On earlier puzzles

The first puzzle from last time was the following:

Puzzle 1 Let $M = (Q, \Sigma, \delta, F)$ be a deterministic finite-state automaton (DFA), with n states. Assume that δ is total. Every $w \in \Sigma^*$ induces a function $\delta_w : Q \to Q$. Show that for each $w \in \Sigma^*$, there is a $u \in \Sigma^*$, $|u| \leq n!$, such that $\delta_w = \delta_u$.

Here is a partial solution from Meena Mahajan (meena@imsc.ernet.in).

Solution to Puzzle 1[†]

Assume that $Q = \{1, 2, \ldots, n\}$. We say that w is optimal if $\forall v \in \Sigma^*$ such that $\delta_v = \delta_w$, $|v| \geq |w|$.

Let $w = a_1 a_2 \ldots a_N$ be a string of length N in Σ^*. Let δ_i denote the function induced by the prefix w_i of w of length i. We represent each δ_i by an n-tuple of states. Clearly, δ_0 is the tuple $\langle 1, 2, \ldots, n \rangle$. Consider the sequence $\delta_0, \delta_1, \ldots, \delta_N$. If w is optimal, then each δ_i, $i \in \{1, 2, \ldots, N\}$ is distinct.

We know that the tuple δ_0 has n distinct elements. Let δ_N have k distinct elements. If $k = n$, then each δ_i is a tuple that permutes the n states. Since there are at most $n!$ such permutations, it follows from the optimality of w that $|w| \leq n!$.

If $k < n$, let us partition the sequence $\delta_0, \delta_1, \ldots, \delta_N$ into segments. For $j \in \{1, 2, \ldots, N\}$, the j-segment has that portion of the sequence where each tuple has exactly j distinct elements. The sequence begins with the n-segment and ends with the k-segment. Some segments may even be empty (e.g., if from n states we directly go to $n-4$ states, segments $n-1$, $n-2$, $n-3$ are empty).

Fix an index j such that the j-segment is non-empty. Pick out the first tuple t in this segment. Pick out a set T of j positions in $[1, 2, \ldots, n]$ such that in t_j, these positions have distinct states. From this segment onwards, it is sufficient to consider the behaviour of w on these positions alone. These positions have distinct states until this segment ends.

The number of arrangements of distinct states in these positions is bounded by $P(n, j)$, the number of permutations of j objects drawn from a set of n distinct objects. Thus,

$$\begin{aligned}
N &= \textstyle\sum_{j=k}^{n} (\text{length of j-segment}) \\
&\leq \textstyle\sum_{j=1}^{n} (\text{max length of j-segment}) \\
&= \textstyle\sum_{j=1}^{n} P(n, j) \\
&= (n!) \textstyle\sum_{j=1}^{n} \frac{1}{(n-j)!} \\
&= (n!) \textstyle\sum_{l=0}^{n-1} \frac{1}{l!} \\
&\leq (n!) \textstyle\sum_{l=0}^{\infty} \frac{1}{l!} \\
&= (n!)e
\end{aligned}$$

We now improve this crude bound. The improvement bounds $N = |w|$ by $n!$ if $k \leq n-3$, by $5(n!)/4$ if $k = n-2$, and by $3(n!)/2$ if $k = n-1$.

Partition the set of states into k parts, $Q = Q_1 \cup Q_2 \cup \ldots \cup Q_k$, based on what they are mapped to by δ_N.

Now, in the j-segment where we have identified a set T, $|T| = j$, of relevant positions, consider the partition $T = T_1 \cup T_2 \cup \ldots \cup T_k$, where $T_i = T \cap Q_i$. Clearly, all positions in T_i eventually get mapped to the same state by some prefix of w. Therefore,

Claim 1: If tuples t', t'' occur in the j-segment of w and they differ on T only by permutations within each of the parts T_i, then w is not optimal.

Proof: Assume otherwise; an optimal w has both t' and t'' in its j-segment. Say t' precedes t''. Delete the substring of w occurring between t' and t''. A modified version of the sequence of tuples after t'' will now occur immediately after t'. The modification involves permuting positions within parts of each Q_i. Since the last segment treats all positions within a part identically, these permutations have no effect on the last segment. So the final tuple is the same, and it has been realised by a string

[†] This solution was arrived at by Anil Seth (seth@imsc.ernet.in) in discussion with Meena Mahajan. The presentation of the solution was finalised by Meena Mahajan following further discussions with K. Narayan Kumar (kumar@smi.ernet.in). Shashank Mehta (skm@cs.unipune.ernet.in) had independently arrived at essentially the same solution; some of his notation has been used as well in this presentation.

shorter than w, contradicting the optimality of w. $\square$

From the above claim, it follows that the maximum possible length of the j-segment is actually much less than $P(n,j)$. It is in fact bounded by

$$\frac{P(n,j)}{|T_1|!|T_2|!\dots|T_k|!}$$

.

