TDDD14/TDDD85 Lecture 2: Deterministic Finite Automata

Victor Lagerkvist (based on slides by Christer Bäckström and Gustav Nordh)

In this lecture note we define (deterministic) finite automata and exemplify how they can be constructed. We use this class of automata to define the class of *regular* languages.

Background and Intuition

Recall the PIN code example from lecture 1: a simple machine which reads symbols from the alphabet $\{0, 1, \ldots, 9\}$ and recognises the code 1234. We previously stated that the language of this machine was $\{1234\}$, but to make the example slightly more realistic we in addition assume that that the machine accepts a numerical string if it contains 1234 as a subsequence. More formally, this implies that the language of the machine is $\{0, 1, \ldots, 9\}^*\{1234\}\{0, 1, \ldots, 9\}^*$. We now want to construct a machine, an *automaton*, for this language.

Definition 1. *(Informal) A* deterministic finite automaton[1] *(DFA) consists of five components: a set of* states*, an alphabet, a* transition function *which for each state and symbol from the alphabet returns a new state, a* start *(or initial) state, and a set of* accept *states.*

[1] A DFA is said to be deterministic since there for each state and symbol exists exactly one possible transition to a new state. An automaton where we have multiple possible transitions is said to be *nondeterministic* and is the topic of the next lecture.

We visualize a DFA as a directed graph where each node represents a state, and where a directed edge between two nodes represents a transition from one state to another by reading a single symbol from the input string. In addition, the accept state is indicated by encircling the node in question, and the initial state is indicated by an arrow $\longrightarrow$. Informally, a DFA then takes a string (over the current alphabet) as input, reads one symbol from this string at a time, and in each iteration proceeds to a state by using the transition function. If there are no symbols left and the machine ends up in an accept state, then the machine *accepts* the string, and otherwise *rejects*.

How can we then construct a DFA for the language of numerical strings containing 1234 as a subsequence? In general, do not try to solve the entire problem at once, and instead concentrate on a part of the problem, e.g., by solving a relaxed problem and then trying to extend this to a full solution. In the case of this DFA, the language appears to be too complicated for us to immediately see all states and transitions. Hence, we initially try to figure out the most important states and then worry about transitions later. Thus, we definitely need one state when we have read 1, one state where we have read

Two hints for constructing a DFA for a given language.

1. Simplify the problem/language.
2. Begin by defining the most important states: worry about transitions and exactly how the states should be achieved later.

1 followed by 2 (i.e., 12), one state where we have read 123, and an accept state which we end up in after having read 1234. In addition, we need a start state, and a series of transitions between the states, and thus obtain the following.

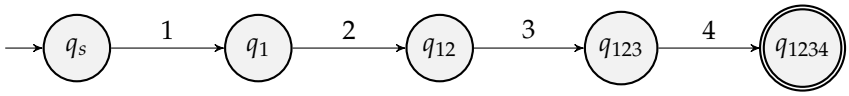

Figure 1: Attempt 1 (incomplete).

Thus, this "automaton"[2] begins in state q_s and transitions to the final state q_{1234} by reading the symbols 1, 2, 3, and 4. The naming scheme for the remaining states should be fairly obvious: q_s is the start state, in q_1 we have read 1, in q_{12} we have read 12, and in q_{123} we have read 123. However, while this automaton correctly accepts the string 1234, we have not yet considered the cases where 1234 is a substring. A first attempt of remedying this might look as follows.

[2] This is not yet a DFA, not even according to the informal definition, since the transition function is not fully specified. But we will worry about that later.

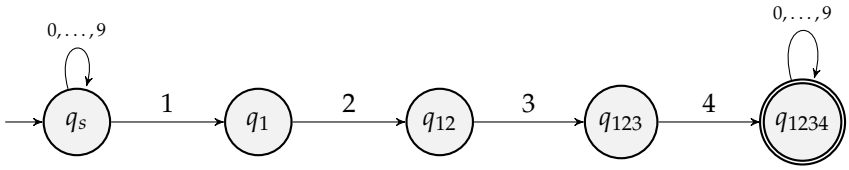

Figure 2: Attempt 2 (incomplete and incorrect).

But now we have introduced an additional error: which state should the automaton transition to if it in state q_s reads the symbol 1: q_s or q_1? This is an example of *nondeterminism* which is not allowed to occur in a DFA. Since this is a rather serious error it is best to fix it immediately. Let us reason as follows. If we in state q_s read 1 then we should transition to q_1. Hence, we need to remove 1 from the loop in q_s, meaning that we stay in q_s only if we read a symbol in $\{0, 2, \ldots, 9\}$. However, when in state q_1 we also need the possibility of going back to q_s if we read a symbol which is *not* 2, or 1, i.e., $0, 3, \ldots, 9$. Furthermore, if we are in state q_1 and read 1, then we should simply stay in q_1. These changes lead to the following automaton.

