TDDD14/TDDD85 Lecture 3: Nondeterministic Finite Automata

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

In this lecture note we consider an extension of DFA where the machine when reading a symbol has the choice between multiple transitions, so-called *nondeterministic finite automata* (NFA). Despite appearing to be a powerful generalisation of DFA we will see that each such automaton can be converted to a DFA accepting the same language, using the *subset construction method*.

Background and Intuition

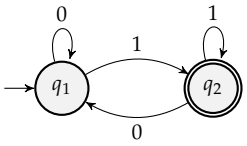

Figure 1: A DFA for the language of Boolean strings ending with 1.

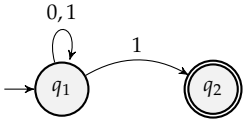

Figure 2: An NFA for the language of Boolean strings ending with 1.

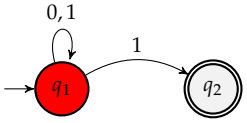

Figure 3: $\underline{0}11$ (the machine is in its initial state).

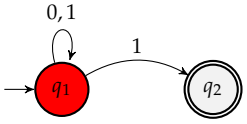

Figure 4: $0\underline{1}1$ (the machine stays in its initial state).

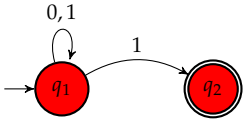

Figure 5: $01\underline{1}$ (the machine may be in both q_1 and q_2).

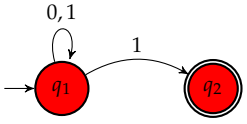

Figure 6: 011 (the machine may be in both q_1 and q_2).

Recall that a DFA is *deterministic*: for each state and symbol there exists precisely one transition to a state. However, in general, might it be possible to have a choice between two or more states by allowing *nondeterministic* transitions? Consider e.g. the automaton in Figure 2, which accepts the same language as the DFA in Figure 1. What happens if the machine in the initial state q_1 reads 1: does it stay in q_1 or proceed to the accept state q_2?

In this lecture we explore this concept by considering a generalisation of DFA making use of nondeterminism, so-called *nondeterministic finite automata* (NFA). Loosely speaking, an NFA is an automaton which when reading a symbol has the possibility of choosing several possible states.

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

The nondeterministic component of an NFA may be understood in two different ways. First, we may view an NFA as an automaton which is allowed to *choose* one of many possible transitions, and different choices can lead to different outcomes. The machine accepts a string if there is *some* sequence of choices leading to a final state, and rejects a string if *every* choice is wrong. The second viewpoint is more pragmatic: each transition may lead to multiple new states which the machine explores in parallel. Thus, the NFA reads the input string symbol by symbol, stores a set of "active" states, and in each iteration applies the transition function to every active state. For example, we may understand the NFA in Figure 2 as follows. If we are in state q_1 and read 1 then we have the choice of transitioning

to either q_1 or q_2. Thus, the machine can stay in q_1 for as long as it wants, repeatedly reading 0 and 1, but once it has transitioned to q_2 by reading 1 it cannot go back to q_1. Hence, the language accepted by this machine is the set of all Boolean strings ending with 1, i.e., the language $\{x1 \mid x \in \{0,1\}^*\}$. Here, it might be instructive to compare this NFA with the corresponding DFA in Figure 1. The behaviour of this NFA on the input string 011 is visualised in Figure 3– 6, where each active state is coloured in red. The machine accepts the string in Figure 6 since q_2 is a final state.

Let us consider one additional example before turning to the formal definition of an NFA. The NFA in Figure 7 is an extension of the NFA in Figure 2 which accepts the set of Boolean strings where the second-to-last symbol is 1, i.e., the set $\{x1y \mid x \in \{0,1\}^*, y \in \{0,1\}\}$. The behaviour of this NFA on input string 0101 is visualised in Figure 8– 12, where active state(s) and the chosen transition(s) are coloured in red. The machine rejects 0101 since no active state in Figure 12 is a final state.

