only for now
- Idea: each state in $S(A)$ accepts a distinct set of suffixes, so count the number of possible sets of suffixes
  - The suffix sets can be arranged in a hierarchy, which is directly related in size to A
  - Motivated by similar arguments for single-string case in [Blumer et al. '86]; string sets in [Blumer et al. '87]

Suffix Sets

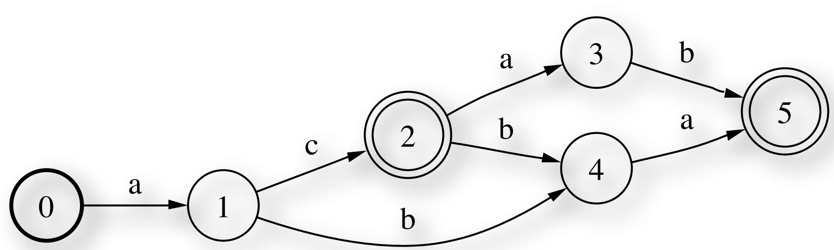
- Automaton A is k-suffix unique if no two strings accepted by A share the same k-length suffix. Suffix-unique if $k = 1$
- Define $end\text{-}set(x)$: set of states in A reachable after reading x
  - e.g., $end\text{-}set(ac) = \{2, 3, 4, 5\}$
- $x \equiv y$ denotes $end\text{-}set(x) = end\text{-}set(y)$
  - This is a right-invariant equivalence relation
  - $[x]$ is the equivalence class of x



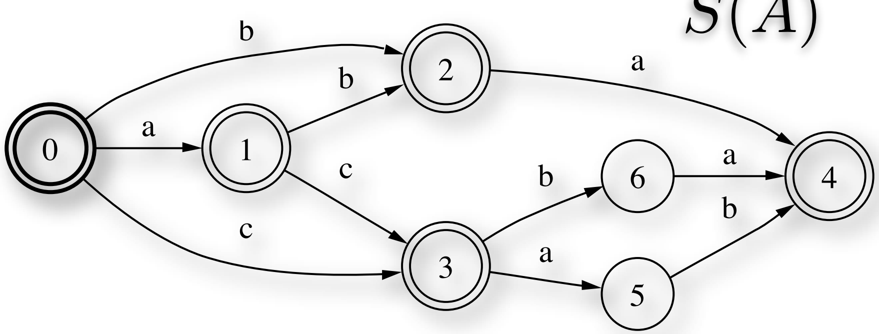
Notation

- N_{str} is number of strings accepted by A
- If q is a state of $S(A)$, $\mathrm{suff}(q)$ is set of suffixes accepted from q
  - e.g., $\mathrm{suff}(3) = \{ab, ba\}$
- $N(q)$ is the set of states in A from which a non-empty string in $\mathrm{suff}(q)$ can be read to reach a final state
  - e.g., $N(3) = \{2, 1\}$

A



$S(A)$



Suffix Set Inclusion

Suffix Set Inclusion

- Lemma: Let A be a suffix-unique automaton and let q and q' be two states of $S(A)$ such that $N(q) \cap N(q') \neq \emptyset$, then

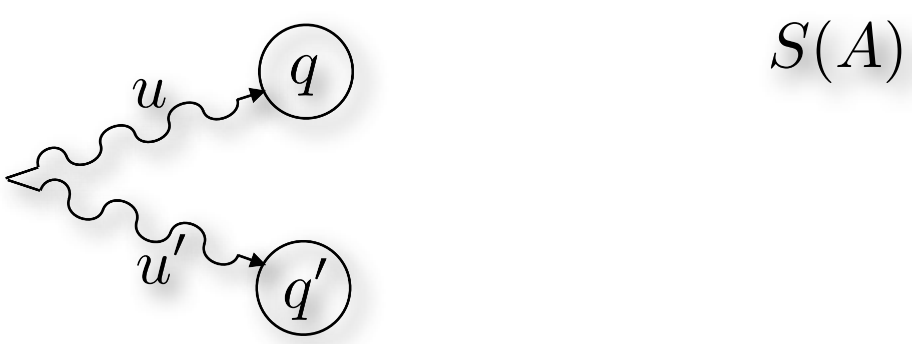
$$\mathrm{suff}(q) \subseteq \mathrm{suff}(q') \ \textit{and} \ N(q) \subseteq N(q') \quad \text{or}$$
$$\mathrm{suff}(q') \subseteq \mathrm{suff}(q) \ \textit{and} \ N(q') \subseteq N(q)$$

