CMPU 240 · Theory of Computation

Deterministic Finite Automata

29 January 2026

Assignment 1

- Released after class today
- Due by the start of class on Tuesday
- You're required to do this one in pairs
  - I've posted pairs on Ed
  - You're free to swap as long as both people agree (but there's nothing wrong with working with someone you don't already know!)

CMPU 240 · Spring 2026

Guide to Assignments

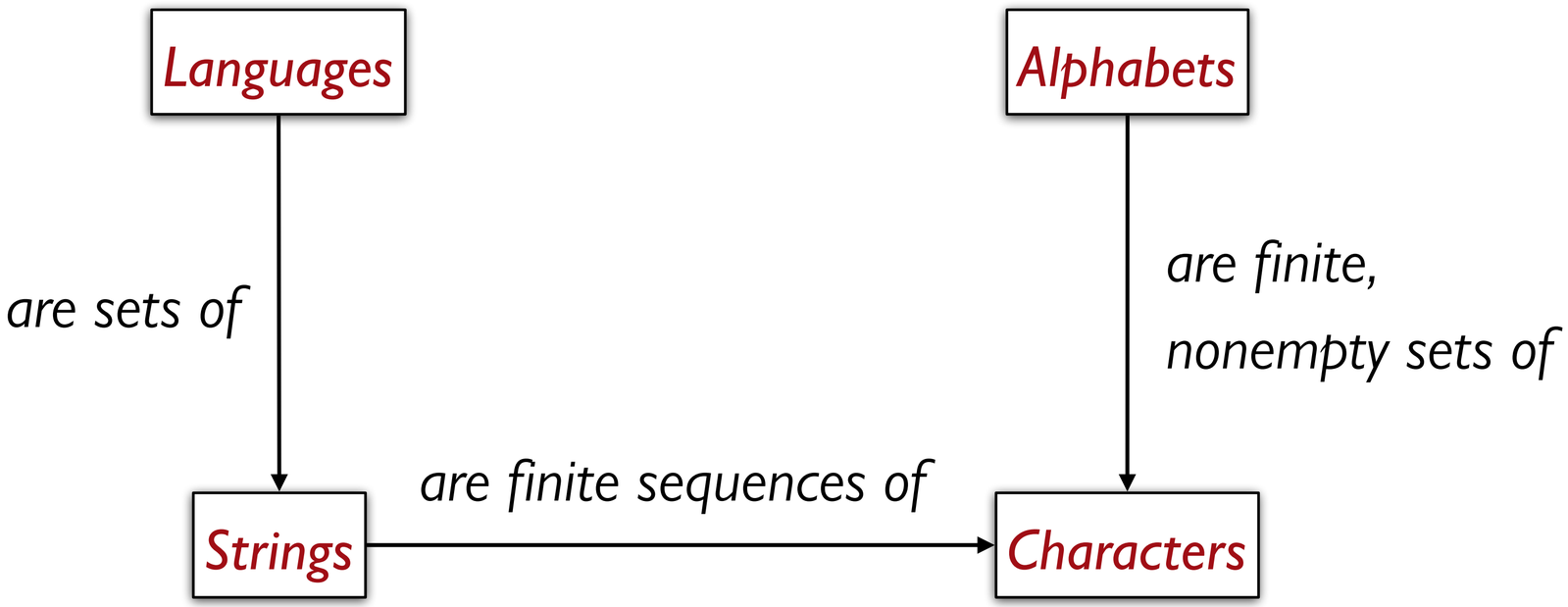
A goal for this course is for you to gain experience using models of computation, formal languages, and proofs to solve problems. The homework assignments are essential practice for you to think about the material outside of class and identify what you don't understand.

Working with a partner

Every homework assignment in CMPU 240 can be completed with a partner. This is a bit unusual, but there are many advantages to working on assignments with a partner!

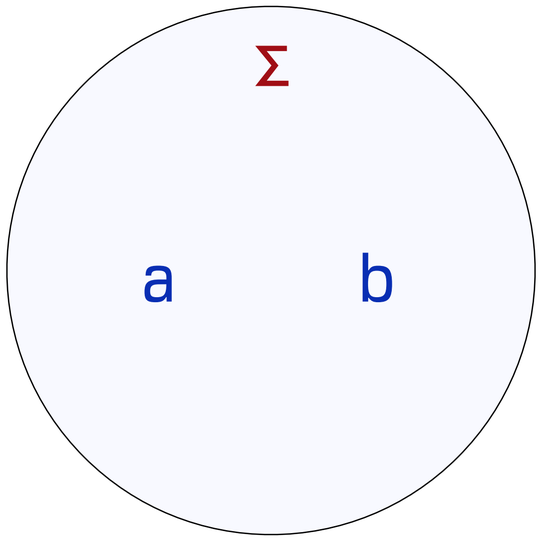
- Your partner can serve as a sounding board for ideas as you're working through trickier problems.
- Having a partner can keep you accountable to make slow and steady progress on the assignment over the week, giving you more time to digest content and ask for help if you need it.
- Once you've written up an answer to a problem, your partner can

Where are we?

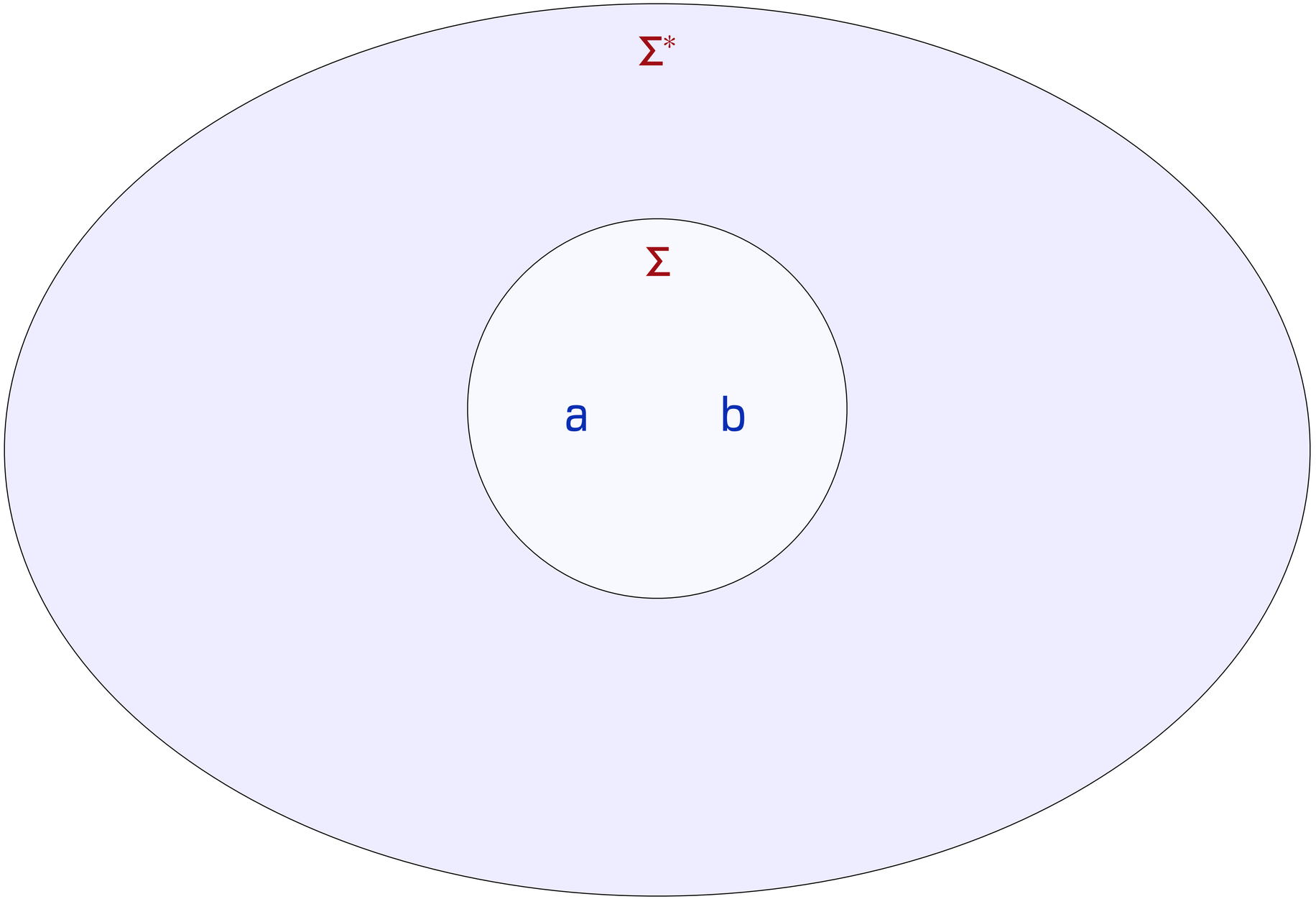

Languages
Alphabets
are sets of
are finite,
nonempty sets of
are finite sequences of
Strings
Characters