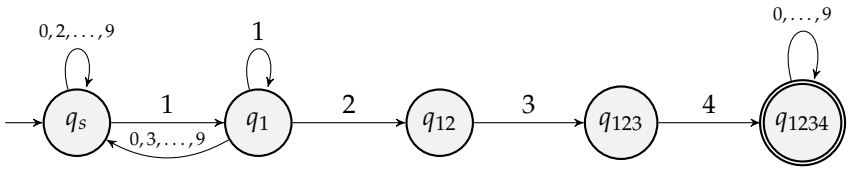

Figure 3: Attempt 3 (incomplete).

This is certainly more promising than earlier attempts, but what happens if we in state q_{12} or state q_{123} read the symbol 1? The most

reasonable course of action is to proceed to q_1, so we need to explicitly add those transitions.

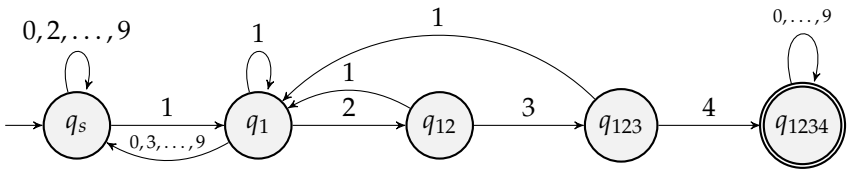

Figure 4: Attempt 4 (incomplete).

Only two uncertainties remain: what happens if we in state q_{12} read a symbol which is not 1 or 3, and if we in state q_{123} read a symbol which is not 1 or 4? Then the string read so far is not 1234, and we have to go back to the initial state q_s. This leads to the following.

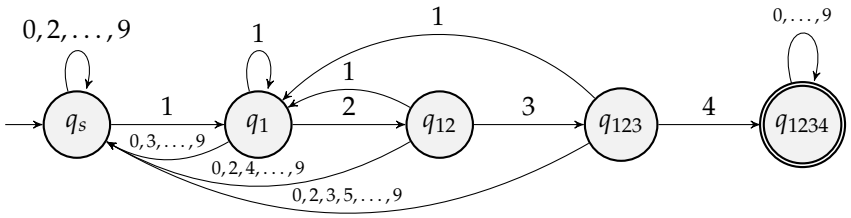

Figure 5: Attempt 5 (complete).

Formal Definition

THE FORMAL DEFINITION of a DFA should now not come as great surprise, given the informal definition in Definition 1 and the construction of the DFA in Figure 5.

Definition 2. *A* deterministic finite automaton *(DFA) is a* 5*-tuple* $(Q, \Sigma, \delta, q_0, F)$ *where*

1. Q *is a finite set called the* states*,*
2. Σ *is an* alphabet*,*
3. $\delta\colon Q \times \Sigma \to Q$ *is the* transition function*,*
4. $q_0 \in Q$ *is the* start state*,*
5. $F \subseteq Q$ *is the set of* accept states*.*

The notation $\delta\colon Q \times \Sigma \to Q$ simply means that δ is a binary (2-arity) function which takes a state from Q and a symbol from Σ as arguments, and returns a state. It is common to represent the

transition function δ as a table where each row consists of a state q, a symbol $x \in \Sigma$, and the new state $\delta(q, x)$. See Figure 6 for an example of a DFA, and Figure 7 for the corresponding transition table.

Example 1. *Let us see how the DFA in Figure 5 can be formally represented as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ via Definition 2.*

1. $Q = \{q_s, q_1, q_{12}, q_{123}, q_{1234}\}$,
2. $\Sigma = \{0, 1, \ldots, 9\}$,
3. $q_0 = q_s$,
4. $F = \{q_{1234}\}$,

and where δ is defined as

- $\delta(q_s, 1) = q_1$, $\delta(q_s, x) = q_s$ *for* $x \in \{0, 2, \ldots, 9\}$,
- $\delta(q_1, 1) = q_1$, $\delta(q_1, 2) = q_{12}$, $\delta(q_1, x) = q_s$ *for* $x \in \{0, 3, \ldots, 9\}$,
- $\delta(q_{12}, 1) = q_1$, $\delta(q_{12}, 3) = q_{123}$, $\delta(q_{12}, x) = q_s$ *for* $x \in \{0, 2, 4, \ldots, 9\}$,
- $\delta(q_{123}, 4) = q_{1234}$, $\delta(q_{123}, 1) = q_1$, $\delta(q_{123}, x) = q_s$ *for* $x \in \{0, 2, 3, 5, \ldots, 9\}$,
- $\delta(q_{1234}, x) = q_{1234}$ *for* $x \in \{0, \ldots, 9\}$.