Suffix Set Inclusion

- Lemma: Let A be a suffix-unique automaton and let q and q' be two states of $S(A)$ such that $N(q) \cap N(q') \neq \emptyset$, then

$$\mathrm{suff}(q) \subseteq \mathrm{suff}(q') \ \textit{and} \ N(q) \subseteq N(q')$$
or
$$\mathrm{suff}(q') \subseteq \mathrm{suff}(q) \ \textit{and} \ N(q') \subseteq N(q)$$

- Proof: Let paths in $S(A)$ to q and q' be labeled with u and u'.

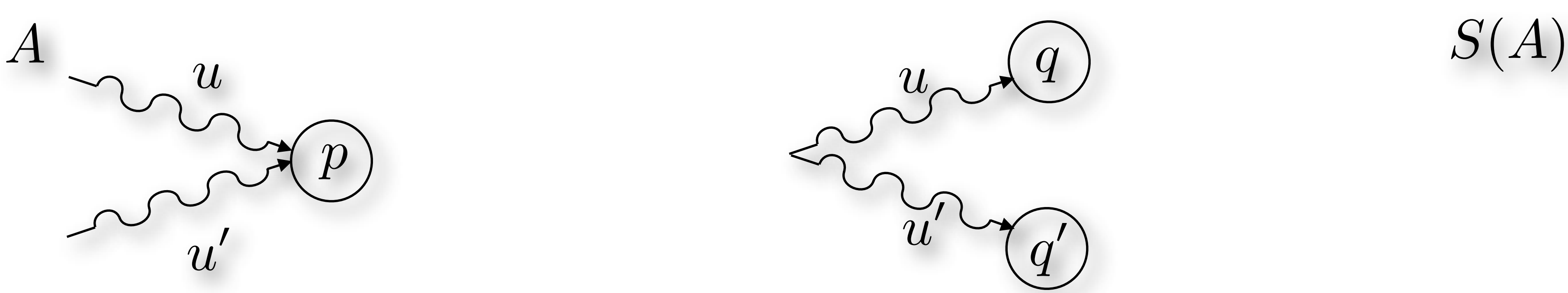


Suffix Set Inclusion

- Lemma: Let A be a suffix-unique automaton and let q and q' be two states of $S(A)$ such that $N(q) \cap N(q') \neq \emptyset$, then

$$\mathrm{suff}(q) \subseteq \mathrm{suff}(q') \;\; and \;\; N(q) \subseteq N(q') \quad \text{or}$$
$$\mathrm{suff}(q') \subseteq \mathrm{suff}(q) \;\; and \;\; N(q') \subseteq N(q)$$

- Proof: Let paths in $S(A)$ to q and q' be labeled with u and u'.
  - Thus A must have a state $p \in N(q) \cap N(q')$