For example, if $\Sigma = \{a, b\}$,

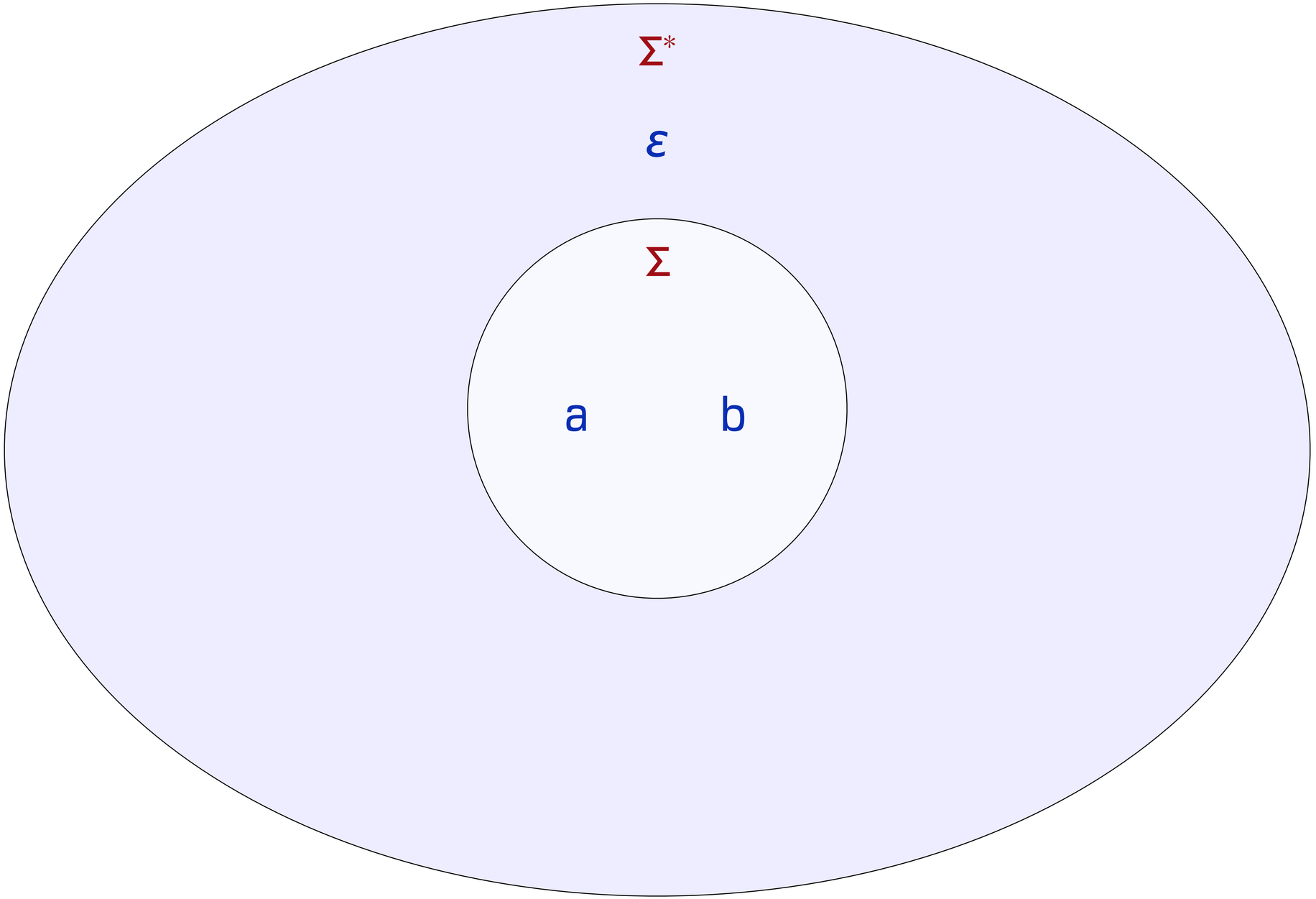
For example, if Σ = {a, b},



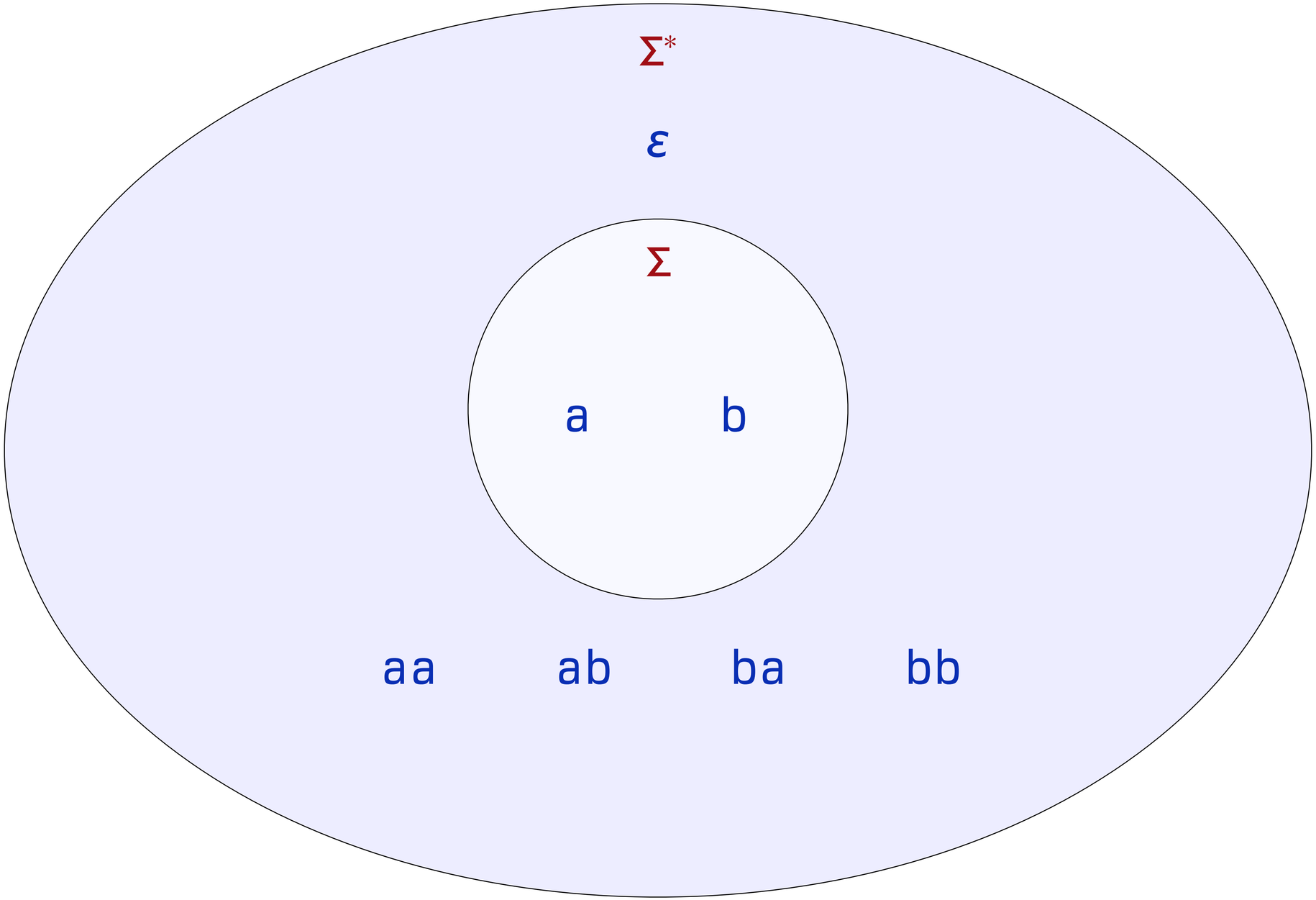
For example, if $\Sigma = \{a, b\}$,



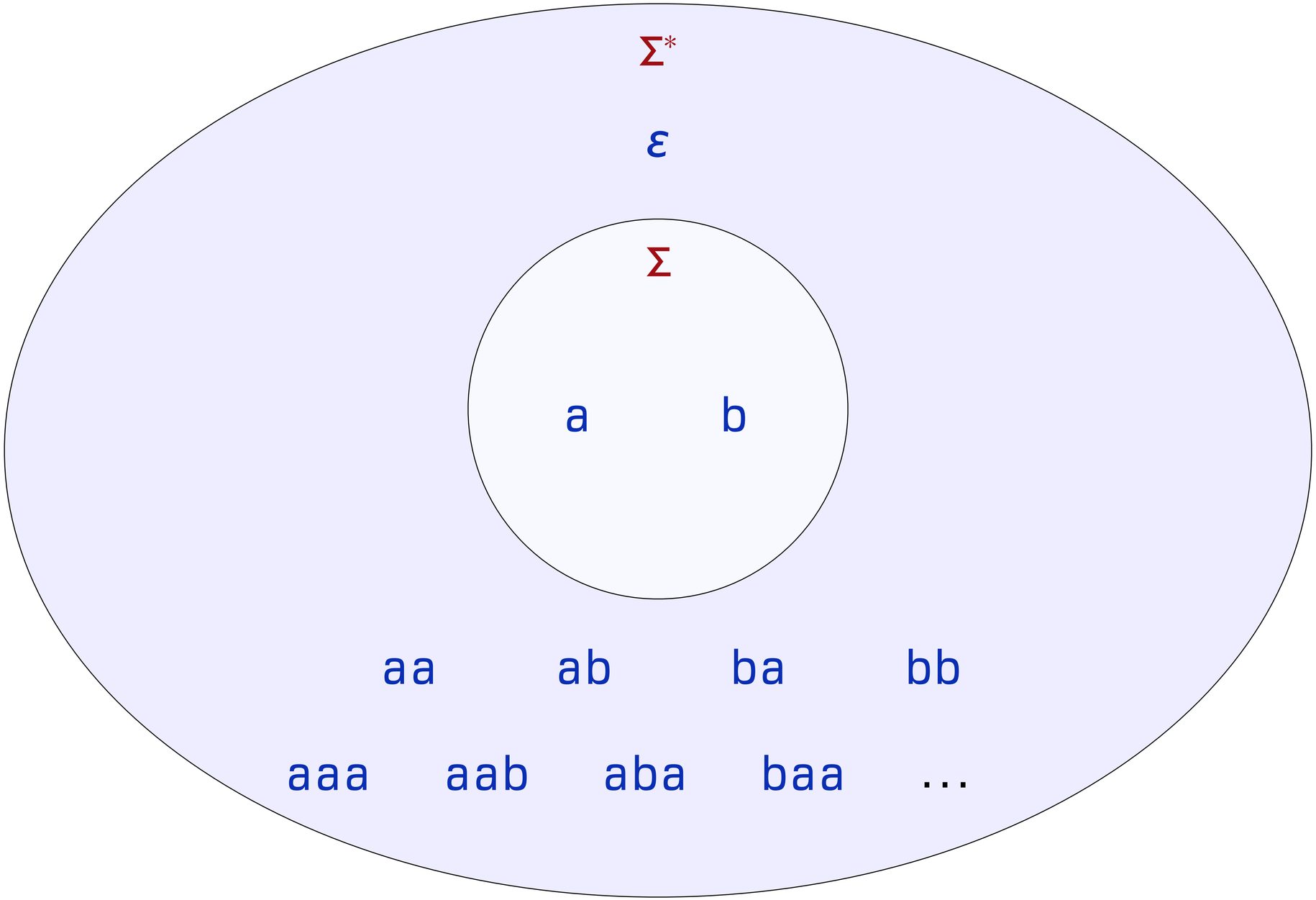
For example, if $\Sigma = \{a, b\}$,



For example, if $\Sigma = \{a, b\}$,



For example, if Σ = {a, b},



Check your understanding

True or false:

$\Sigma \subseteq \Sigma^*$?

$\varepsilon \in \Sigma$?

$\varepsilon \in \Sigma^*$?

$a = aa$?

$ab = ba$?

We represent computational problems as languages.

For example, the problem of testing whether a number is prime could be treated as the language

$$\{w \in \{0, \ldots, 9\}^* \mid \text{the number with decimal representation } w \text{ is prime}\}$$

Check your understanding

True or false:

$\varepsilon \in \mathtt{abc}$?

$\{\mathtt{abc}\} \cup \varnothing = \{\mathtt{abc}\}$?

$\{\mathtt{abc}\} \cup \{\varepsilon\} = \{\mathtt{abc}\}$?

A *finite automaton* is a collection of *states* joined by *transitions*.

- Some state is designated as the *start state*.
- Some number of states are designated as *accept states*.

The automaton processes a string by beginning in the start state and following the indicated transitions.

- If the automaton ends in an accept state, it *accepts* the input.
- Otherwise, the automaton *rejects* the input.

If A is an automaton that processes strings over Σ, the *language of A*, denoted $L(A)$, is the set of all strings A accepts.

