Daily Problem: 25–Jan–2026

Problem Statement

For each integer $n \geq 1$, construct regular languages A_n and B_n over the alphabet $\Sigma = \{a, b\}$ such that:

- There exist DFAs for A_n and B_n of size $O(n)$, but
- Any DFA recognizing the concatenation $A_n \cdot B_n = \{xy : x \in A_n, y \in B_n\}$ must have at least 2^n states.

Construction of the Languages

Fix $n \geq 1$. Define:

$$A_n = \{\, w \in \{a, b\}^* \mid \text{the } n\text{-th symbol from the right of } w \text{ is } a \,\},$$

and

$$B_n = \{\, w \in \{a, b\}^n \mid w[\text{last symbol}] = a \,\}.$$

In other words:

- A_n consists of all strings whose n-th from last position is a,
- B_n consists of all strings of *exactly length* n ending in a.

Part 1: DFA Size for A_n and B_n

DFA for A_n

A DFA for A_n must remember the last n symbols to determine whether the n-th from last was a. This can be done with a $(n + 1)$-state DFA using a "sliding window" strategy:

- States $q_0, q_1, \dots, q_n$, where q_k means “I have read k symbols so far (if $k < n$) or I am tracking the last n symbols (if $k = n$)”.
- After reading n symbols, the automaton always stays in q_n while tracking the last n symbols and accepting appropriately.

Thus $|DFA(A_n)| = O(n)$.

DFA for B_n

B_n consists of strings of *exact length* n ending in a. A DFA for B_n simply counts input length up to n and checks the last letter:

- The DFA has $n + 2$ states: one for each length $0, 1, \dots, n$, plus a sink for lengths $> n$.
- Final states are exactly those representing length n with last symbol a.

Hence $|DFA(B_n)| = O(n)$.

Part 2: Lower Bound for the Concatenation $A_n \cdot B_n$

We now show that any DFA for $A_n \cdot B_n$ must have at least 2^n states.

Approach

A string x should be in $A_n \cdot B_n$ iff it can be broken as $x = uv$ where:

$$u \in A_n \quad \text{and} \quad v \in B_n.$$

Since B_n consists of strings of length exactly n, we have:

$$x \in A_n \cdot B_n \quad \iff \quad |x| \geq n \text{ and the (} n\text{-th symbol from the right in } x \text{ is } a.$$

But this is exactly the same condition that defined A_n itself.

Thus:

$$A_n \cdot B_n = A_n.$$

So the problem reduces to showing that A_n requires 2^n states when recognized by any DFA that scans left-to-right and determines acceptance at the end.

Exponential Lower Bound

We use a standard distinguishability argument.

Consider all binary strings (over $\{a, b\}$) of length n. There are 2^n such strings. Let:

$$S = \{u \in \{a, b\}^n\}.$$

Claim. For any $u, v \in S$ with $u \neq v$, a DFA for $A_n \cdot B_n$ must reach different states after reading u and v.

Proof. Since $u \neq v$, they differ in at least one position. In particular, they differ in their n-th-from-last position (which for strings of length n is the first symbol).

- Suppose the first symbol of u is a and that of v is b.
- Take $w \in B_n$, for example $w = b^{n-1}a$.

Then:

$$uw \in A_n \cdot B_n \quad \text{but} \quad vw \notin A_n \cdot B_n.$$

Thus u and v must lead to different states, or the DFA would accept one and reject the other from the same state, a contradiction.

Since this holds for all distinct $u, v \in S$, the DFA must have at least 2^n distinct states.

Every DFA recognizing $A_n \cdot B_n$ must have at least 2^n states.

Conclusion

We have constructed regular languages A_n and B_n such that:

- $|DFA(A_n)| = O(n)$,
- $|DFA(B_n)| = O(n)$,
- $|DFA(A_n \cdot B_n)| \geq 2^n$.

Therefore, concatenation of regular languages may cause an exponential blow-up in DFA size.