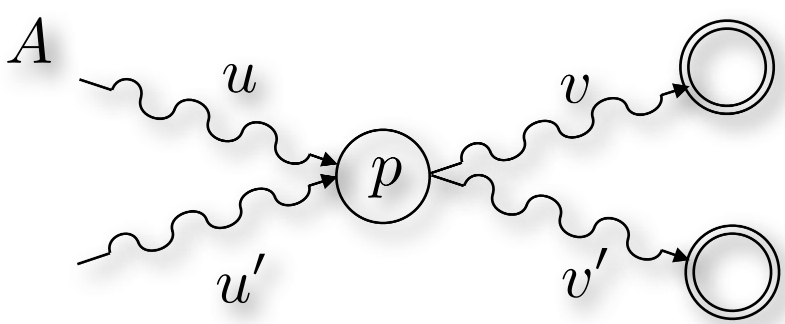


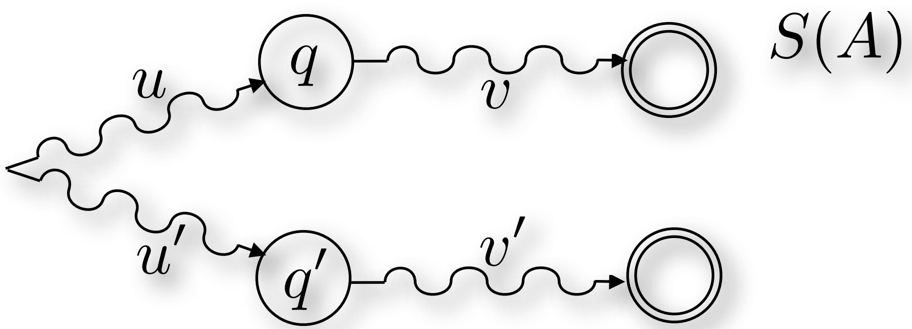
Suffix Set Inclusion

- Lemma: Let A be a suffix-unique automaton and let q and q' be two states of $S(A)$ such that $N(q) \cap N(q') \neq \emptyset$, then

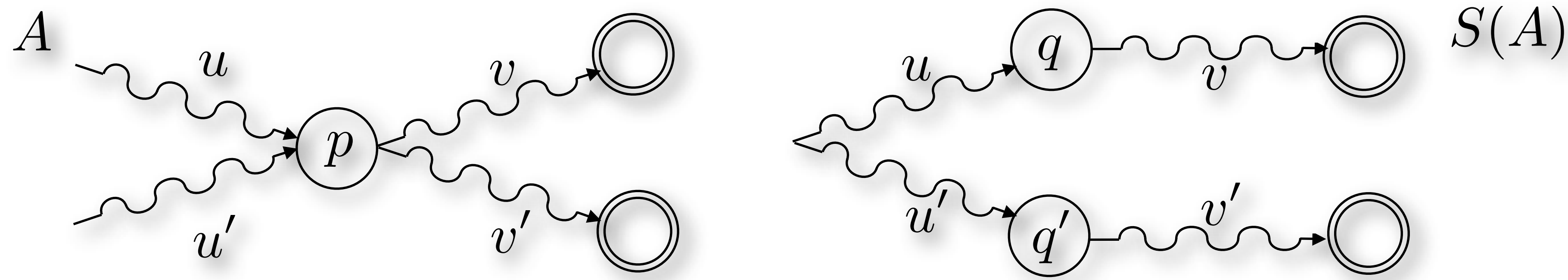
$$\mathrm{suff}(q) \subseteq \mathrm{suff}(q') \textit{ and } N(q) \subseteq N(q') \quad \text{or}$$
$$\mathrm{suff}(q') \subseteq \mathrm{suff}(q) \textit{ and } N(q') \subseteq N(q)$$

- Proof: Let paths in $S(A)$ to q and q' be labeled with u and u'.
  - Thus A must have a state $p \in N(q) \cap N(q')$
  - Thus, exist paths $v \in \mathrm{suff}(q)$ and $v' \in \mathrm{suff}(q')$ from p to final





Suffix Set Inclusion



- Since A is suffix-unique, any string accepted by A and ending in v must also end in uv
  - Thus, any path from initial to p must end in u
  - By same reasoning, it must also end in u'
  - Hence, u is a suffix of u', or vice versa
  - Assume the former, then $\mathrm{suff}(q') \subseteq \mathrm{suff}(q)$, thus $N(q') \subseteq N(q)$ QED.