Formally,

$$L(A) = \{w \in \Sigma^* \mid A \text{ accepts } w\}$$

A *deterministic finite automaton* (DFA) is the simplest type of automaton we'll study.

A DFA is defined relative to some alphabet Σ.

For each state in the DFA, there must be exactly one transition defined for each symbol in Σ – this is the "deterministic" part!

Designing DFAs

Design strategy for DFAs

At each point in its execution, the DFA can only remember what state it's in.

Therefore, build each state to correspond to some piece of information that you need to remember.

- Each state acts as an indicator of what you've already seen, sufficient to let you decide what to do next.
- There can only be finitely many states, so the DFA can only remember finitely many things.

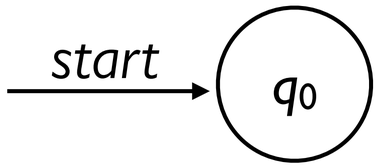
Example

Consider the language

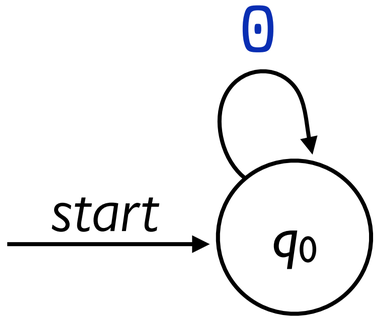
$$L = \{w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ contains } \texttt{11} \text{ as a substring}\}$$

How can we design a DFA to recognize L?

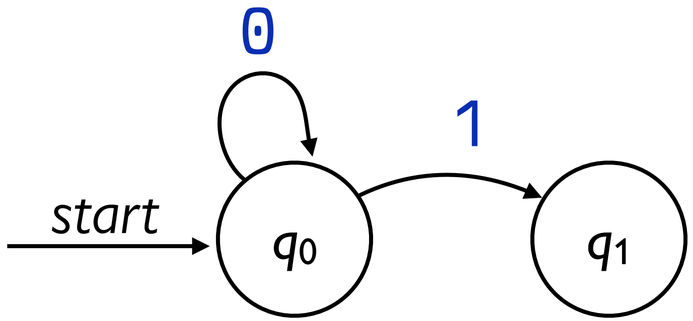
$$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$$



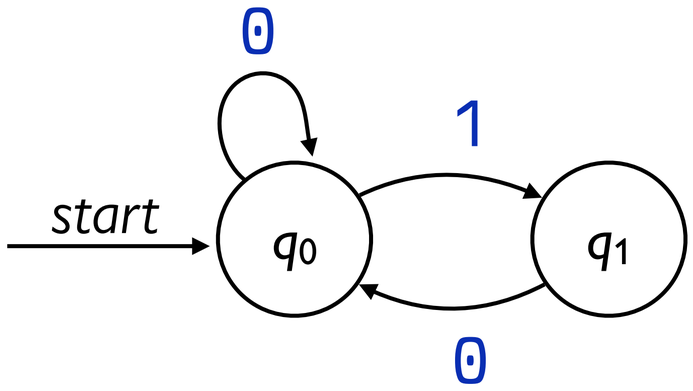
$$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$$



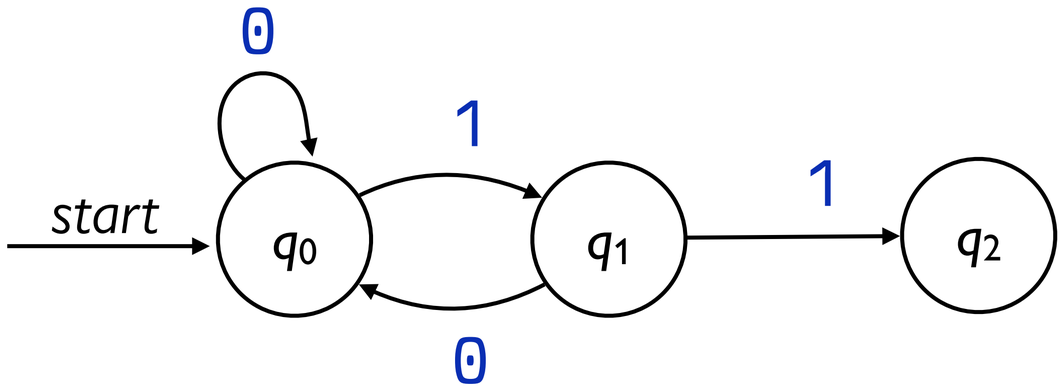
$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$



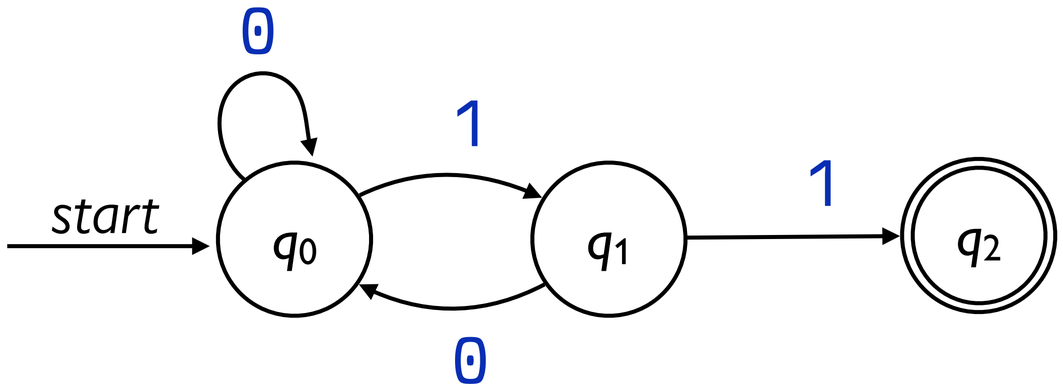
$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$



