SE 3310b

Theoretical Foundations of Software Engineering

Week 2:

DFAs Continued. Introduction to Non-determinism.

Aleksander Essex

Western Engineering

DFAs Continued

Regular Language

Let A be a finite automaton and let L be the set of all strings that are accepted by A, *i.e.*, $L = \{s : A \text{ accepts } s\}$. We call L the **language** of A. Similarly we say A **recognizes** L.

Definition 1 (Regular Language)**.**

A language L is called a ***regular language*** if there exists some finite automaton A that recognizes it.

Strategies for Creating DFAs

Suppose I give you an expression in set-builder notation, or simply an English language description of a regular language and asked you to give me the associated DFA. How would you do it?

1. Start by writing out all the possible states you think you will need, and assign them some kind of semantic meaning
2. Assign a start state, and final state(s)
3. Add in transitions to define the behavior

DFA Example 1

Give a deterministic finite automaton that accepts all strings with an odd number of `0`'s and an odd number of `1`'s:

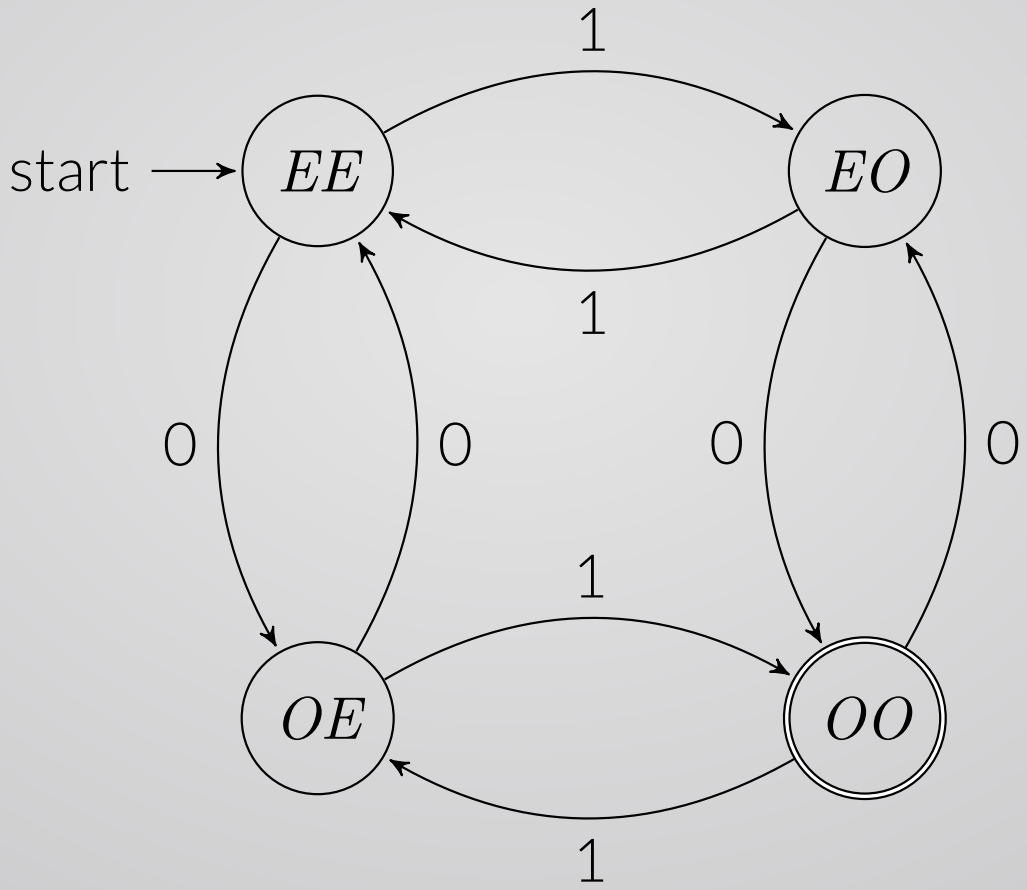
1. **Define states**: what are all the possible scenarios (i.e., states) we might encounter as we read a string? Our string could have:
   - 1.1 Even 0's, even 1's
   - 1.2 Even 0's, odd 1's
   - 1.3 Odd 0's, even 1's
   - 1.4 Odd 0's, odd 1's (our final state as per the question)

   Let's call these states: EE, EO, OE, OO respectively. EE is our start state, since zero is even. OO is our accept state as per the question.

