# Problem statement

Design an automaton that validates usernames for a messaging app under the following constraints:

1. The username must be exactly **5 characters** long.
2. It must start with a **capital letter** (A–Z).
3. It can contain only letters (A–Z, a–z) and digits (0–9).
4. It must contain **at least one digit** among the 5 characters.

(Problem statement taken from the provided assignment PDF.) :contentReference[oaicite:1]index=1

# 1 High-level approach

We model the username as a fixed-length string of five symbols. Because we must ensure at least one digit appears in the string, the automaton needs to track both the *position* (how many characters consumed) and whether a *digit has been seen* so far.

We will:

- First present an NFA-style state diagram that explicitly marks states indicating the number of characters consumed and whether a digit has been seen.
- Give the formal NFA tuple.
- Convert the NFA to a DFA using the subset (powerset) construction (the conversion is straightforward because the NFA has no $\varepsilon$-moves and transitions are effectively deterministic once the letter/digit choice is forced).
- Present the equivalent DFA diagram and the transition table.
- Answer the additional deliverable discussion points.

# 2 Design of the NFA

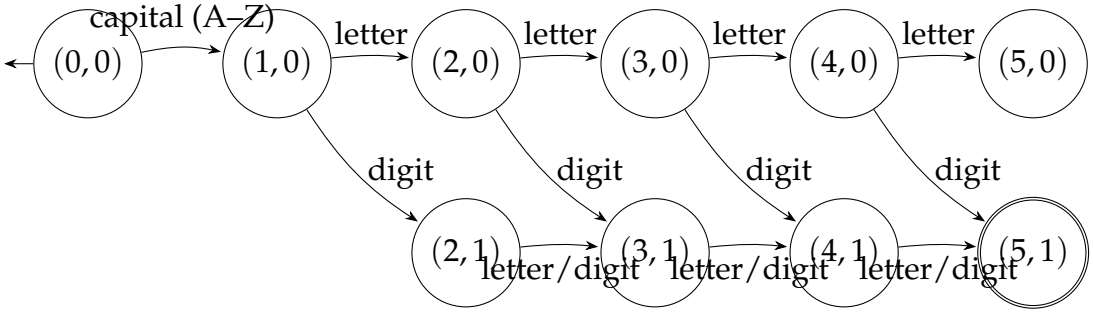
We use states labeled $(i, d)$ where $i \in \{0, 1, 2, 3, 4, 5\}$ is the number of characters consumed so far, and $d \in \{0, 1\}$ indicates whether a digit has been seen (0 = no digit yet, 1 = digit seen). The start state is $(0, 0)$. After consuming exactly 5 characters we are in some $(5, d)$ state; acceptance is allowed only if $d = 1$ (i.e. at least one digit has been seen).

**Alphabet:** $\Sigma = \{\text{letters } (A\text{-}Z, a\text{-}z),\ \text{digits } (0\text{-}9)\}$.

**Informal transitions:**

- From $(0,0)$ on a **capital letter** (A–Z) go to $(1,0)$. All other inputs from $(0,0)$ are rejected (forbidden): digits are not allowed as first character and lowercase letters for first char are not allowed because first character must be capital.
- For positions $1 \leq i \leq 4$: from $(i,0)$
  - on a letter (A–Z or a–z) go to $(i+1,0)$ (digit not yet seen),
  - on a digit (0–9) go to $(i+1,1)$ (digit now seen).
- From any $(i,1)$ for $1 \leq i \leq 4$:
  - on a letter go to $(i+1,1)$,
  - on a digit go to $(i+1,1)$ (once digit seen, remain in digit-seen mode).
- Only $(5,1)$ is an accepting state (exactly 5 characters consumed **and** at least one digit occurred).

### TikZ diagram: NFA (position + digit flag)




**Remarks:** The start state is $(0,0)$; the unique accepting state is $(5,1)$ (double circle). State $(5,0)$ is non-accepting and corresponds to strings of length 5 with *no digit*.