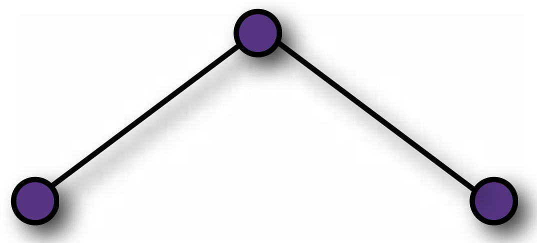
Suffix-unique Bound

- Theorem: If A is a suffix-unique deterministic and minimal automaton, then the number of states of $S(A)$ is bounded as

$$|S(A)|_Q \leq 2|A|_Q - 3$$

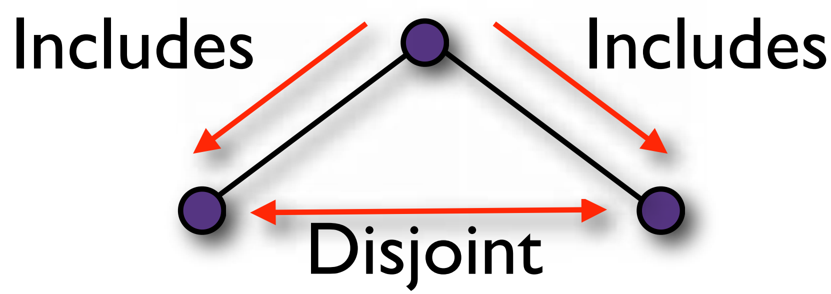
- Proof (sketch):
  - Lemma: For any two states of the suffix automaton, either suffix sets are disjoint, or one includes the other
  - We can show that each state q of $S(A)$ corresponds to a distinct equivalence class $[x]$, count these to get bound
  - The equivalence sets induce a suffix sets hierarchy which we will analyze

Suffix Sets: Non-branching



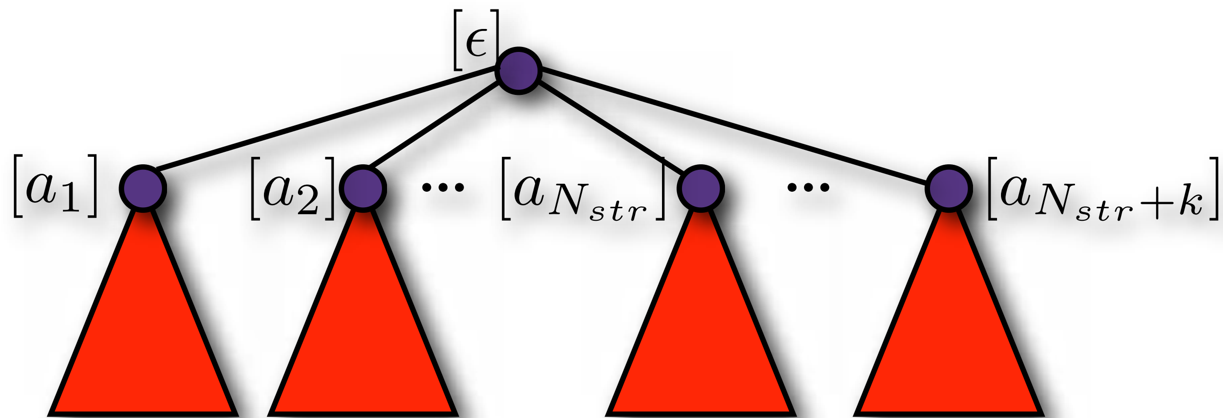
- Count non-branching, branching nodes separately
- Consider state in $S(A)$ with equivalence class $[x]$, x longest
- The only way to have a branching node is if there exist factors $ax, bx (a \neq b)$ (since $\equiv$ is a right-equivalence relation)
  - Node is only non-branching when x is a prefix or suffix
  - $|A|_Q - 2$ distinct prefixes, suffix only when final state: N_{str}
- Total non-branching nodes $N_{nb} \leq |A|_Q - 2 + N_{str}$

