Deterministic Finite State Automata (DFA or DFSA)

DFA/DFSA: A DFA is a quintuple $(Q, \Sigma, q_0, F, \delta)$ where Q is a fixed, finite, non-empty set of states. Σ is a fixed (finite, non-empty) alphabet ($Q \cap \Sigma = \{\}$). $q_0 \in Q$ is the initial state. $F \subseteq Q$ is the set of accepting ("final") states. $\delta : Q \times \Sigma \to Q$ is a transition function (i.e., for each $q \in Q$, $a \in \Sigma$, $\delta(q, a)$ is the next state of the DFA when processing symbol a from state q)

Given a state and a single input symbol, a transition function gives a new state. **Extended transition function** $\delta^*(q, s)$ gives new state for DFA after processing string $s \in \Sigma$ starting from state $q \in Q$. It can be defined recursively, as follows:

$$\delta^*(q, s) = \begin{cases} q & \text{if } s = \epsilon \text{ (empty)} \\ \delta(\delta^*(q, s'), a) & \text{if } s = s'a \text{ for some } s' \in \Sigma^* \text{ and } a \in \Sigma \end{cases}$$

Example 1. *Remember our vending machine example from the previous session*

	0	5	10	15	20	25	30+
n	5	10	15	20	25	30+	30+
d	10	15	20	25	30+	30+	30+
q	25	30+	30+	30+	30+	30+	30+

For this machine, we have:

$$\begin{aligned} \delta^*(5, ndn) &= \delta(\delta^*(5, nd), n) = \delta(\delta(\delta^*(5, n), d), n) = \delta(\delta(\delta(\delta^*(5, \epsilon), n), d), n) \\ &= \delta(\delta(\delta(5, n), d), n) = \delta(\delta(10, d), n) = \delta(20, n) = 25 \end{aligned}$$

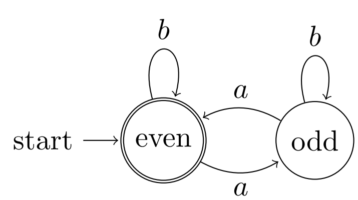
Acceptance: A string $s \in \Sigma^*$ is **accepted** by a DFA A iff $\delta^*(q_0, s) \in F$; otherwise, s is **rejected**. The language accepted by a DFA A is defined as

$$L(A) = \{s \in \Sigma^* : \text{ A accepts } s \text{ (i.e., } \delta^*(q_0, s) \in F)\}$$

Example 2. *Come up with a DFA that accepts* $L = \{s \in \{a, b\}^* : \ s \text{ contains an even number of } a\text{'s}\}$.

In the DFA the states should represent the information about the string processed so far. In this case, only need to remember if number of a's seen so far is even or odd, so only need two states "even" and "odd". Before reading any symbol, the number of a's processed so far is 0, which is even. Hence, the initial state should be even. We want the DFA to accept the strings in L (strings with even number of a). Hence, we should choose even to also represent our accepting state.

To represent transition function, transition diagrams are a useful notation. Each state represented by a node (hallow circle), transitions represented by directed edges labelled with input symbol (i.e., $\delta(q, a) = q'$ represented by edge from q to q' labelled with a). Initial state has an in-edge, and accepting states have double circles for nodes.



Therefore we can describe the following transition function for the DFA represented by the transition diagram.

$$\delta(\text{even}, a) = \text{odd}, \;\; \delta(\text{even}, b) = \text{even}$$
$$\delta(\text{odd}, a) = \text{even}, \;\; \delta(\text{odd}, b) = \text{odd}$$

Let's call the DFA associated with the above transition diagram A. We have to prove that $L(A) = L$.

Proof. In order to prove $L(A) = L$, we just need to prove the following state invarance.

$$\delta^*(\text{even}, s) = \begin{cases} \text{even} & \text{if s contains even number of a's} \\ \text{odd} & \text{if s contains odd number of a's} \end{cases}$$

We will prove this by induction on $|s|$.