DFA Example 1 Cont'd

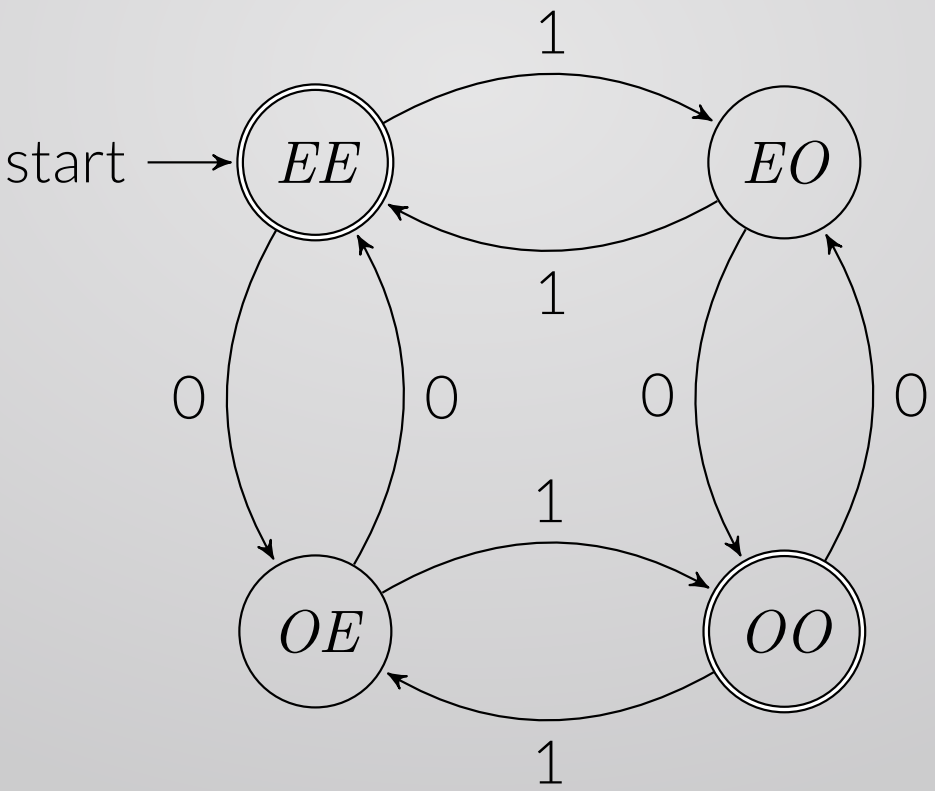
2. **Define a start state**: If you haven't seen any 0's or 1's yet, which of our states would that correspond to?
3. **Define final state(s)**: The question asks us to accept strings with even number of 0's and even number of 1's. Which of our states does that correspond to?
4. **Fill in the arrows to define behavior**: for each state, consider the scenario where you receive a '0' as input, and a '1' as input and then transition to the appropriate state.

DFA Example 1 Cont'd



DFA Example 2

Suppose we modify the previous example to accept an even number of 0's and an even number of 1's ***or*** an odd number of 0's and an odd number of 1's, *i.e.*, EE and OO accept. We can reuse our previous solution and just make EE an accept state:



DFA Example 2 Cont'd

But is the best we can do? Are we now maybe using more states than we strictly need to? What if we combine the EE/OO (accepting) states, and combine the EO/OE (non-accepting) states?

Exercise: Does there exist a set of transitions that will recognize our language? If so, fill in the transitions.

DFA Example 3

Give a DFA that accepts all strings that, when interpreted as a binary integer, are divisible by 4, *i.e.*,give the DFA for:

$$L = \{s : s = t00, t \in \{0,1\}^*\}$$

Using our strategy from before:

1. **Define states**: what are all the possible scenarios (i.e., states) we might encounter as we read a string? At any given moment, our string:
   - 1.1 Ends in no 0's
   - 1.2 Ends in one 0
   - 1.3 Ends in two 0's

DFA Example 3 Cont'd

We also need to consider two special cases:

1. $s = \varepsilon$. The empty string should *not* be accepted.
2. $s = \{0\}^+$. A string of one or more zeros *should* be accepted.