Suffix Sets: Non-branching



- Count non-branching, branching nodes separately
- Consider state in $S(A)$ with equivalence class $[x]$, x longest
- The only way to have a branching node is if there exist factors $ax, bx (a \neq b)$ (since $\equiv$ is a right-equivalence relation)
  - Node is only non-branching when x is a prefix or suffix
  - $|A|_Q - 2$ distinct prefixes, suffix only when final state: N_{str}
- Total non-branching nodes $N_{nb} \leq |A|_Q - 2 + N_{str}$

Suffix Sets: Branching



- If $a_1, \ldots, a_{N_{str}}$ are the distinct final symbols of each string accepted by A then each $[a_i]$ is a child of the root $[\epsilon]$
- Let tree rooted at $[a_i]$ have n_{a_i} leaves($n_{a_i} - 1$ branching nodes)
- Total number of leaves is $|A|_Q - 2$ (not initial and super-final)
- Total branching $N_b \leq \sum_{i=1}^{N_{str}+k}(n_{a_i} - 1) + 1 \leq |A|_Q - 2 - N_{str}$
- Total size of tree $N_{nb} + N_b \leq 2|A|_Q - 4$
- Add "super-final" state, get $|S(A)|_Q \leq 2|A|_Q - 3$ QED.

Final Size Result

Final Size Result

- If A is a prefix tree representing a set of strings U then
  $|S(U)|_Q \leq 2|A|_Q - 2 \quad |F(U)|_Q \leq 2|A|_Q - 2$
  $|S(U)|_E \leq 3|A|_E - 4 \quad |F(U)|_E \leq 3|A|_E - 4$

Final Size Result

- If A is a prefix tree representing a set of strings U then
  $|S(U)|_Q \leq 2|A|_Q - 2 \quad |F(U)|_Q \leq 2|A|_Q - 2$
  $|S(U)|_E \leq 3|A|_E - 4 \quad |F(U)|_E \leq 3|A|_E - 4$
- Substantial improvement over previous: $|S(U)|_Q \leq 2||U|| - 1$
  $|F(U)|_E \leq 3||U|| - 3$

Final Size Result

- If A is a prefix tree representing a set of strings U then
  $|S(U)|_Q \leq 2|A|_Q - 2 \quad |F(U)|_Q \leq 2|A|_Q - 2$
  $|S(U)|_E \leq 3|A|_E - 4 \quad |F(U)|_E \leq 3|A|_E - 4$
- Substantial improvement over previous: $|S(U)|_Q \leq 2||U|| - 1$
  $|F(U)|_E \leq 3||U|| - 3$
- When A is k-suffix unique, deterministic and minimal, and accepts n strings and A_k is the part of A after removing all suffixes of length k

$$|S(A)|_Q \leq 2|A_k|_Q + 2kn - 3 \qquad |F(A)|_Q \leq 2|A_k|_Q + 2kn - 3$$

$$|S(A)|_E \leq 2|A_k|_E + 3kn - 3k - 1 \quad |F(A)|_E \leq 2|A_k|_E + 3kn - 3k - 1$$