Base Case: $\delta^*(\text{even}, \epsilon) = \text{even}$ and ϵ contains an even number of a's (zero is even). Hence, state invariance holds for $s = \epsilon$.

Induction Step: Suppose $n \in \mathbb{N}$ and state invariance holds for all $s \in \Sigma^n$ (IH) –recall that Σ^n is the set of all strings of length n over Σ. We want to show that state invariance holds for all $s \in \Sigma^{n+1}$.

Suppose $s \in \Sigma^{n+1}$. Since $n \geq 0$, $n + 1 \geq 1$ so $s = t \circ c$ for some $t \in \Sigma^n$ and $c \in \Sigma$. Then, by definition, $\delta^*(even, s) = \delta(\delta^*(even, t), c)$. Consider the possible values of $\delta^*(even, t)$.

Case 1: Suppose $\delta^*(\text{even}, t) = \text{even}$. Then, t contains an even number of a's (by the IH, since t has length n). Consider the possible values of c.

Subcase A: Suppose $c = a$. Then $\delta^*(\text{even}, s) = \delta(\text{even}, a) = \text{odd}$, and $s = t \circ a$ contains an odd number of a's (since t contains an even number).

Subcase B: Suppose $c = b$. Then $\delta^*(\text{even}, s) = \delta(\text{even}, b) = \text{even}$, and $s = t \circ b$ contains an even number of a's (same as t). In both subcases, state invariance holds.

Case 2: Suppose $\delta^*(\text{even}, t) = \text{odd}$. Then, t contains an odd number of a's (by the IH, since t has length n). Consider the possible values of c.

Subcase A: Suppose $c = a$. Then $\delta^*(\text{even}, s) = \delta(\text{odd}, a) = \text{even}$, and $s = t \circ a$ contains an even number of a's (since t contains an odd number).

Subcase B: Suppose $c = b$. Then $\delta^*(\text{even}, s) = \delta(\text{odd}, b) = \text{odd}$, and $s = t \circ b$ contains an odd number of a's (same as t). In both subcases, state invariance holds. We can conclude, in both cases, state invariance holds. Hence, by induction, state invariant holds for all strings $s \in \Sigma^*$.

NOTE: The proof has one case for each possible state and one sub-case for each possible input symbol.

From state invariant, we can now conclude:

- If A accepts s, then $\delta^*(even, s) = even$ so by state invariance, s contains an even number of a's, i.e., $s \in L$.
- If A rejects s, then $\delta^*(even, s) = odd$ so by state invariance, s contains an odd number of a's, i.e., $s \notin L$.

Hence, A accepts s iff $s \in L$, i.e., $L(A) = L$. $\square$

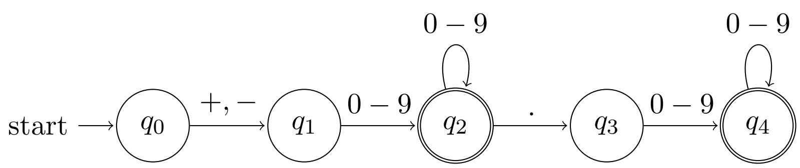
Exercise. *See textbook for other detailed examples.*

To simplify transition diagrams, we will introduce the following additional conventions:

- Combine multiple transitions from one state to another labelled with different input symbols into one edge with a compound label consisting of symbols separated by commas; (e.g., for vending machine's DFA, instead of having three edges from state 25 to state 30 – one for each input symbol n, d, q – have single edge with label "n,d,q")

- **Dead states** (states from which an accepting state can never be reached) are not drawn. Be Careful! With this additional convention, a "missing" transition in a diagram does NOT mean DFA stays in that state: it means DFA goes to dead state and rejects.