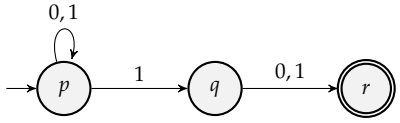

Figure 7: An NFA for $\{x1y \mid x \in \{0,1\}^*, y \in \{0,1\}\}$.

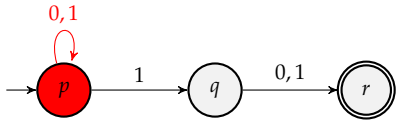

Figure 8: $\underline{0}101$

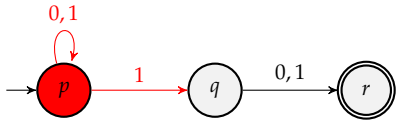

Figure 9: $0\underline{1}01$

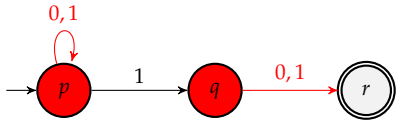

Figure 10: $01\underline{0}1$

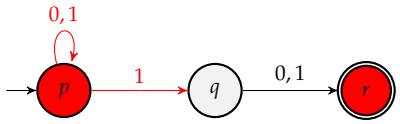

Figure 11: $010\underline{1}$

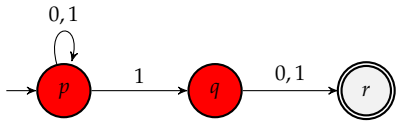

Figure 12: 0101 (rejected)

Formal Definition

Recall that $\mathcal{P}(A) = \{X \mid X \subseteq A\}$ denotes the powerset of a set A, i.e., the set of all subsets of A. For example, $\mathcal{P}(\emptyset) = \{\emptyset\}$, $\mathcal{P}(\{q\}) = \{\emptyset, \{q\}\}$, $\mathcal{P}(\{q_1, q_2\}) = \{\emptyset, \{q_1\}, \{q_2\}, \{q_1, q_2\}\}$. The formal definition of an NFA is now surprisingly undramatic, but we will make one additional generalisation where we allow transitions without reading a symbol from the input string, so-called *ε-transitions*.

Definition 2. *A* nondeterministic finite automaton *(NFA) 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. $\delta \colon Q \times \Sigma \cup \{\varepsilon\} \to \mathcal{P}(Q)$ *is the* transition function.
4. $q_0 \in Q$ *is the* start state,
5. $F \subseteq Q$ *is the set of* accept states.

An NFA with ε-transitions is sometimes called an *ε-NFA*. Note that the only difference to a DFA lies in the transition function δ which (1) allows the NFA to jump from one state to another without reading a symbol from the input string, if the two states have an ε-transition between them, and (2) returns a set of potential states instead of a single state.

Example 1. *Let us see how the NFA in Figure 7 can be formally represented as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ via Definition 2.*

1. $Q = \{p, q, r\}$,
2. $\Sigma = \{0, 1\}$,
3. $q_0 = p$,
4. $F = \{r\}$,

and where δ is defined as

- $\delta(p, 0) = \{p\}$, $\delta(p, 1) = \{p, q\}$,
- $\delta(q, 0) = \delta(q, 1) = \{r\}$,
- $\delta(r, 0) = \delta(r, 1) = \emptyset$.

Let us immediately consider an example showing the usefulness of ε-transitions.