## 3 Formal NFA definition

We can write the NFA as a 5-tuple $N = (Q, \Sigma, \delta, q_0, F)$ where:

$Q = \{(i,d) \mid i \in \{0,1,2,3,4,5\},\ d \in \{0,1\}\}$,
$\Sigma = \{\text{letters } L,\ \text{digits } D\}$ (where $L$ stands for any A–Z or a–z, and $D$ stands for 0–9),
$q_0 = (0,0)$,
$F = \{(5,1)\}$.

Transition function $\delta$ (described at high level):

$$\begin{aligned}
&\delta((0,0),\text{capital}) = \{(1,0)\}, && \text{(first char must be capital)}\\
&\delta((i,0),L) = \{(i+1,0)\}, && \text{for } 1 \le i \le 4,\\
&\delta((i,0),D) = \{(i+1,1)\}, && \text{for } 1 \le i \le 4,\\
&\delta((i,1),L) = \{(i+1,1)\}, && \text{for } 1 \le i \le 4,\\
&\delta((i,1),D) = \{(i+1,1)\}, && \text{for } 1 \le i \le 4,\\
&\text{undefined otherwise (reject).}
\end{aligned}$$

# 4 Conversion to DFA

Because the above NFA uses deterministic transitions for each relevant symbol class (capital vs other letters vs digits) and no $\varepsilon$-moves, the subset construction will produce a DFA whose states correspond to subsets of $Q$. However in practice each reachable subset contains at most one of the $(i,d)$ states for a given $i$: the machine is effectively deterministic once we distinguish capital letters at the first position.

Thus the DFA is essentially identical in structure to the NFA described above (but viewed as a DFA). For clarity, below we list the DFA states and the transition table using symbol classes:

Symbol classes: $C = \{$Capital (A–Z), $L =$ lower or upper letter (A–Z,a–z), $D =$ digits (0–9)$\}$.

(When in positions after the first, both uppercase and lowercase letters are allowed; the first symbol must be capital.)

## DFA states

$q_0 = (0,0)$ (start),

$q_1 = (1,0)$, $q_2 = (2,0)$, $q_3 = (3,0)$, $q_4 = (4,0)$, $q_5 = (5,0)$ (no-digit states)

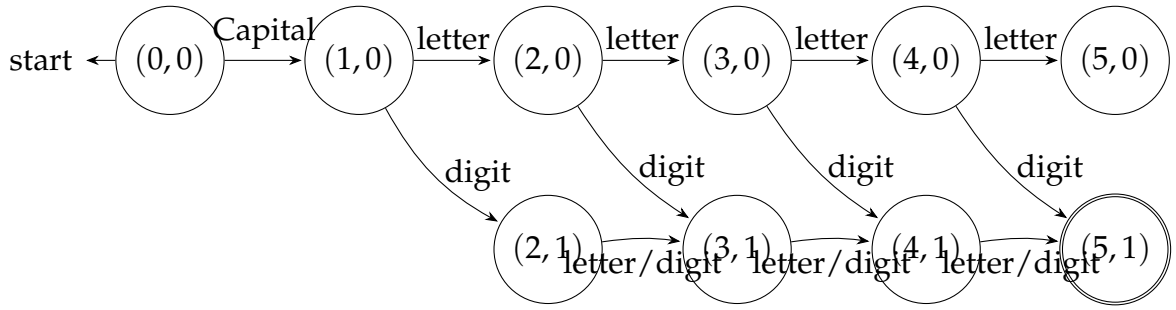
$r_2 = (2,1)$, $r_3 = (3,1)$, $r_4 = (4,1)$, $r_5 = (5,1)$ (digit-seen states).

Accepting state: $r_5 = (5,1)$ only.