Example 3. *DFA to accept floating-point numbers of the form* $+/-n$ *or* $+/-n.m$*, where* n *and* m *are decimal integers (non-empty strings over the digits* $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$*). E.g.,* $+3.0$*,* $+2$*,* -0.01 *are acceptable but* $3.$*,* $-.5$*,* $4.2.3$*,* $--1$ *are not.*



Note that we have used $0-9$ *for* $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ *in the above diagram.*

Consider how the above DFA processes $--1$. The DFA starts in state q_0 and after processing the $-$ sign, it will jump to state q_1. Then, it processes the other $-$ sign; however, there is no transition associated with this input in the diagram. In other words, the DFA has gone to a dead state. Hence the DFA rejects $--1$.

Regular Expressions

Regular expressions describe sets of strings using a small number of basic operators.

Regular Expression: The set of regular expressions (regexps or REs) over alphabet Σ is defined as (with usual convention $\{\} \notin \Sigma$, $\epsilon \notin \Sigma$):

- $\{\}$ (empty set symbol), ϵ (empty string symbol) are regexps

- a is a regexp for all symbols a $\in \Sigma$

- if R and S are regexps over Σ, then so are:

  - $R + S$ (union) – lowest precedence
  - RS (concatenation)
  - R^* (star) – highest precedence

- nothing else is a regexp over Σ

Remark. *This should look familiar: it is a recursive definition of the type we used when we were discussing structural induction.*

$L(R)$**:** For each regexp R, recursively define the language described by R ($L(R)$) as follows:

- $L(\{\}) = \{\}$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\}$ for every symbol a $\in \Sigma$
- If R is a regular expresssion then either $R = (S + T)$, or $R = ST$ or $R = S^*$ for some regular expressions S and T. Then:
  - $L(S + T) = L(S) \cup L(T)$
  - $L(ST) = L(S) \circ L(T)$
  - $L(S^*) = L(S)^{\circledast}$

Remark. *This definition is weaker (more limited) than set of regular expression operators commonly found in programming libraries and UNIX command-line utilities. That's because they are expanded versions with additional operations.*

Remark. *Why do we need regexps when we have operations on languages? The idea is to study what types of languages can be defined with restricted set of operations.*

Example 4. *Examples of regular expressions:*

- $L(a + b) = \{a, b\}$
- $L(ab) = \{ab\}$
- $L((a + b)a) = \{aa, ba\} = L(aa + ba)$
- $L(a^*) = \{\epsilon, a, aa, aaa, \cdots\}$ *(zero or more repetitions of a)*
- $L(aa^*) = \{a, aa, aaa, \cdots\} = L(a^*a)$ *(one or more repetitions of a)*
- $L((ab)^*) = \{\epsilon, ab, abab, ababab, \cdots\}$ *(zero or more repetitions of ab)*
- $L(a^*b^*) = \{\epsilon, a, aa, aaa, \cdots, b, ab, aab, aaab, ..., bb, abb, aabb, \cdots\}$ *(any number of a's followed by any number of b's)*
- $L((a + b)^*) = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, \cdots\}$ *(zero or more repetitions of a's or b's, i.e., every string of a's and b's)*
- $L(a^* + b^*) = \{\epsilon, a, b, aa, bb, aaa, bbb, aaaa, bbbb, \cdots\}$ *(every string consisting entirely of a's or entirely of b's)*
- $L((a + b)(a + b)^*) = \{a, b, aa, ab, ba, bb, \cdots\}$ *(every nonempty string of a's and b's)*
- $L(a(ba + c)^*) = \{a, aba, ac, ababa, abac, acba, acc, \cdots\}$
- *All strings of a's and b's that have the same first and last symbol:* $\epsilon + a + b + a(a + b)^*a + b(a + b)^*b$
- *RE for* $L = \{$*all strings of a's and b's that contain at least one a*$\}$*:* $(a + b)^*a(a + b)^*$ *or* $b^*a(a + b)^*$ *or* $(a + b)^*ab^*$