Example 2. *Assume that we now want to define an NFA for the language of Boolean strings which (1) ends with* 1 or *where (2) the second symbol to the right is* 1. *Hence, we want to construct an NFA for the language $\{x1 \mid x \in \{0,1\}^*\} \cup \{x1y \mid x \in \{0,1\}^*, y \in \{0,1\}\}$. Recall that we already have a DFA recognising $\{x1 \mid x \in \{0,1\}^*\}$ (Figure 1) and an NFA recognising $\{x1y \mid x \in \{0,1\}^*, y \in \{0,1\}\}$ (Figure 7). Can we somehow use these automata to avoid having to construct an NFA from scratch? Here, ε-transitions prove to be very useful indeed. Consider the NFA in Figure 13 which is constructed by taking the two aforementioned NFAs and creating a new start state from which we have ε-transitions to the original two machines. Then the machine starts in the new start state but may use ε-transitions to proceed to any of the original machines. The behaviour of this machine on the input string* 01 *is visualised in Figure 15– 18 in the appendix.*

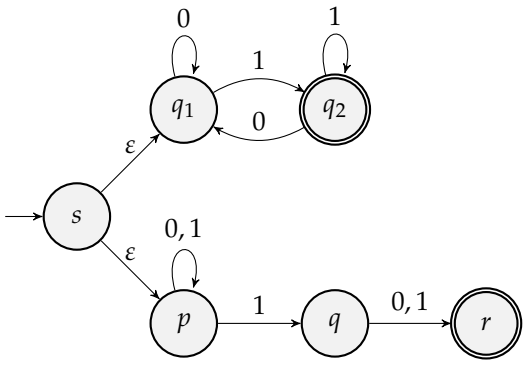

Figure 13: An NFA with ε-transitions recognising $\{x1 \mid x \in \{0,1\}^*\} \cup \{x1y \mid x \in \{0,1\}^*, y \in \{0,1\}\}$.

Last, we generalise the notion of "acceptance" in the context of NFAs.

Definition 3. *Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and s be a string of length n over Σ. Say that N* accepts *s if there is a sequence of states $r_0, r_1, \ldots, r_m$, $m \geq n$, from Q such that s can be written as $s = s_1 s_2 \ldots s_m$ where each $s_i \in \Sigma \cup \{\varepsilon\}$ such that*

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

This might look slightly awkward, but the condition $s = s_1 s_2 \ldots s_m$ for $s_i \in \Sigma \cup \{\varepsilon\}$ means that we are allowed to "insert" the empty string ε in s when we want to use an ε-transition.

Thus, an NFA accepts a string if there *exists* a sequence of transitions from the start state to an accepting state, and rejects if *no* such sequence of transitions exists.

Example 3. *Let us consider the NFA in Figure 7 and the string* 0101. *Proving that this string is* not *accepted is more difficult than in the DFA case since we have to rule out every possible sequence of state transitions. Consider the following applications of δ and compare them with Figure 8–12.*

1. $\delta(p, 0) = \{p\}$[1].
2. $\delta(p, 1) = \{p, q\}$[2].
   (a) $\delta(p, 0) = \{p\}$[3].
   (b) $\delta(p, 1) = \{p, q\}$[4].
   (a) $\delta(q, 0) = \{r\}$[5].
   (b) $\delta(r, 1) = \emptyset$[6].

[1] Read 0, in state p: stay in p.
[2] Read 1 in state p: go to p or q.
[3] Read 0 in state p: go to p.
[4] Read 1 in state p: go to p or q.
[5] Read 0, in state q: go to r.
[6] Read 1, in state r: no transition.

The above applications show that there does not exist a sequence of states $r_0 = p, r_1, r_2, r_3, r_4 = r$ where $r_1 \in \delta(r_0, 0)$, $r_2 \in \delta(r_1, 1)$, $r_3 \in \delta(r_2, 0)$, $r_4 \in \delta(r_3, 1)$. Hence, we conclude that the NFA does not accept the string 0101.

Similarly to DFAs we then let $L(N)$ be the set of strings accepted by an NFA N. We will shortly see that there for each NFA N exists a DFA D such that $L(N) = L(D)$.