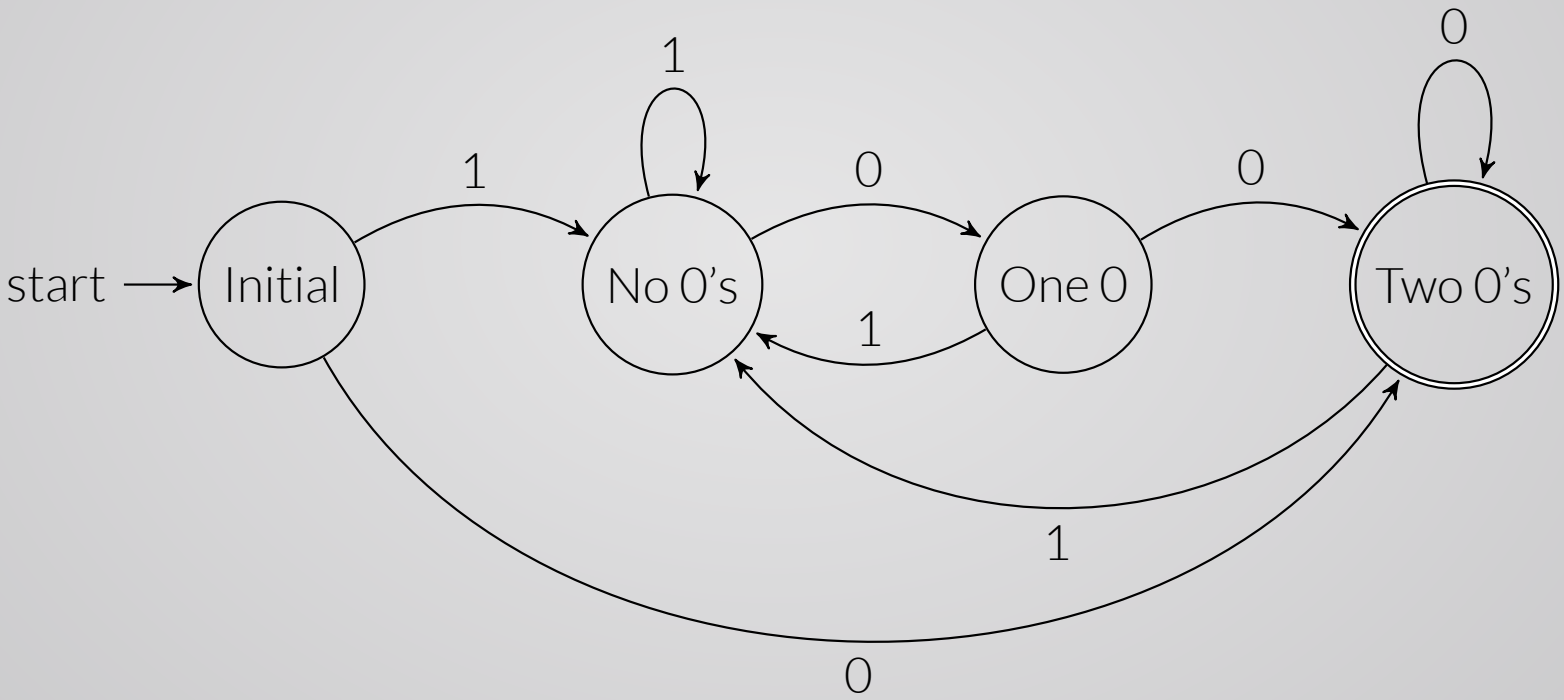
We can handle these cases by adding an additional (initial) state:

1. Initial state
2. Ends in no 0's
3. Ends in one 0
4. Ends in two 0's

DFA Example 3 Cont'd

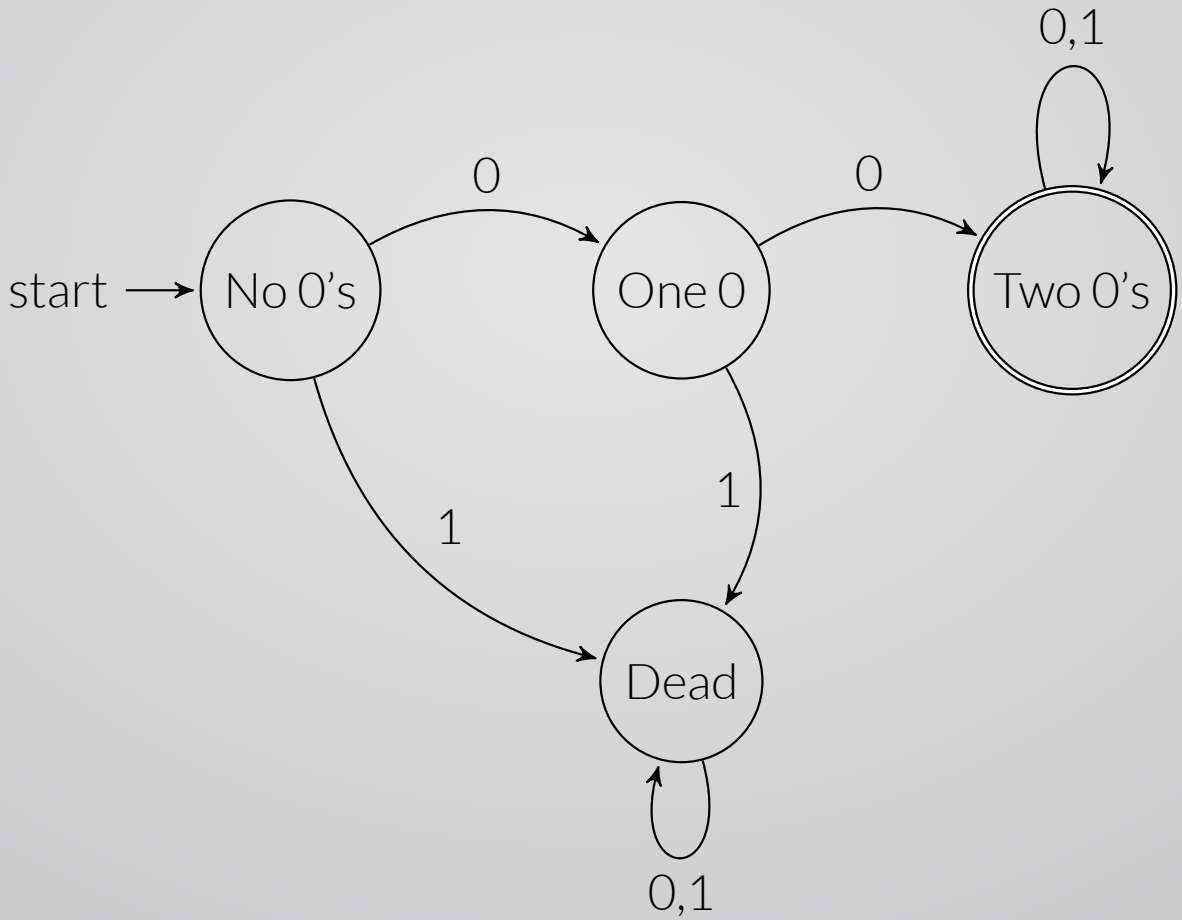
2. **Define a start state**: Let's begin in the "initial state".
3. **Define final state(s)**: From the question, our final state is the "ends in two 0's" state.
4. **Fill in the arrows to define behavior**: See diagram.

DFA Example 3 Cont'd



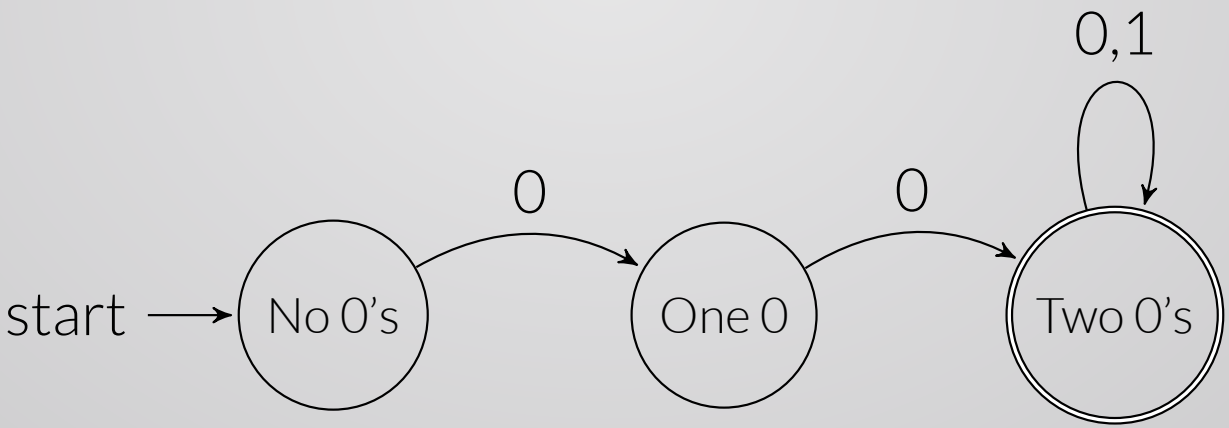
DFA Example 4

How about the language of all strings that start with 2 zeros?