Claim 2: Given natural numbers $a_1, a_2, \dots a_m$ adding up to p,

$$a_1!a_2!\dots a_m! \geq 2^{p-m}$$

Proof: The expression $a_1!a_2!\dots a_m!$ has p factors, since $(a_i)!$ contributes a_i factors and $a_1 + a_2 + \dots + a_m = p$. Of these, m factors are 1, one from each $(a_i)!$. This leaves $p-m$ factors, each an integer greater than or equal to two. The inequality follows. $\square$

Claim 3: The length of the j-segment is at most $\frac{1}{2^{j-k}}\frac{n!}{(n-j)!}$.

Proof: Follows from the above two Claims. $\square$

So now,

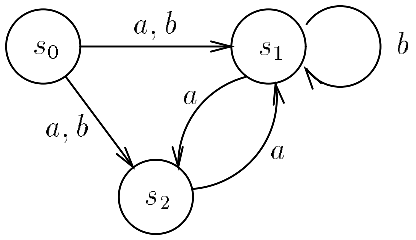
$$\begin{aligned} N &= \textstyle\sum_{j=k}^{n} (\text{length of } j\text{-segment}) \\ &\leq \textstyle\sum_{j=k}^{n} \frac{1}{2^{j-k}}\frac{n!}{(n-j)!} \\ &= (n!)\textstyle\sum_{l=0}^{n-k} \frac{1}{l!}\frac{1}{2^{(n-k)-l}} \end{aligned}$$

Now it can be easily verified that for $k \leq n-3$, $N \leq n!$.

The second puzzle from last time was the following:

Puzzle 2 Are there Nondeterministic Finite Automata (NFA) with n states that accept all strings of length less than or equal to n but not all of Σ^*?

A.A. Diwan (aad@cse.iitb.ernet.in) supplies the counter example on the right. All states in this NFA are final states. The shortest string which this NFA does not accept is *abab*. By cascading this NFA, it is possible to show a general superlinear lower bound.



However, the situation is actually much worse, as K. Narayan Kumar (kumar@smi.ernet.in) reports.

Solution to Puzzle 2[‡]

It turns out that there is a constant c such that there are NFAs of size cn that accept all strings of length less than or equal to 2^n but not all of Σ^*. Any DFA of n states that accepts all strings of length $\leq n$ must accept Σ^*. The determinisation of an NFA produces a DFA of size at most 2^n and so any NFA of size n accepting all strings of length less than or equal to 2^n must necessarily accept Σ^*. Hence, the situation for NFAs is almost as bad as it can get.

We first describe the result over alphabets of size 2^n and then show how the proof can be modified to work over constant size alphabets.

Consider $\Sigma = \{0,1\}^n$. Let b_i denote the n-bit binary representation of the number i. Then $\Sigma = \{b_0, b_1, \dots, b_{2^n-1}\}$. Consider a finite automaton that rejects an input if and only if it is the 2^n length sequence $b_0b_1\dots b_{2^n-1}$.

The automaton operates as follows: It checks whether the first letter is b_0 and accepts otherwise. It then nondeterministically checks whether the i+1st letter of the input is obtained from the ith letter by adding one. If not, it accepts. If the last letter is not b_{2^n-1} it accepts. It does not accept otherwise. The only string that this automaton rejects is $b_0b_1\dots b_{2^n-1}$.

[‡]Eric Allender (allender@cs.rutgers.edu) pointed out that automata that work over computation sequences of space-bounded Turing Machines suffice to solve this problem. The simplification (?) described here was arrived at during discussions with Meena Mahajan (meena@imsc.ernet.in).

Unfortunately this automaton has size 2^n: to verify that the $i+1$st letter is one more than the ith one, it has to remember the ith letter and the alphabet is of size 2^n. Hence there is a need for an efficient technique for detecting whether the $i+1$st letter should follow ith letter (in the fixed sequence we have in mind).