The Subset Construction

We now describe a method for converting an NFA to an *equivalent* DFA accepting the same language. For an NFA $N = (Q, \Sigma, \delta, q_0, F)$ and $R \subseteq Q$ let $E(R)$, the *ε-closure of R*, be the set of states that can be reached from R using 0 or more ε-transitions. For example, if we take the NFA in Figure 13 then $E(\{s\}) = \{s, q_1, q_2\}$, $E(\{q_1\}) = \{q_1\}$, and $E(\{s, r\}) = \{s, q_1, p, r\}$.

Theorem 1. *For each NFA N there exists a DFA D such that $L(N) = L(D)$.*

Proof. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We will construct a DFA $D = (Q', \Sigma, \delta', q_0', F')$ where

1. $Q' = \mathcal{P}(Q)$,
2. $q_0' = E(\{q_0\})$, and
3. $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$.

In other words: in the new set of states each state is a set of states over Q, the start state is formed by taking the start state q_0 and all states that can be reached by ε-transitions, and the set of accepting states contains every state which contains at least one accepting state from F.

The transition function δ' is then for each $R \in Q'$ and $a \in \Sigma$ defined by $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\}$. □

In other words: $\delta'(R, a)$ contains the states obtained by picking a state $r \in R$, following the transition from r as prescribed by the symbol a, and then following 0 or more ε-transitions.

For correctness and additional details, see Theorem 1.39 in Sipser [7]. The proof of Theorem 1 suggests an algorithm for converting an NFA to a DFA by iteratively computing the new transition function δ'. We illustrate this method, the *subset construction*, by an example.

[7] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition, 2013

Example 4. *We will use the subset construction from the proof of Theorem 1 to convert the NFA $N = (Q, \Sigma, \delta, q_0, F)$ in Figure 13 to an equivalent DFA $(\mathcal{P}(Q), \Sigma, \delta', q_0', F')$. From the figure we see that $Q = \{s, q_1, q_2, p, q, r\}$, $\Sigma = \{0, 1\}$, $q_0 = s$, and $F = \{q_2, r\}$.*

We could in principle do this by computing the powerset $\mathcal{P}(Q)$ of Q and for each $R \subseteq Q$ and $a \in \Sigma$ compute the corresponding entry in δ', but this is cumbersome since even in this small example we have that $|\mathcal{P}(Q)| = 2^6 = 64$, which is hard to compute by hand. Can we do better?

In practice, it is often much simpler to iteratively compute δ' by only computing new entries when they are needed. Thus, we begin by computing the new start state $q_0' = E(\{s\}) = \{s, q_1, p\}$. The interpretation of this state is then that we initially are in either state s, state q_1, or state p, depending on whether we have done an ε-transition or not. We then want to compute $\delta'(\{s, q_1, p\}, 0)$ and $\delta'(\{s, q_1, p\}, 1)$ in order to complete this row in the transition table of δ'. According to the proof of Theorem 1 we know that $\delta'(\{s, q_1, p\}, 0)$ contains those state reachable from s, q_1, or p by (optionally) first following an ε-transition, and then following a transition by reading 0. Using Figure 13 it is easy to verify that we can only reach q_1 and p in this way. Similarly, if we are in s, q_1, or p, and read 1, then the only possible states are q_2, p, and q. Hence, we obtain the following row in the

transition table of δ', which we mark with $\rightarrow$ to indicate that its the starting state.

	0	1
$\rightarrow \{s, q_1, p\}$	$\{q_1, p\}$	$\{q_2, p, q\}$

Since there are no entries for $\{q_1, p\}$ and $\{q_2, p, q\}$ we continue with those two entries, starting with $\{q_1, p\}$.