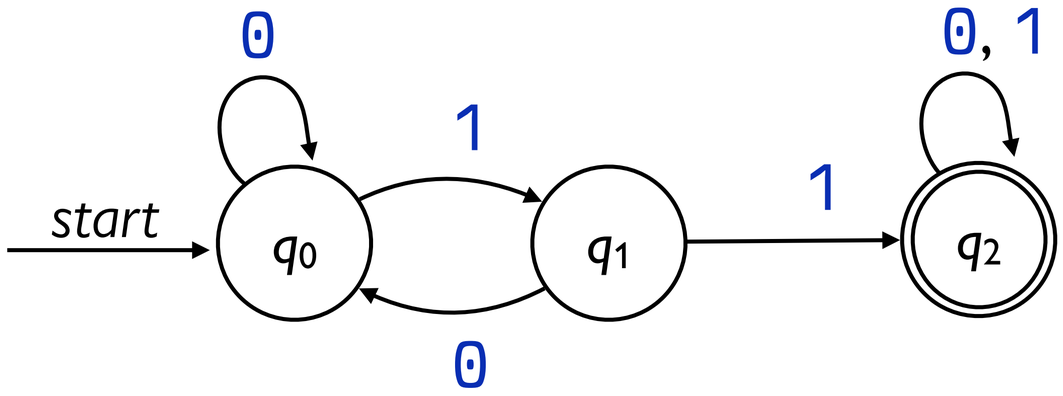
$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$



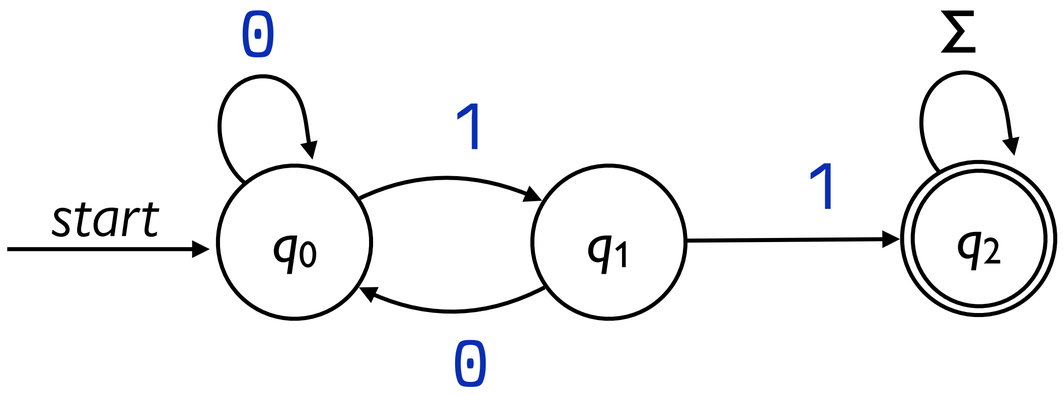
$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$



$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$



$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$



Example

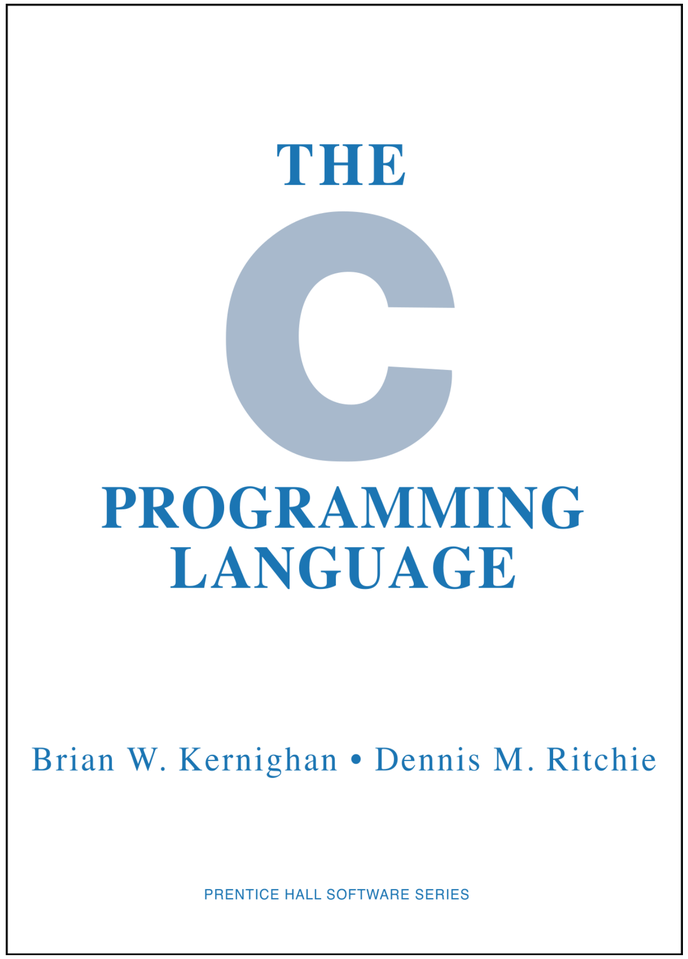
```
#include <stdio.h>

/* print Fahrenheit-Celsius table
    for fahr = 0, 20, ..., 300; floating-point version */
main()
{
    float fahr, celsius;
    int lower, upper, step;

    lower = 0;      /* lower limit of temperature table */
    upper = 300;    /* upper limit */
    step = 20;      /* step size */

    fahr = lower;
    while (fahr <= upper) {
        celsius = (5.0/9.0) * (fahr-32.0);
        printf("%3.0f %6.1f\n", fahr, celsius);
        fahr = fahr + step;
    }
}
```

Example



C-style comment

```
#include <stdio.h>

/* print Fahrenheit-Celsius table
    for fahr = 0, 20, ..., 300; floating-point version */
main()
{
    float fahr, celsius;
    int lower, upper, step;

    lower = 0;      /* lower limit of temperature table */
    upper = 300;    /* upper limit */
    step = 20;      /* step size */

    fahr = lower;
    while (fahr <= upper) {
        celsius = (5.0/9.0) * (fahr-32.0);
        printf("%3.0f %6.1f\n", fahr, celsius);
        fahr = fahr + step;
    }
}
```

Example

Consider the language

$$L = \{w \in \{\texttt{a}, \texttt{*}, \texttt{/}\}^* \mid w \text{ represents a C-style comment}\}$$

We're using the symbol `a` as a placeholder for any character that isn't a star or slash (including spaces) to keep things simple.

Example

Just like when you're programming, it helps to come up with sets of test cases to accept and reject.

$$L = \{w \in \{\texttt{a}, \texttt{*}, \texttt{/}\}^* \mid w \text{ represents a C-style comment}\}$$

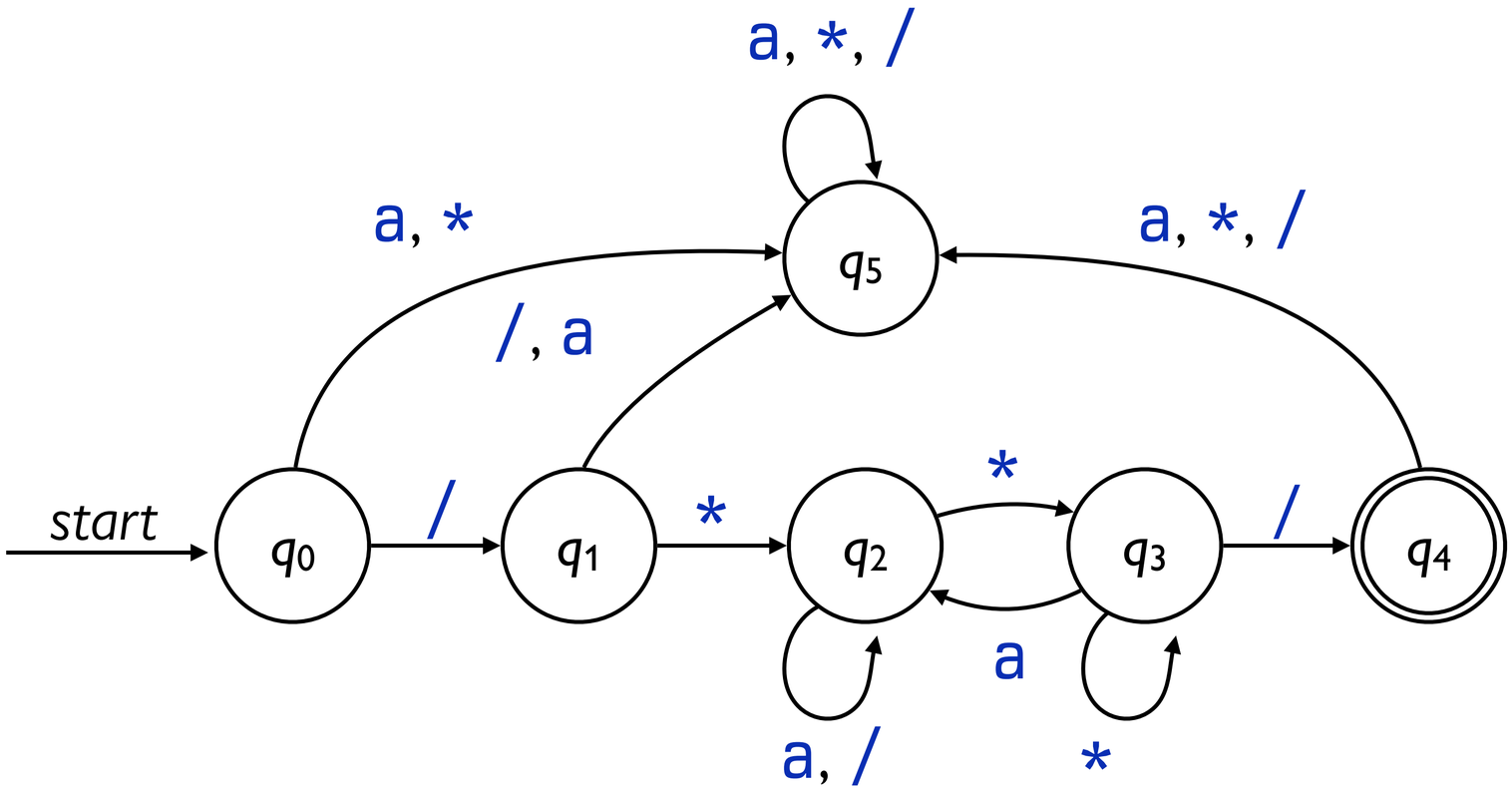
Accept

```
/*a*/
/**/
/***/
/*aaa*aaa*/
/*a/a*/
```