We solve this problem by "expanding" our sequence and using nondeterminism to compare just small parts of these strings (constant sized substrings) in order to verify that the $i+1$st letter is allowed to follow the ith.

We now consider sequences over $\Sigma = (\{0,1\} \times \{0,1\})^n$. A typical element looks like

$$\begin{pmatrix} 0100000... \\ 1010101... \end{pmatrix}$$

The idea is to expand the original sequence with the details of how to add 1 to the bit vector b_i to obtain b_{i+1}. The bottom row holds the number and the top row the information about carry bits. For instance, for $n = 3$, in the sequence $b_0 b_1 \ldots b_{2^n-1}$ the letter $b_4 = 100$ would immediately follow the string $b_3 = 011$. Over the new alphabet the sequence $b_3 b_4$ would be replaced by following string:

$$\begin{pmatrix} 000 \\ 011 \end{pmatrix}\begin{pmatrix} 001 \\ 011 \end{pmatrix}\begin{pmatrix} 010 \\ 010 \end{pmatrix}\begin{pmatrix} 100 \\ 000 \end{pmatrix}\begin{pmatrix} 000 \\ 100 \end{pmatrix}$$

We refer to the letter which contains all 0s in the top row and b_i in the bottom row as d_i. The letters that appear between d_i and d_{i+1} describe the process of incrementing b_i to b_{i+1} using carries. We want our new automaton to reject the string $d_0 \ldots d_1 \ldots\ldots d_{2^n-1}$ corresponding to the original string $b_0 b_1 \ldots b_{2^n-1}$.

Our new finite automaton works as follows: It verifies that the first letter is indeed d_0. It then skips some letters (going to the accepting state whenever a letter with more than one 1 in the top row is encountered). While skipping a letter it also nondeterministically chooses a $j \in \{1, \ldots, n\}$ and remembers the block $\begin{pmatrix} ab \\ cd \end{pmatrix}$ located at the jth position.

$$\begin{array}{c} 1234 \ldots j \ldots\ n \\ \begin{pmatrix} 0100 \ldots ab \ldots 0 \\ 1101 \ldots cd \ldots 1 \end{pmatrix} \end{array}$$

(At any point the position j and the corresponding entries a, b, c and d in the letter last read are stored in the state.)

The automaton then nondeterministically goes to a state where it verifies the correctness of the jth element of the next letter—the correct value can be determined from the values of a, b, c and d of the previous letter. If the value is incorrect then it accepts. Finally, it also accepts if the last letter is not d_{2^n-1}.

The only sequence which this automaton does not accept is the one which starts with d_0 and incrementally leads up to d_{2^n-1}.

The part of the automaton that checks that the first letter is d_0 has size 1. The second part that skips over its input while remembering the index j as well as the block $\begin{pmatrix} ab \\ cd \end{pmatrix}$ at position j has size $O(n)$. The last part that verifies the addition from one step to the next has constant size. In all the automaton has size $O(n)$ and the length of the only string it rejects is more than 2^n.

How can we modify the construction to work over a constant sized alphabet? Simple: just use the fact that each letter in Σ can be treated as a word of length n over the alphabet $\{0,1\} \times \{0,1\}$. Take $\Sigma' = \{\{0,1\} \times \{0,1\}\} \cup \{\#\}$. Any string over this alphabet can be considered as a sequence of strings over $\{\{0,1\} \times \{0,1\}\}$ separated by #s. The new automaton works over Σ' and rejects exactly the string that represents the sequence of strings $d_0 \ldots d_1 \ldots\ldots d_{2^n-1}$.

Like the earlier automaton, this has an inital part that verifies that the first string (the string up to the first #) is indeed d_0. This part is of size $O(cn)$. The second part skips nondeterministically through a number of strings, accepting whenever more than one 1 is encountered in the first component between any pair of #'s. It also guesses and remembers the position j together with the block $\begin{pmatrix} ab \\ cd \end{pmatrix}$ at position j in the last string it encountered. At some point it nonderministically moves in to a third component that verifies that the jth letter of the next string can follow the ones in its memory. If not it accepts. The automaton also accepts if the last string is not d_{2^n-1}. It is not difficult to check that the second and third components have size $O(n)$ so that the entire automaton is of size cn for some c. Further the construction is uniform in that for each n the constant c does not change.

Thus, there are automata of size cn for each n which accept all of Σ^* except for one string of length $> 2^n$, where Σ is of size 5.