### Transition table (condensed)

| Current state | Input class | Next state |
|---|---|---|
| $(0,0)$ | Capital | $(1,0)$ |
| $(0,0)$ | lower-letter or digit | (reject / no transition) |
| $(1,0)$ | letter (A–Z or a–z) | $(2,0)$ |
| $(1,0)$ | digit | $(2,1)$ |
| $(2,0)$ | letter | $(3,0)$ |
| $(2,0)$ | digit | $(3,1)$ |
| $(2,1)$ | letter/digit | $(3,1)$ |
| $(3,0)$ | letter | $(4,0)$ |
| $(3,0)$ | digit | $(4,1)$ |
| $(3,1)$ | letter/digit | $(4,1)$ |
| $(4,0)$ | letter | $(5,0)$ |
| $(4,0)$ | digit | $(5,1)$ |
| $(4,1)$ | letter/digit | $(5,1)$ |
| $(5,0)$ | any | (no outgoing; length exceeded) |
| $(5,1)$ | any | (no outgoing; length exceeded) |

### TikZ diagram: DFA




**Note:** The DFA shown above is the practical deterministic machine used for validation. It accepts exactly the 5-character strings that start with an uppercase letter, contain only letters/digits, and have at least one digit.

## 5 Examples (testing)

- **User1** Inputs: U (capital) $\rightarrow$ s (letter) $\rightarrow$ e (letter) $\rightarrow$ r (letter) $\rightarrow$ 1 (digit) $\Rightarrow$ ends in $(5,1)$: *ACCEPT*.
- **A1b2C** Accept (digits at positions 2 and 4).

- **Z9xY8** Accept (digits at positions 2 and 5).
- **Hey** Reject (length $< 5$).
- **longUsername** Reject (length $> 5$).
- **12345** Reject (first char not capital).
- **NoDigits** Reject (length 7 and no digit; or if 'NoDig' length 5 but no digit then rejects because $(5, 0)$).
- **Invalid!** Reject (contains special character).

# 6 Deliverable: Answers to the "complex problem" sub-questions

Below are brief answers to the questions (a) – (g) in the deliverables.

(a) **Does the solution need in-depth engineering knowledge?**
No; the problem requires solid understanding of finite automata (theory of computation) and careful design of state-space to track a small amount of history (position and whether a digit occurred). It is not engineering-heavy in terms of system architecture, but requires theoretical rigor.

(b) **Does the solution involve wide-ranging or conflicting technical, engineering, and other issues?**
Not significantly. The main technical requirement is correctness of input validation (finite alphabet, length constraint). Practical deployment might consider UX (error messages) or internationalization (non-ASCII usernames), but those are outside the core automaton design.

(c) **Is the solution well-known, or does it require abstract thinking and analysis to formulate?**
The pattern is a standard finite-state pattern (fixed-length constraints + a "must have at least one of X" property). It is well-known in automata theory; however, formalizing it into states and performing NFA-to-DFA conversion needs understanding of the subset construction.

(d) **Does the solution involve infrequently encountered issues?**
No; the issues are common textbook-style constraints (length, character classes, presence of at least one digit).

(e) **Does the solution need adherence to standards and codes of practice?**
For the automaton itself, no. In production username validation, one should follow security and usability standards (e.g., normalization, Unicode handling, preventing homograph attacks); those are outside this assignment.

(f) **Does the solution involve stakeholders with conflicting technical requirements?**
Potentially yes in real systems: product teams may prefer permissive usernames while security teams want stricter rules. Those policy conflicts must be resolved at design time, but they do not change the finite-state reasoning presented here.

(g) **Does the solution involve interdependence between sub-problems or parts?**
The automaton couples *position* and *digit-seen* information (two aspects). This is a simple interdependence which is naturally captured by using state pairs $(i, d)$.

# Conclusion

This document presented a complete automaton-based solution for the username validation problem: an NFA-style design (position + digit flag), formal definition, DFA (via subset construction discussion), diagrams, transition tables, and example tests. The DFA is the structure to use in actual validation code because it is deterministic and efficient (constant small number of states).