DFA Example 4 Cont'd

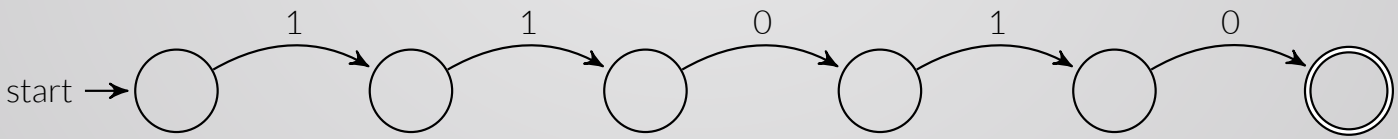
The **dead state** is a state that cannot ever lead to accepting state regardless of any future input. For notational convenience, we will omit dead states and any transitions leading to them:



DFA Example 5

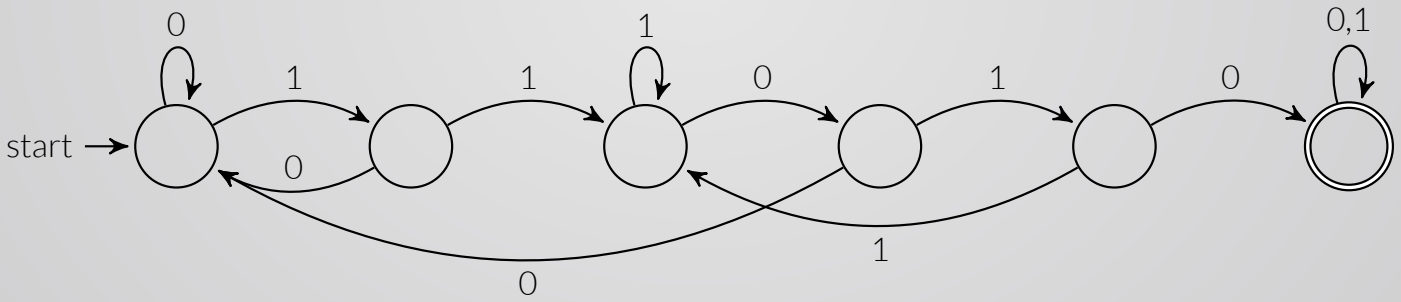
Give a DFA that accepts all strings containing the string `11010`.

Start out with a straight-line set of states and transitions:



DFA Example 5 Cont'd

Fill in the other arrows paying attention to the most recently seen characters:



DFA Example 5 Cont'd

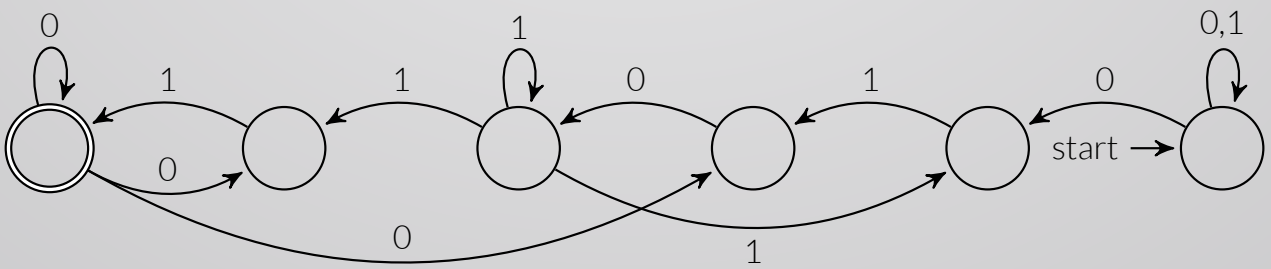
This is an example of string matching/searching, which is an important application of finite automata. The strategy is to use states to represent "progress."

