CS 360
Fall 2021
The Separating Words Problem

Jeffrey Shallit
School of Computer Science
University of Waterloo
shallit@uwaterloo.ca
https://cs.uwaterloo.ca/~shallit
In this lecture I’m going to tell you about one of the simplest and yet most perplexing problems about automata. It was introduced about 35 years ago and is still unsolved.

It is possibly the simplest problem about computation still open. (P versus NP is much harder even to state.)

This lecture is enrichment, and you won’t be tested on it. Nevertheless, I hope you find this problem is intriguing and someone will work on it and solve it! At the very end, there is a bonus.
Imagine a stupid computing device with very limited powers…

What is the simplest computational problem you could ask it to solve?
The Simplest Computational Problem?

- not the addition of two numbers
- not sorting
- it’s telling two inputs apart - distinguishing them

Thanks to Gavin Rymill for letting me use his Dalek image.
Our main computational model is the **deterministic finite automaton**, or DFA.

(You can also ask similar questions for NFA’s or other computing models.)
We want to know how many states suffice to tell one length-$n$ binary string from another.

On *average*, it’s easy. That’s because with 50% probability, just looking at the first letter is enough. With 75% probability, just looking at the first two letters is enough. With 87.5% probability, just looking at the first three letters is enough. And so forth...

— but how about in the worst case?

Motivation: a classical problem from the early days of automata theory:

Given two automata, how big a string do we need to distinguish them?
Motivation

More precisely, given two DFA’s $M_1$ and $M_2$, with $m$ and $n$ states, respectively, with $L(M_1) \neq L(M_2)$, what is a good bound on the length of the shortest string accepted by one but not the other?

◮ The direct product construction gives an upper bound of $mn - 1$ (make a DFA for the symmetric difference)
◮ But a better upper bound of $m + n - 2$ can be proved.
◮ Furthermore, this bound is best possible.
◮ For NFA’s the bound is exponential in $m$ and $n$
Our problem is the inverse problem: given two distinct strings, how big an automaton do we need to separate them?

That is, given two strings \( w \) and \( x \) of length \( \leq n \), what is the smallest number of states in any DFA that accepts one string, but not the other?

Call this number \( \text{sep}(w, x) \).
A machine $M$ separates the string $w$ from the string $x$ if $M$ accepts $w$ and rejects $x$, or vice versa.

For example, the machine below separates 0010 from 1000.

However, no 2-state DFA can separate these two strings. So $\text{sep}(1000, 0010) = 3$. 
Let
\[ S(n) := \max_{|w|=|x|\leq n, w \neq x} \text{sep}(w, x), \]
the worst case for separating two distinct strings of length \( \leq n \).

In other words, \( S(n) \) is the smallest \( b \) such that every pair of distinct strings of length \( \leq n \) can be separated by some automaton of \( \leq b \) states.

The \textit{separation problem} is to find good upper and lower bounds on the size of \( S(n) \).
The separation problem was first studied by Goralcik and Koubek 1986, who proved $S(n) = o(n)$.

In 1989 Robson obtained the upper bound $S(n) = O(n^{2/5}(\log n)^{3/5})$.

I’m not going to prove an upper bound in this lecture. Instead I’ll show that if the two strings differ in “obvious ways”, then they can be separated with few states.

Easy case #1: if the two strings are of different lengths, both \( \leq n \), we can separate them with a DFA of size \( O(\log n) \).

For by the prime number theorem, if \( k \neq m \), and \( k, m \leq n \) then there is a prime \( p = O(\log n) \) such that \( k \equiv m \pmod{p} \).

So we can accept one string and reject the other by using a cycle \( \mod{p} \), and the appropriate residue class.
Example: suppose $|w| = 22$ and $|x| = 52$. Then $|w| \equiv 1 \pmod{7}$ and $|x| \equiv 3 \pmod{7}$. So we can accept $w$ and reject $x$ with a DFA that uses a cycle of size 7, as follows:
Since it’s easy to separate strings where the lengths differ, for the remainder of the lecture, then, we only consider the case where $|w| = |x|$.