Reject

```
/**
/**/a/*aa*/
aaa/**/aa
/*/
/**a
//aaaa
/*/**/*/
```

$L = \{w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment}\}$



If a machine can't remember all the symbols it has seen so far in an input string, it has to change state based on other information, e.g.,

L_1 = the set of all strings with an odd number of `1`s over Σ = {`0`, `1`}

Don't need to remember exactly how many `1`s have been seen – just whether we've read an even or odd number.

Exercise

Build an automaton to recognize the set of strings that end with `ing`.

Formal DFAs

Formal definition of a deterministic finite automaton (DFA)

A DFA is represented as a five-tuple $(Q, \Sigma, \delta, q_0, F)$ where

Q is a finite set of *states*,

Σ is the *alphabet*, a finite set of input symbols,

δ: $Q \times \Sigma \rightarrow Q$ is the *transition function*,

$q_0 \in Q$ is the *start state*, and

$F \subseteq Q$ is a set of zero or more *accept states*.

Transition function δ

Takes two arguments: a state and an input symbol.

$\delta(q,$ a$)$ = the state the DFA goes to when it is in state q and reads input symbol a.

Since δ is a total function; there is always a next state.

If there's no transition you want, you must add a "dead state".

Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let w be a string, $w_1 w_2 \ldots w_n$, where each $w_i \in \Sigma$.

M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$
3. $r_n \in F$

Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let w be a string, $w_1 w_2 \ldots w_n$, where each $w_i \in \Sigma$.

M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$

It begins at the start state,

2. $\delta(r_i, w_{i+1}) = r_{i+1}$
   for $i = 0, \ldots, n-1$

3. $r_n \in F$

Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let w be a string, $w_1 w_2 \ldots w_n$, where each $w_i \in \Sigma$.

M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$

2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$

3. $r_n \in F$

It begins at the start state,

each transition is allowed by the transition function for the corresponding input symbol, and

Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let w be a string, $w_1 w_2 \ldots w_n$, where each $w_i \in \Sigma$.

M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$

   It begins at the start state,

2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$

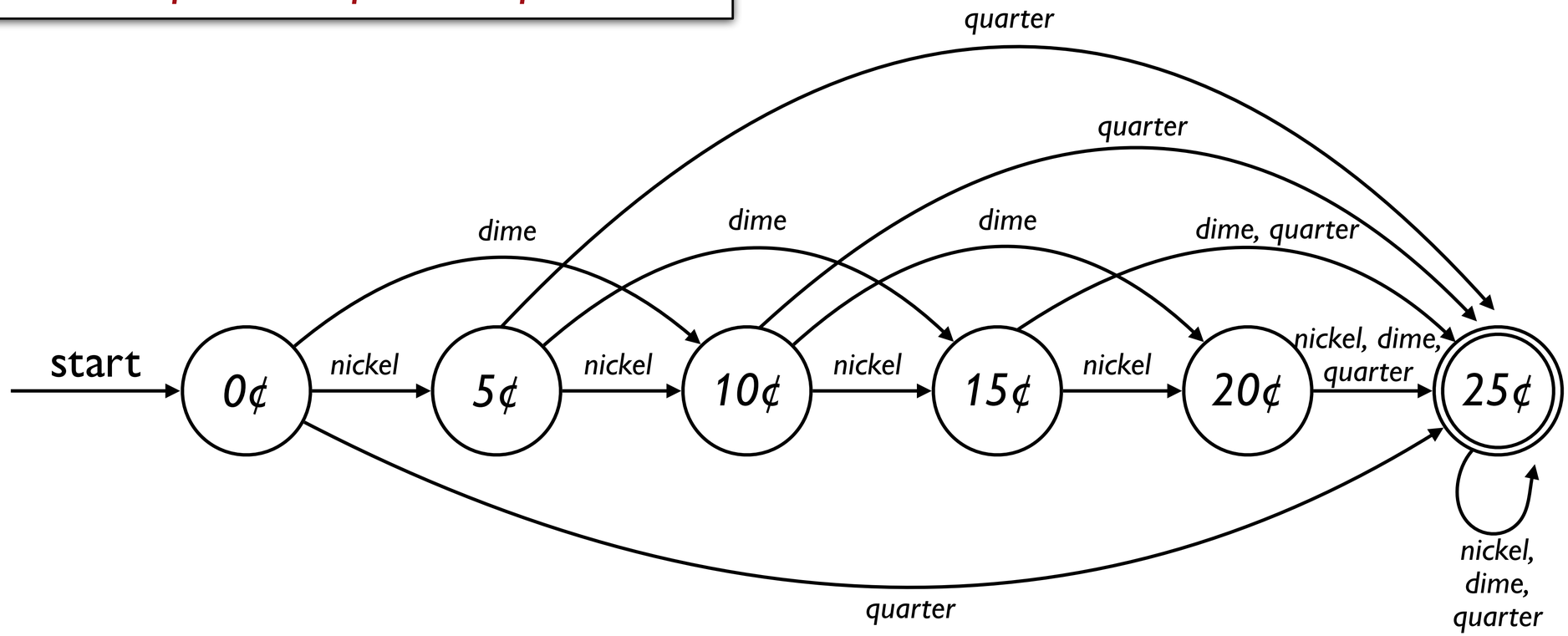
   each transition is allowed by the transition function for the corresponding input symbol, and

3. $r_n \in F$

   it ends in an accept state.

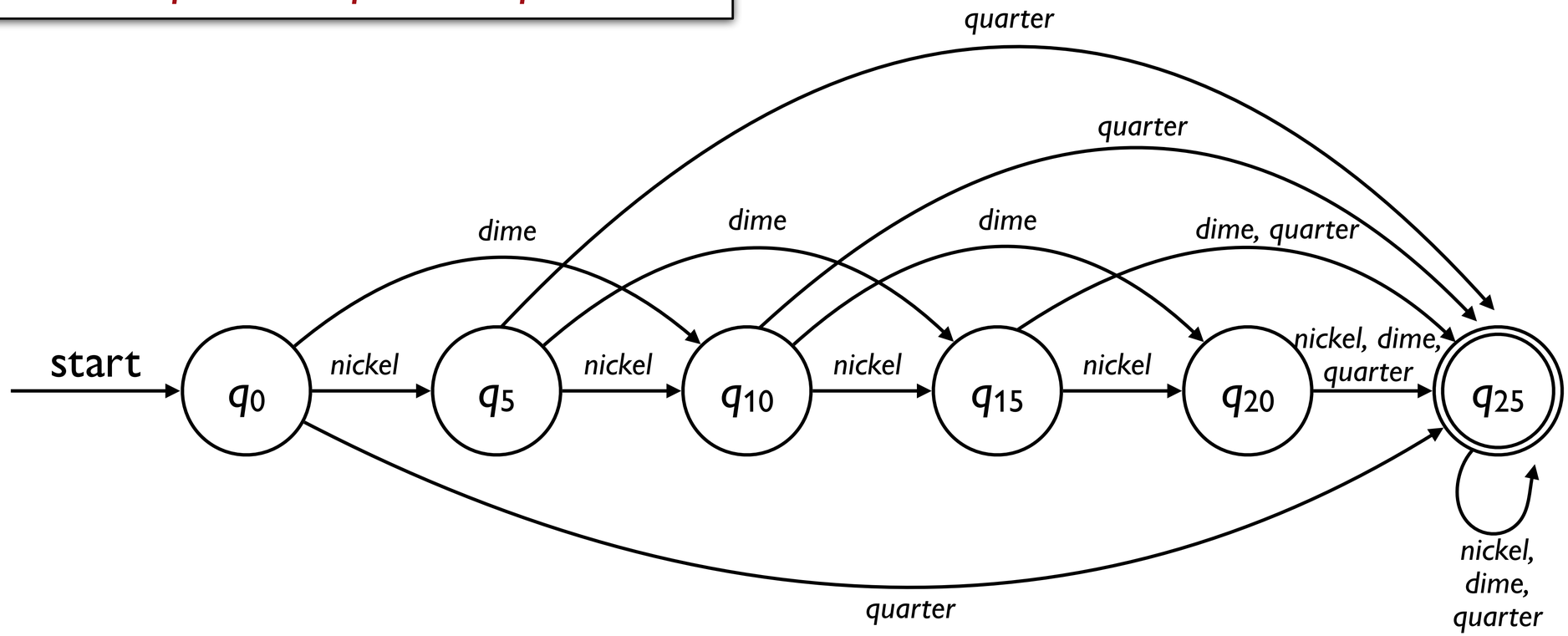
Example

What's the formal definition of this DFA?