The work comes in figuring out how far back you have to go on a particular mismatch. The Knuth-Morris-Prat algorithm can generate these kind of DFAs in an amount of time that grows linearly in the number of states/alphabet.

DFA Example 6

What if we asked to give a DFA that accepts all strings containing the **reverse** of `11010`. Could we take our `11010` machine and run it backwards? How could we do that?

1. We could make the starting state the accept state and vice versa,
2. Reverse arrow direction



DFA Example 6 Cont'd

At first glance this change wouldn't seem to make sense: some states have more than one arrow coming out of them with the same symbol. For example, if you receive a 0 in the start state, which state should you transition to?

Or could you maybe somehow take both **at the same time**?

Non-Determinism

Determinism vs. Non-Determinism

Given current state $q_i \in Q$ and next input symbol $x \in \Sigma$, the transition function δ of a DFA defines exactly one state $q_j \in Q$ to move to. We call this **determinism**: the next state is *determined* by the current state and next input symbol.

In graphical terms this means **exactly one** arrow comes out of every state for each letter in the alphabet.

Determinism vs. Non-Determinism

Non-determinism challenges the idea that there must be a single path to a computation. What if many paths of the computation could be followed at the same time? In that sense non-determinism can be viewed as a kind of parallel computation in which many ”threads” can run concurrently. If **at least one** such thread accepts, the computation accepts.

Non-deterministic Finite Automaton

A **Non-deterministic Finite Automaton** (NFA) is like a DFA, except that:

1. One **or more** transitions are allowed for each symbol
2. Transitions can be made without reading *any* symbol (so-called ε-transitions)
3. The machine accepts if **any** sequence of choices leads to an accept state

Non-deterministic Finite Automata (NFA)

Definition 2 (Non-deterministic Finite Automaton (NFA))**.**

A *non-deterministic finite automaton* is a $5-$tuple $(Q, \Sigma, \delta, q_0, F)$

1. Q: a finite set of ***states***
2. Σ: a finite ***alphabet***
3. δ: $Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$: a ***transition function***
4. $q_0 \in Q$: the ***starting state***
5. $F \subseteq Q$: a set of ***accepting states***

Non-deterministic Finite Automata (NFA)

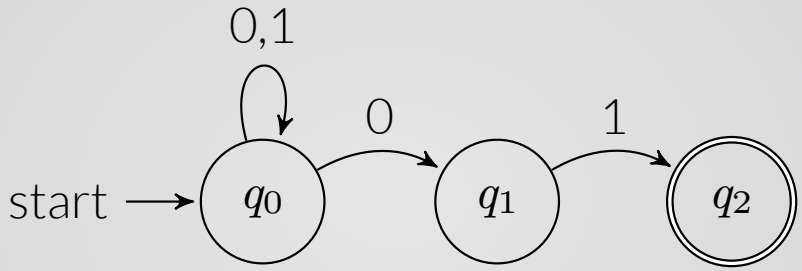
The main differences in NFAs relative to DFAs pertain to the alphabet and the transition function δ.

- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$
- Recall $\mathcal{P}(Q)$ denotes the **power set** of Q.

In other words, the transition function takes a state and a symbol (or the empty string ε) and and produces a *set of possible next states*.

NFA Example 1

Give an NFA that accepts string that end in `01`:

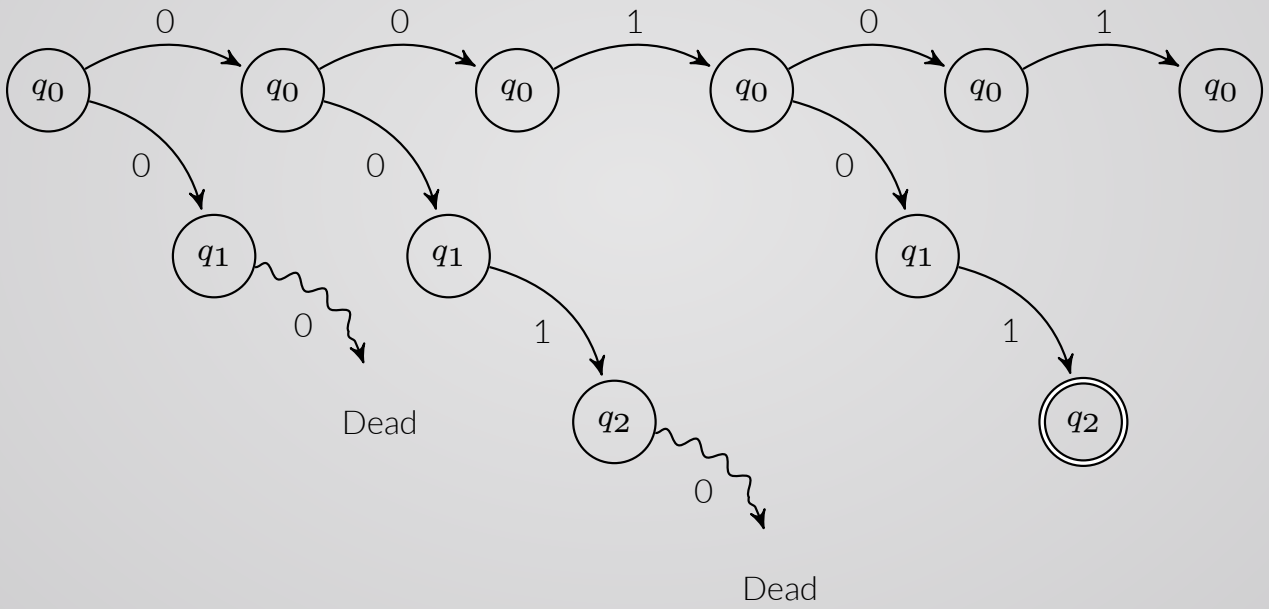


Formally, $A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$ where δ =

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\varnothing$	$\{q_2\}$
q_2	$\varnothing$	$\varnothing$

NFA Example 1 Cont'd

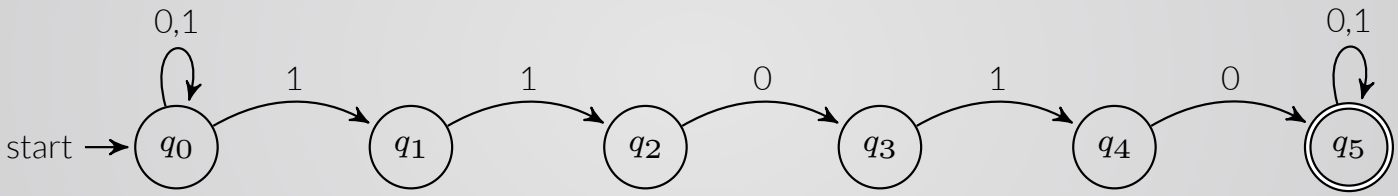
Let's look at the NFA's computation paths, given input string `00101`:



Since one thread accepts, the computation accepts.

NFA Example 2

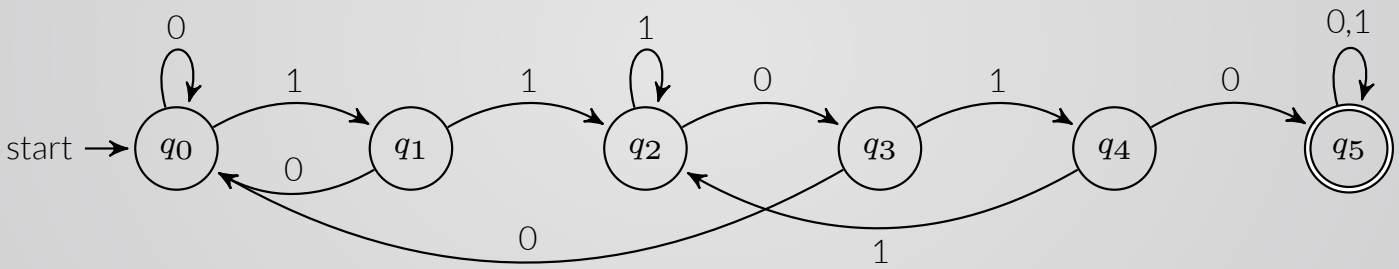
Give an NFA that accepts all strings that contain string `11010`:



Question: What is the role/purpose of the loop transitions in q_0 and q_5? How would the language the machine recognizes change if you omitted either of them?