Definition 3. *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and $s = s_1 s_2 \ldots s_n$ a string over Σ. Then M accepts s if there is a sequence of states $r_0, r_1, \ldots, r_n$ from Q such that*

- $r_0 = q_0$,
- $\delta(r_i, s_{i+1}) = r_{i+1} (i \in \{0, \ldots, n-1\})$, *and*
- $r_n \in F$.

In other words there exists a sequence of state transitions from the initial state to the accept state where each transition is determined by the current symbol from the input string, the current state, and the transition function. It is important to understand and be able to apply Definition 3 since it can be used to verify whether a constructed DFA is actually correct.

Definition 4. *A DFA $M = (Q, \Sigma, \delta, q_0, F)$* recognises *the language $A \subseteq \Sigma^*$ if $A = \{s \mid M \text{ accepts } s\}$, and we let $L(M)$ denote the language recognised by M.*

We now have everything in place to define the important concept of a regular language.

Definition 5. *Let Σ be an alphabet. A language $A \subseteq \Sigma^*$ is said to be* regular *if $L(M) = A$ for some DFA $M = (Q, \Sigma, \delta, q_0, F)$.*

Let us wrap up by showing a concrete example of Definition 3. Consider the DFA in Figure 6 where the transition function δ is defined according to the table in Figure 7, and the string $s = s_1s_2s_3s_4s_4 = 0101$.

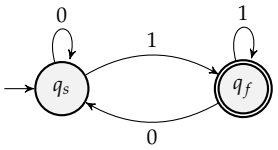

Figure 6: A DFA for the language of Boolean strings ending with 1.

	0	1
q_s	q_s	q_f
q_f	q_s	q_f

Figure 7: The transition table corresponding to the DFA in Figure 6.

Using Definition 3 we can easily prove that the DFA accepts s by iteratively computing the states $r_0, r_1, \ldots, r_4$ as follows (the state transitions are visualised in Figures 8– 12, where the underlined number represents the number currently being inspected, and where the current state is coloured red).

1. $r_0 = q_s$.
2. $r_1 = \delta(r_0, s_1) = \delta(q_s, 0) = q_s$.
3. $r_2 = \delta(r_1, s_2) = \delta(q_s, 1) = q_f$.
4. $r_3 = \delta(r_2, s_3) = \delta(q_f, 0) = q_s$.
5. $r_4 = \delta(r_3, s_4) = \delta(q_s, 1) = q_f$.

Since $r_4 = q_f$ is an accepting state we conclude that the DFA accepts 0101.

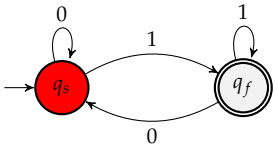

Figure 8: $\underline{0}101$

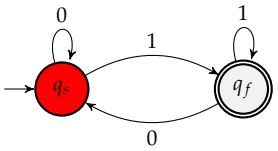

Figure 9: $0\underline{1}01$

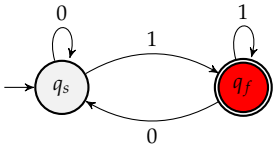

Figure 10: $01\underline{0}1$

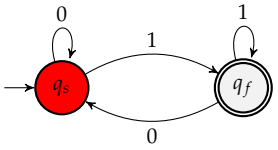

Figure 11: $010\underline{1}$

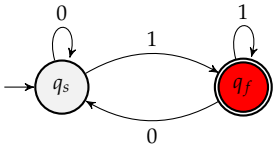

Figure 12: 0101

Summary

We defined deterministic finite automata, leading to the class of regular languages. How much can we generalise a DFA without leaving the realm of regular languages? We investigate this question in the next lecture by defining and exemplifying *nondeterministic* finite automata.

Food for Thought

1. Are there any devices/systems around you that could be modelled as a DFA?
2. Is the device that you are currently reading this document on a DFA?
3. How could you simulate a DFA on an input string in your favourite programming language?
4. We required the transition function δ to be fully specified. Assume that we relax this and allow δ to be a partial function, meaning

that δ is allowed to be undefined for some states and symbols. Is every language accepted by such an automaton accepted by some DFA?