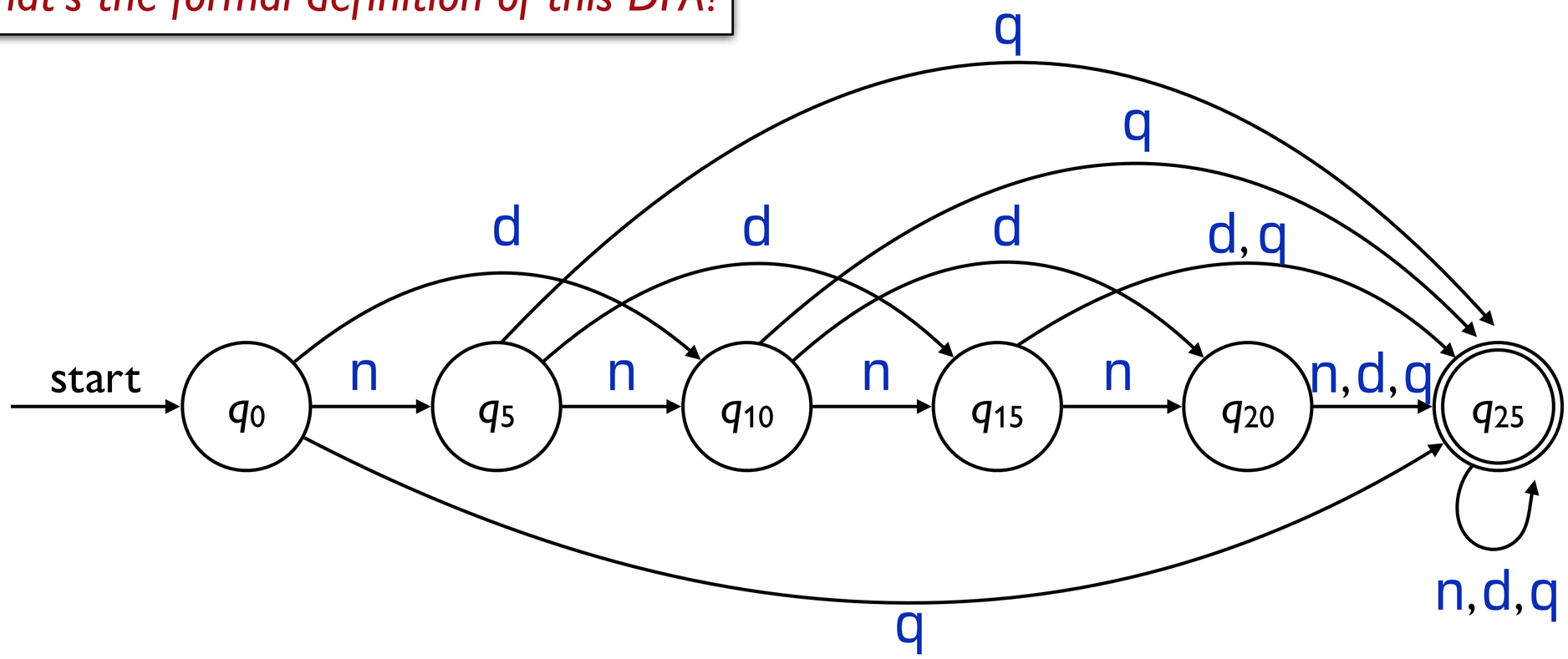
Example

What's the formal definition of this DFA?



Example

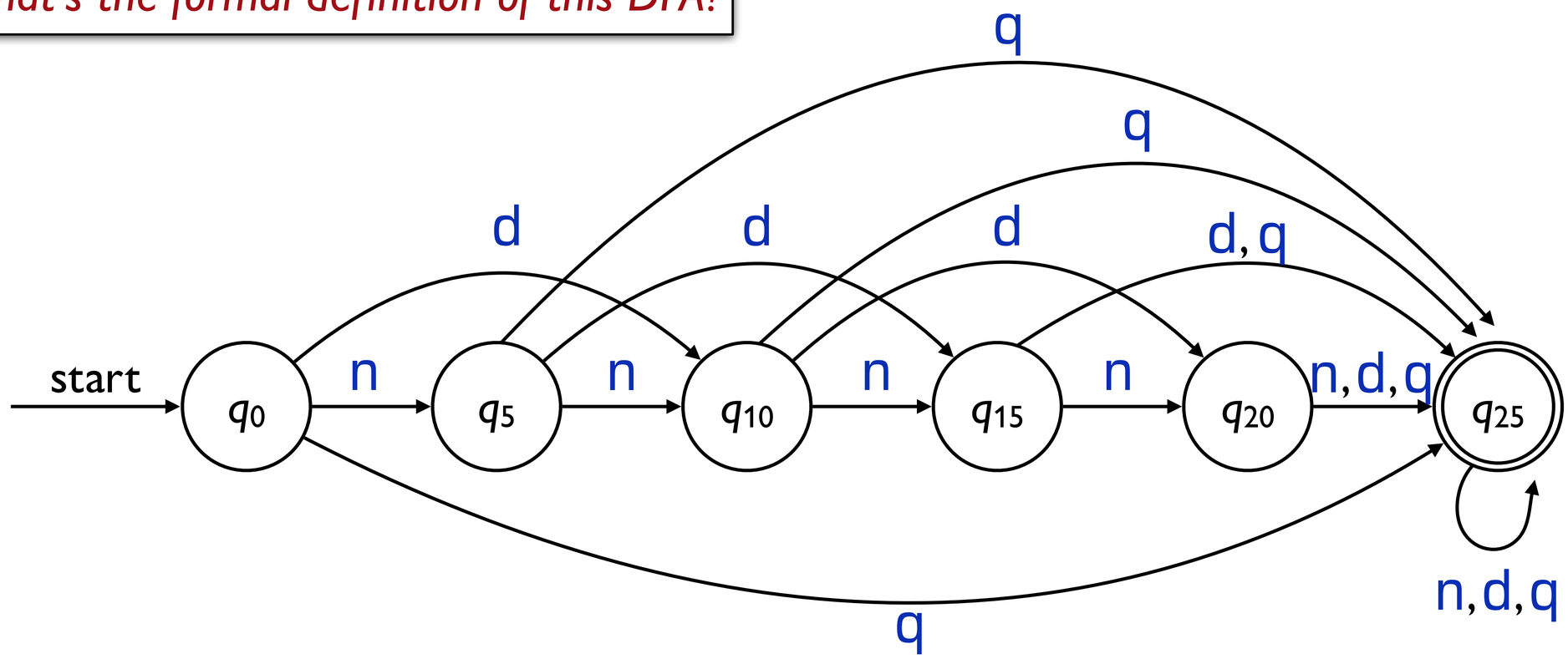
What's the formal definition of this DFA?



$M_{news} = (Q, \Sigma, \delta, q_0, F)$

Example

What's the formal definition of this DFA?