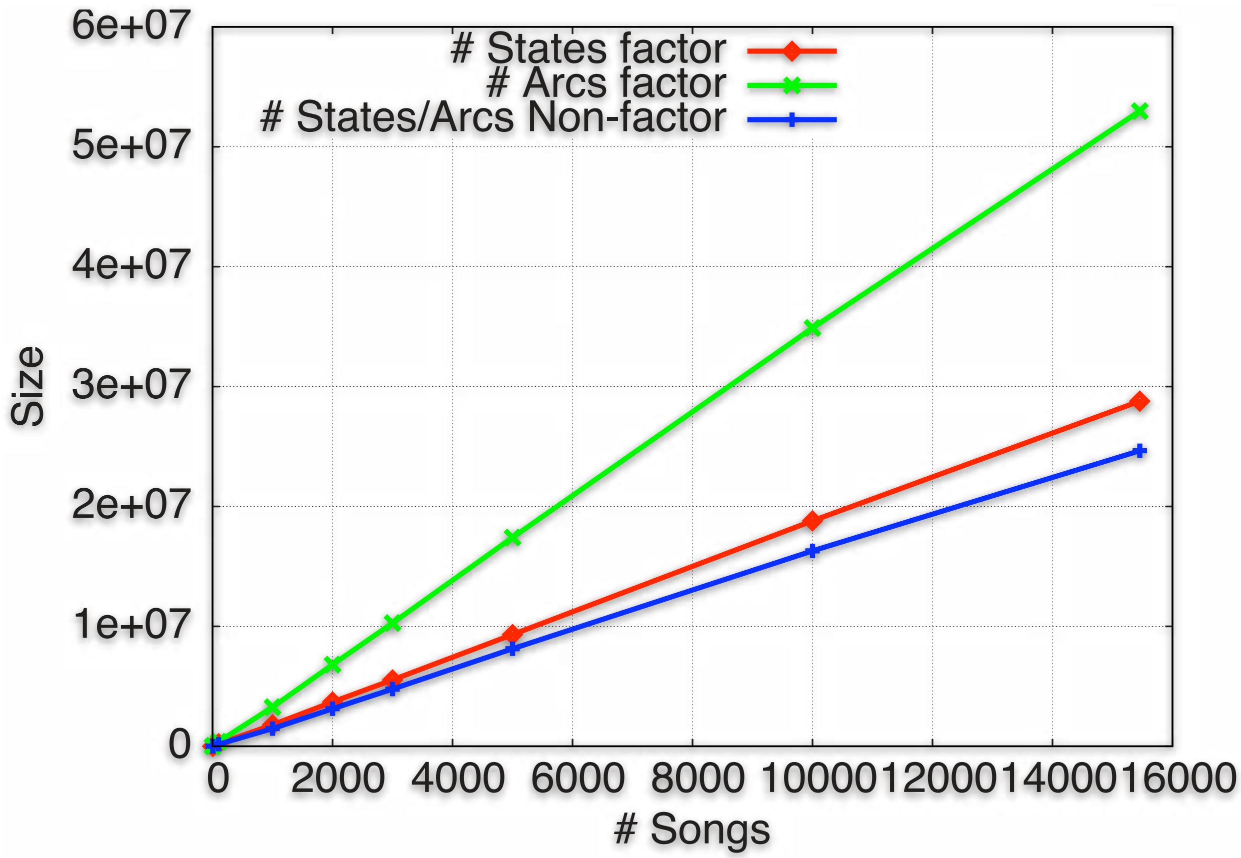
- Proof idea: add terminal symbols to make string set suffix-unique, construct suffix automaton, remove symbols

Application

- Application: large-scale music identification
  - Matching audio recording to a large song database
- Approach: learn inventory of music sounds ("phonemes")
  - A song is described by unique music phone sequence
  - Each song represented by unique string, set of music phonemes is the alphabet