Easy case #2: we can separate $w$ from $x$ using $d + O(1)$ states if they differ in some position $d$ from the start, since we can build a DFA to accept strings with a particular prefix of length $d$. 
For example, to separate

01010011101100110000

from

01001111101011100101

we can build a DFA to recognize strings that begin with 0101:

(Transitions to a dead state omitted.)
Easy case #3: we can separate $w$ from $x$ using $d + O(1)$ states if they differ in some position $d$ from the end.

The idea is to build a pattern-recognizer for the suffix of $w$ of length $d$, ending in an accepting state if the suffix is recognized.
Separating Strings With Different Suffix

For example, to separate

1111010011001010101

from

1111011010010101101

we can build a DFA to recognize those strings that end in 0101:
Easy case #4: we can separate two strings having differing numbers of 1’s.

By the prime number theorem, if $|w|, |x| = n$, and $w$ and $x$ have $k$ and $m$ 1’s, respectively, then there is a prime $p = O(\log n)$ such that $k \not\equiv m \pmod{p}$.

So we can separate $w$ from $x$ just by counting the number of 1’s, modulo $p$. 
Using the same ideas, we can handle

Easy case #5: we can separate two length-$n$ strings $w, x$ using $O(d \log n)$ states if there is a pattern of length $d$ occurring a differing number of times in $w$ and $x$. 
The Hamming distance between $w$ and $x$ is the number of positions where they differ.

Easy case #6: if the Hamming distance between $w$ and $x$ is small, say $< d$, we can separate two length-$n$ strings using $O(d \log n)$ states.

The idea is as follows:

$$x = \begin{array}{ccccccc}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdots & \cdot \\
1 & & & & & & \\
\end{array}$$

$$y = \begin{array}{ccccccc}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdots & \cdot \\
0 & & & & & & \\
\end{array}$$

Let $i_1, i_2, \ldots, i_d$ be the positions where $x$ and $y$ differ.
Now consider $N = (i_2 - i_1)(i_3 - i_1) \cdots (i_d - i_1)$. Then $N < n^{d-1}$.

So $N$ is not divisible by some prime $p = O(\log N) = O(d \log n)$.

So $i_j \not\equiv i_1 \pmod{p}$ for $2 \leq j \leq d$.

Now count the number, modulo 2, of 1’s occurring in positions congruent to $i_1 \pmod{p}$.

These positions do not include any of $i_2, i_3, \ldots, i_d$, by the way we chose $p$, and the two strings agree in all other positions.

So $x$ contains exactly one more 1 in these positions than $w$ does, and hence we can separate the two strings using $O(d \log n)$ states.
So, as we’ve seen, if two strings differ in various “easy to state” ways, then a small automaton will separate them.

The reason why an upper bound is so hard to get is that even after we remove all the pairs that differ in an “easy to state” way, there are still lots of pairs unaccounted for.

With that in mind, let’s turn to a lower bound.
Claim: $S(n) = \Omega(\log n)$.

To see this, consider the two strings

$$0^{t-1}1^{lcm(1,2,\ldots,t)}1^{t-1}$$

and

$$0^{t-1}1^{t-1}1^{lcm(1,2,\ldots,t)}.$$

Proof in pictures:
So no $t$-state machine can distinguish these strings.

Now $\text{lcm}(1, 2, \ldots, t) = e^{t+o(t)}$ by the prime number theorem, and the lower bound $S(n) = \Omega(\log n)$ follows.
To sum up:

There is an upper bound of about $O(n^{1/3})$ on the separating words problem.

There is a lower bound $\Omega(\log n)$ on the separating words problem.

These bounds are widely separated!

I offer an automatic 100 for the course to anyone who can improve either of these bounds.