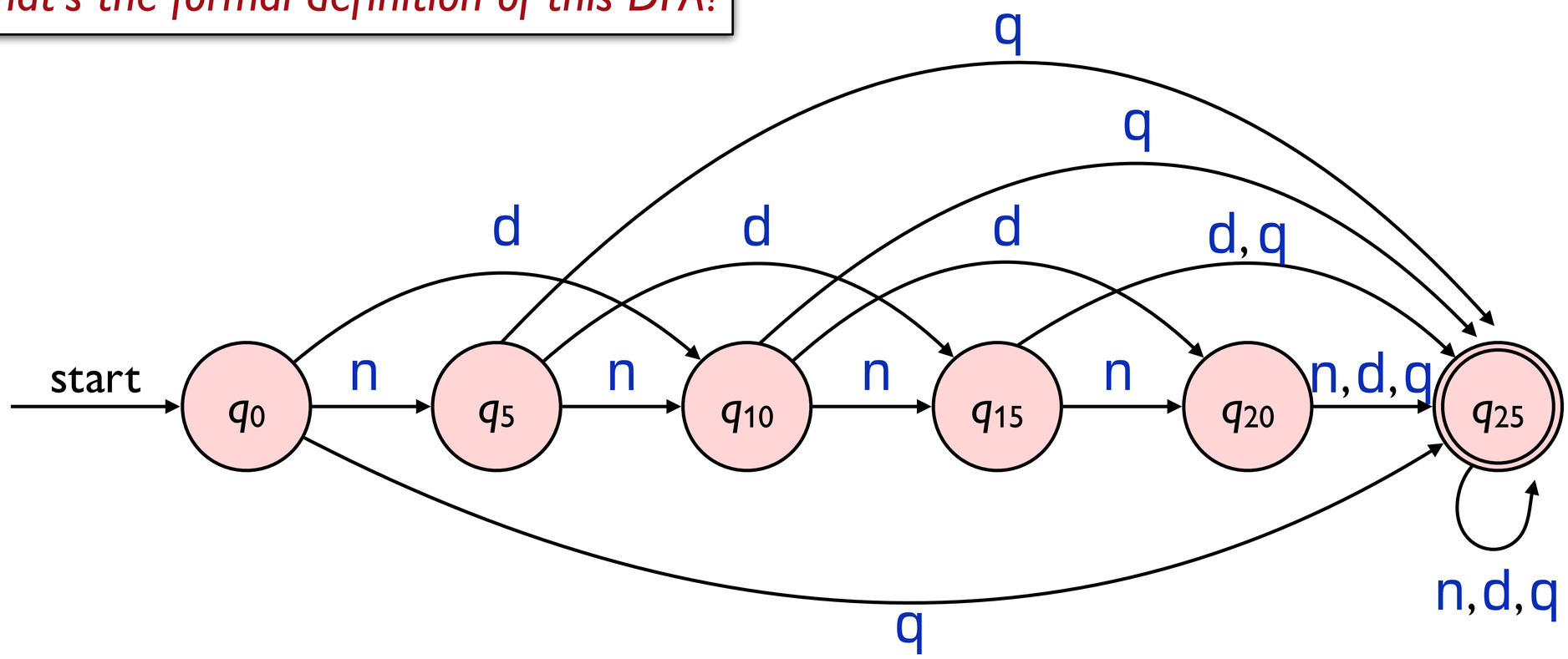


Example

$M_{news} = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\}$

What's the formal definition of this DFA?



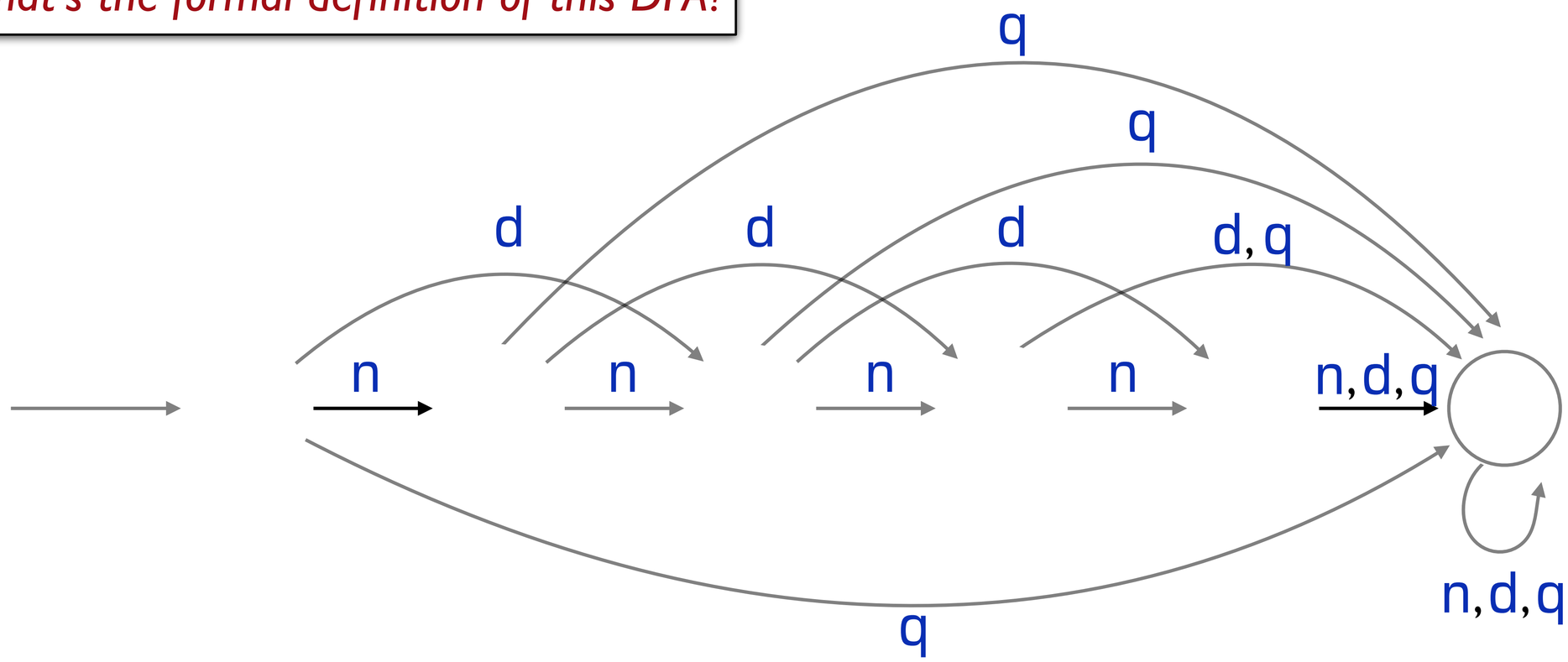
Example

$M_{news} = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\}$

$\Sigma = \{n, d, q\}$

What's the formal definition of this DFA?



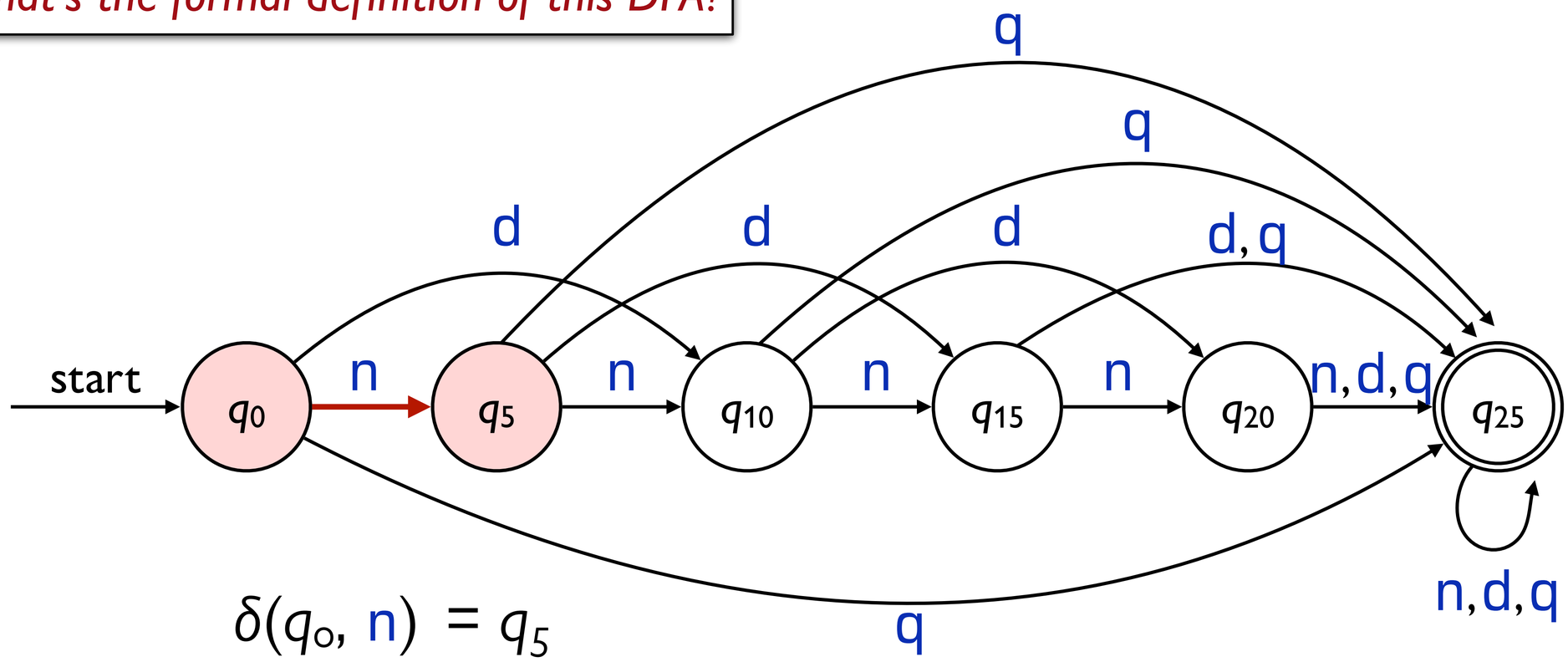
Example

$M_{news} = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\}$

$\Sigma = \{n, d, q\}$

What's the formal definition of this DFA?