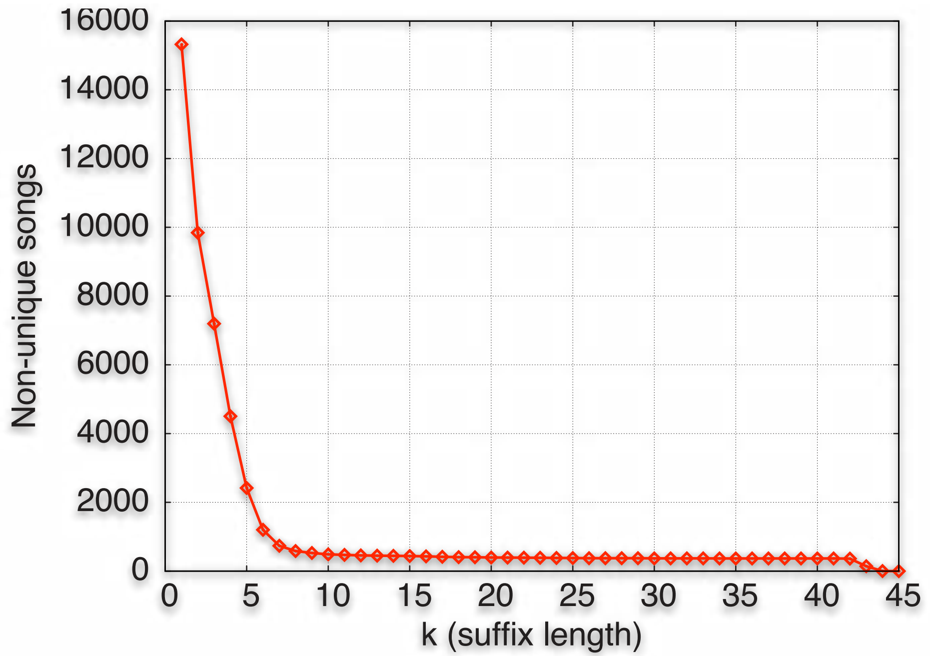
Music ID Experiments

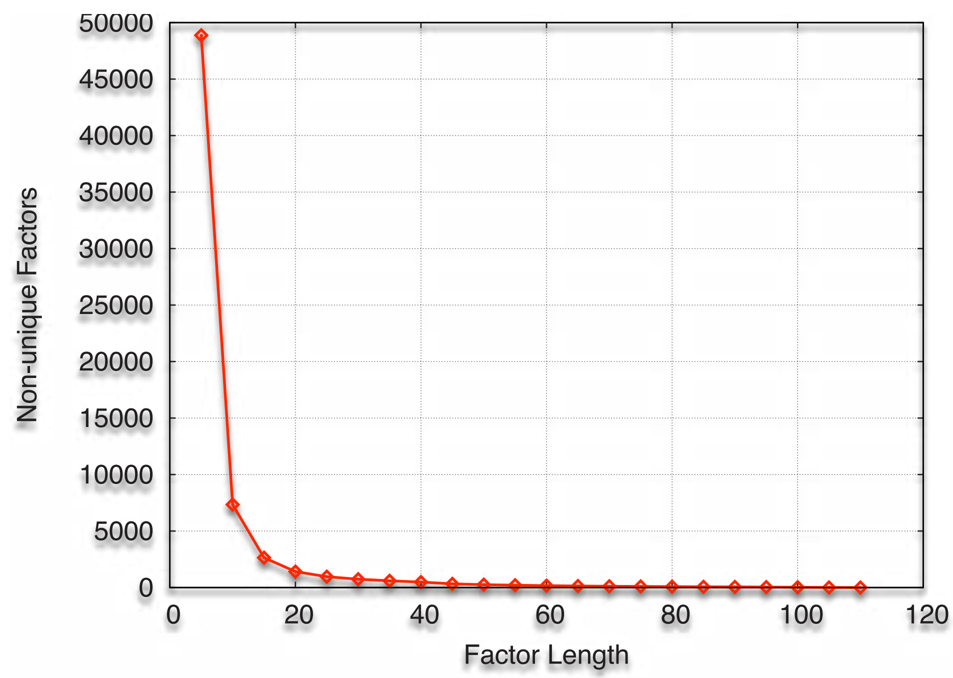
- In our music ID application, we have $|F(A)|_E \approx 2.1|A|_E$
- Factor automaton size scales linearly with # of songs



Music ID Experiments

- For 15,000+ songs, string set is 45-suffix unique
- Number of "collisions" among song suffixes/factors drops off rapidly with increasing length





Summary

- We have addressed the size of a factor automaton of a set of strings, or more generally of another automaton
  - We have proven substantially better size bounds
  - This suggests factor automata are useful for indexing potentially very large sets of strings
- Our conclusions are verified experimentally in our music identification system
- In the future, do a finer analysis
  - Tighten the kn term in the k-suffix unique bound

Factor Automata of Automata and Applications

Mehryar Mohri[1,2], Pedro Moreno[2], Eugene Weinstein[1,2]
mohri@cs.nyu.edu, pedro@google.com, eugenew@cs.nyu.edu

[1] Courant Institute of Mathematical Sciences
[2] Google Inc.