NFA Example 2 Cont'd

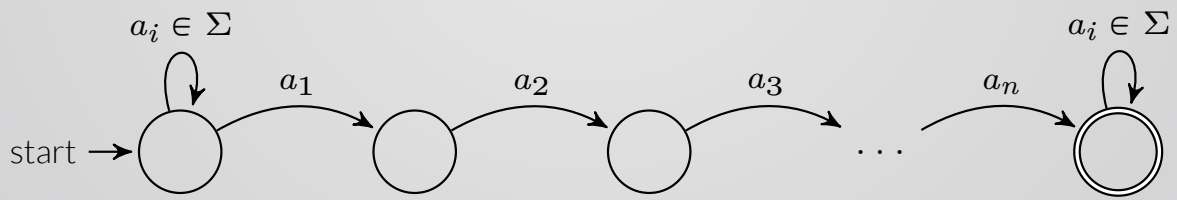
Recall the equivalent DFA:



Notice the NFA was much easier to write (and read).

NFA String Matching

In fact, given any string $s = a_1\, a_2\, a_3 \ldots a_n$ and alphabet Σ, the NFA that accepts all strings containing s is:

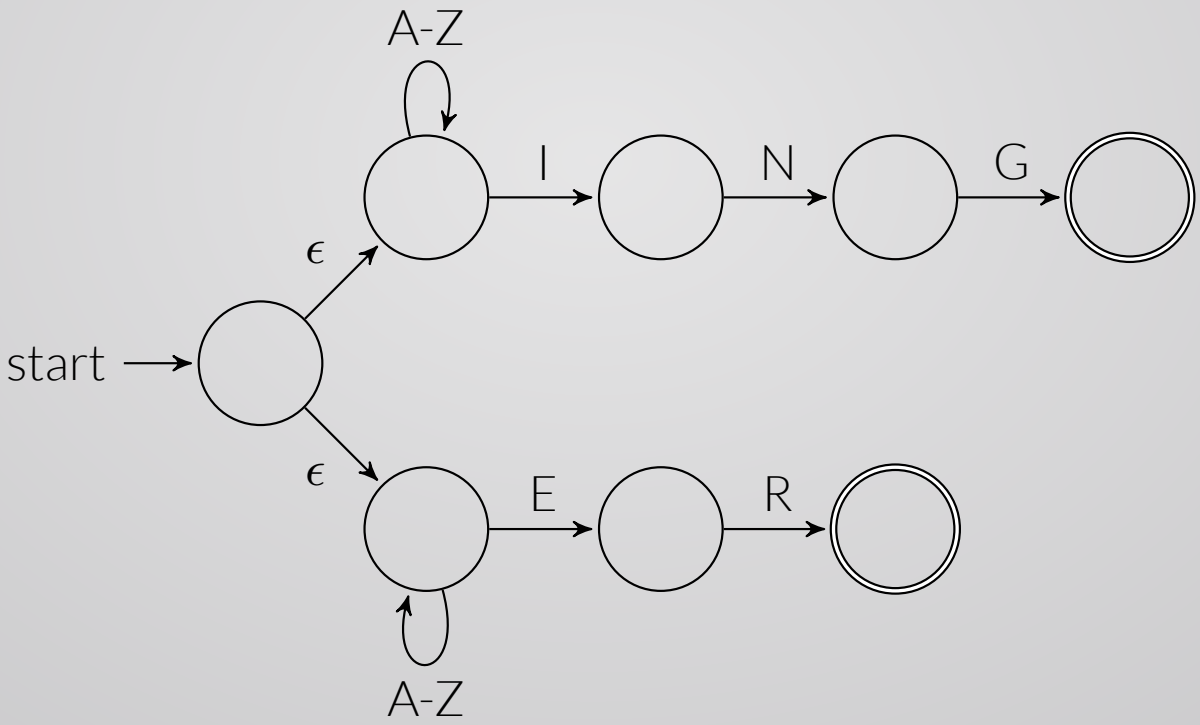


ϵ-Transitions

An NFA can make a transition without receiving any input. These transitions, called ϵ-transitions can sometimes greatly simplify the design.

ϵ-Transitions

Using ϵ-transitions, give an NFA that recognizes the set of string that end in "ING" or "ER" for $\Sigma = \{A \ldots Z\}$:



Non-determinism: Two Interpretations

There are two equivalent ways of thinking about non-determinism:

- **Parallel Computation:** The computation proceeds along all possible paths concurrently
- **Lucky Guess:** The computation proceeds along one path only, but always picks the path that leads to an accepting state (if such a path exists)