$\delta(q_0, n) = q_5$

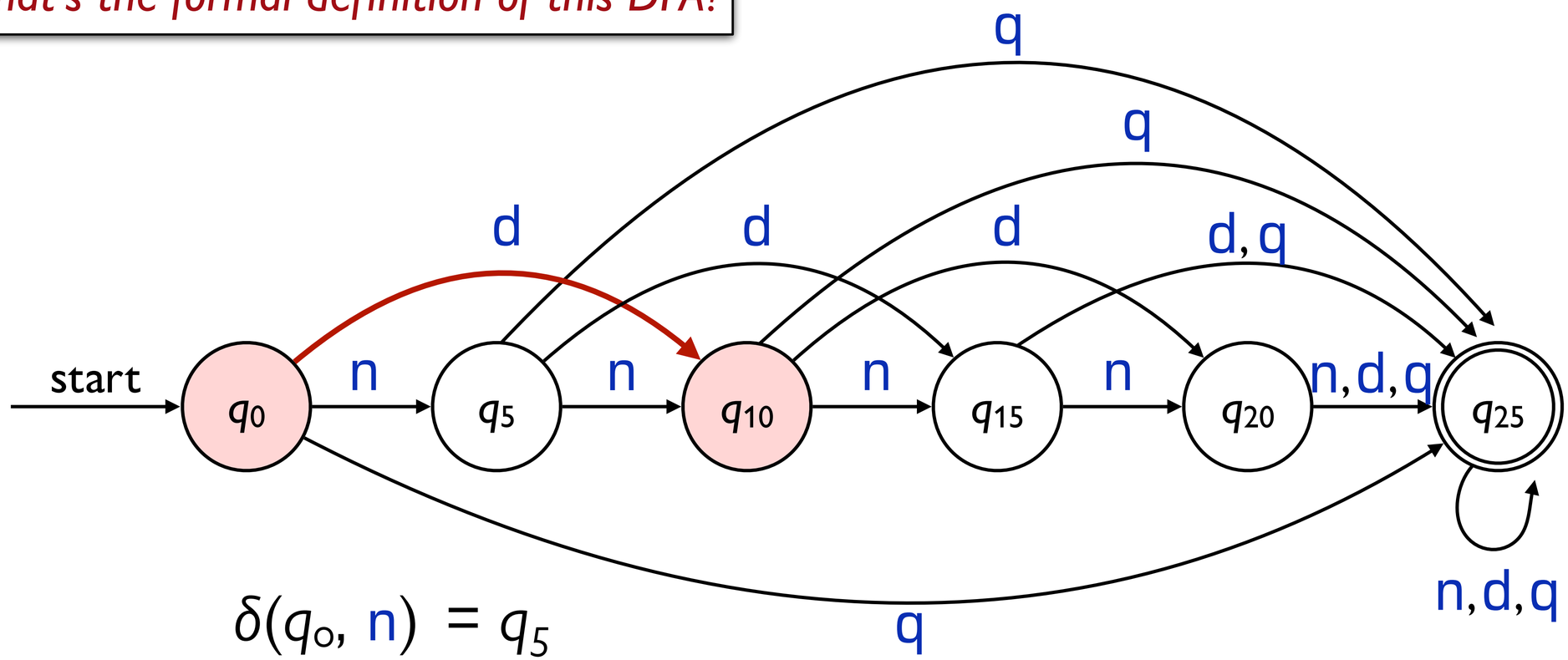
Example

$M_{news} = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\}$

$\Sigma = \{n, d, q\}$

What's the formal definition of this DFA?



$\delta(q_0, n) = q_5$

$\delta(q_0, d) = q_{10}$

Example

What's the formal definition of this DFA?

$$M_{news} = (Q, \Sigma, \delta, q_0, F)$$

$$Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\}$$

$$\Sigma = \{\text{n}, \text{d}, \text{q}\}$$

$$\delta(q_0, \text{n}) = q_5$$

$$\delta(q_0, \text{d}) = q_{10}$$

$$\delta(q_0, \text{q}) = q_{25}$$

$$\delta(q_5, \text{n}) = q_{10}$$

$$\delta(q_5, \text{d}) = q_{15}$$

$$\delta(q_5, \text{q}) = q_{25}$$

$$\vdots$$

$$F = \{q_{25}\}$$

Example

What's the formal definition of this DFA?

$$M_{news} = (\{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\}, \{n, d, q\}, \delta, q_0, \{q_{25}\})$$

$$\delta(q_0, n) = q_5$$

$$\delta(q_0, d) = q_{10}$$

$$\delta(q_0, q) = q_{25}$$

$$\delta(q_5, n) = q_{10}$$

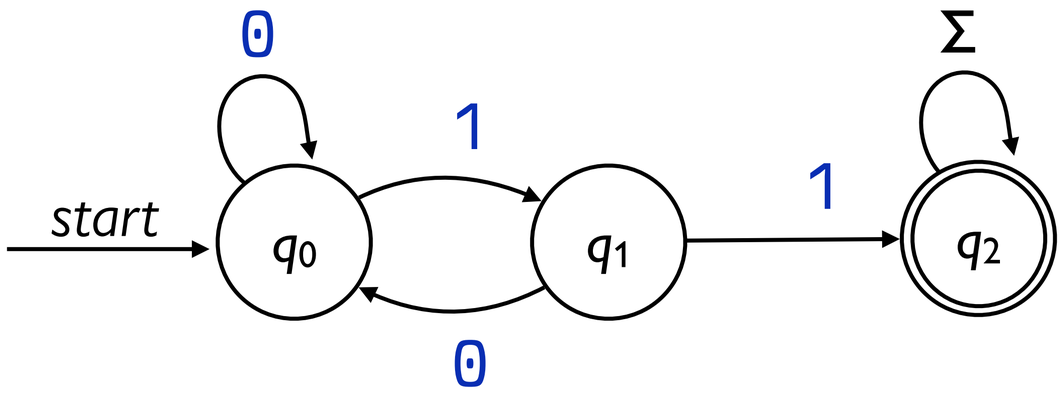
$$\delta(q_5, d) = q_{15}$$

$$\delta(q_5, q) = q_{25}$$

$$\vdots$$

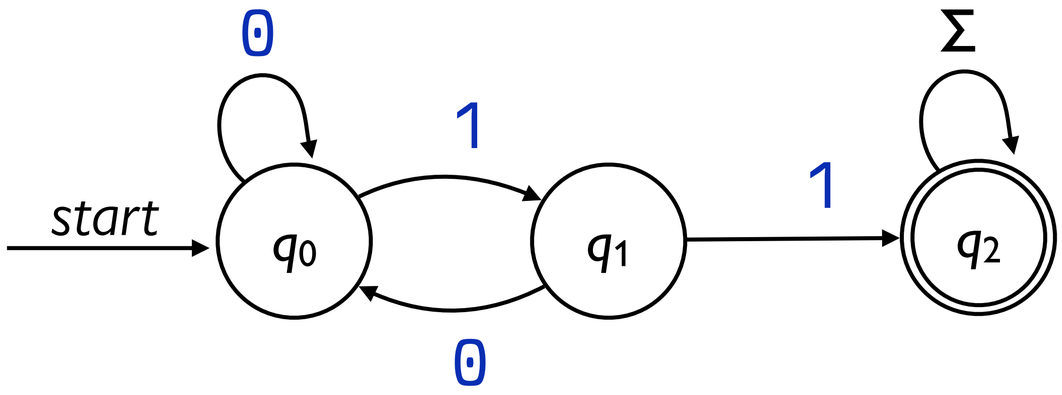
Tabular DFAs

Another way we can write down the transition function for a DFA is as a transition table:



	0	1
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2

Another way we can write down the transition function for a DFA is as a transition table:



		0	1
→	q_0	q_0	q_1
	q_1	q_0	q_2
*	q_2	q_2	q_2

*By marking the start state with → and accept states with *, the transition table that defines δ also specifies the entire DFA!*

Tabular DFAs suggest how easy it is to implement a DFA in software.

```
transition_table = {
    "q0": {"0": "q0", "1": "q1"},
    "q1": {"0": "q0", "1": "q2"},
    "q2": {"0": "q2", "1": "q2"}
}

accept_states = ["q2"]

def run_dfa(word: str) -> bool:
    state = "q0"
    for char in word:
        state = transition_table[state][char]
    return state in accept_states
```

CMPU 240 · Theory of Computation · Spring 2026

Class 3: Tabular DFAs

[2] 5s

```
%%capture

%pip install tock

from tock import *
```

Drawing automata

We can draw a DFA:

1. Click to make a new state
2. Double click on a state to make it an accept state.
3. Drag from one state to another to add a transition.
4. When the transition is for multiple symbols, put each on a newline (don't use commas the way we draw them in class or in the textbook).

[4] 0s

```
dfa = FiniteAutomaton()
dfa.edit()
```

Regular languages

Regular languages

DEFINITION A language L is called a *regular language* if there exists a DFA D such that $L(D) = L$.

If L is a language and $L(D) = L$, we say that D *recognizes* the language L.

Acknowledgments

This lecture incorporates material from:

David Chiang, University of Notre Dame

Nancy Ide, Vassar College

Keith Schwarz, Stanford University

Michael Sipser, *Introduction to the Theory of Computation*