	0	1
$\rightarrow \{s, q_1, p\}$	$\{q_1, p\}$	$\{q_2, p, q\}$
$\{q_1, p\}$	$\{q_1, p\}$	$\{q_2, p, q\}$

$\delta'(\{q_1, p\}, 0) = \{q_1, p\}$ and $\delta'(\{q_1, p\}, 1) = \{q_2, p, q\}$ were obtained in exactly the same way as earlier: if we are in state q_1 or state p and read 0, then the only possible states are q_1 and p, meaning that we stay in $\{q_1, p\}$. Similarly, if we are in state q_1 or p and read 1, then we end up in q_2, p, or q, explaining $\delta'(\{q_1, p\}, 1) = \{q_2, p, q\}$. We continue with the entry for $\{q_2, p, q\}$, and since this state is an accepting state we mark the corresponding row with F.

	0	1
$\rightarrow \{s, q_1, p\}$	$\{q_1, p\}$	$\{q_2, p, q\}$
$\{q_1, p\}$	$\{q_1, p\}$	$\{q_2, p, q\}$
$F\{q_2, p, q\}$	$\{q_1, p, r\}$	$\{q_2, p, q, r\}$

We compute $\delta'(\{q_2, p, q\}, 0) = \{q_1, p, r\}$ and $\delta'(\{q_2, p, q\}, 1) = \{q_2, p, q, r\}$, and continue with the entry for $\{q_1, p, r\}$, which is marked as an accepting state since it contains r.

	0	1
$\rightarrow \{s, q_1, p\}$	$\{q_1, p\}$	$\{q_2, p, q\}$
$\{q_1, p\}$	$\{q_1, p\}$	$\{q_2, p, q\}$
$F\{q_2, p, q\}$	$\{q_1, p, r\}$	$\{q_2, p, q, r\}$
$F\{q_1, p, r\}$	$\{q_1, p\}$	$\{q_2, p, q\}$

We compute $\delta'(\{q_1, p, r\}, 0) = \{q_1, p\}$ and $\delta'(\{q_1, p, r\}, 1) = \{q_2, p, q\}$, and since both of these entries are already in the current transition table we pick the only remaining state $\{q_2, p, q, r\}$, mark it as an accept state, and complete the table.

	0	1
$\rightarrow \{s, q_1, p\}$	$\{q_1, p\}$	$\{q_2, p, q\}$
$\{q_1, p\}$	$\{q_1, p\}$	$\{q_2, p, q\}$
$F\{q_2, p, q\}$	$\{q_1, p, r\}$	$\{q_2, p, q, r\}$
$F\{q_1, p, r\}$	$\{q_1, p\}$	$\{q_2, p, q\}$
$F\{q_2, p, q, r\}$	$\{q_1, p, r\}$	$\{q_2, p, q, r\}$

This DFA is visualized in Figure 14.

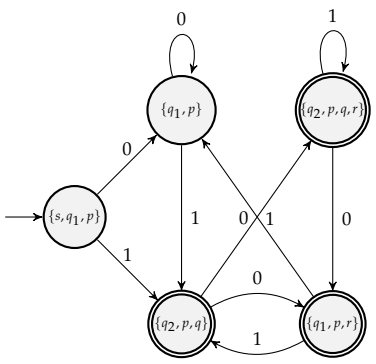

Figure 14: The DFA corresponding to the NFA in Figure 13 constructed via the subset method.

Summary

We defined nondeterministic finite automata: a seemingly powerful generalisation of DFA but which turned out to be yet another characterization of regular languages since each NFA can be converted into an equivalent DFA with the subset construction method. In the forthcoming lecture we will describe yet another characterization of regular languages using *regular expressions*, and investigate closure properties of regular languages.

Food for Thought

1. We described the subset method for constructing an equivalent DFA for a given NFA. In the worst-case, how many states does the resulting DFA have if the given NFA has k states?
2. Assume that you are given an NFA $N = (Q, \Sigma, \delta, q_0, F)$. Can you think of a simple condition (involving δ) for when N is a DFA in disguise, i.e., each transition is deterministic?
3. How could you simulate an NFA on an input string in your favourite programming language? How would such a simulation differ from simulating a DFA?

References

M. Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition, 2013.

Appendix

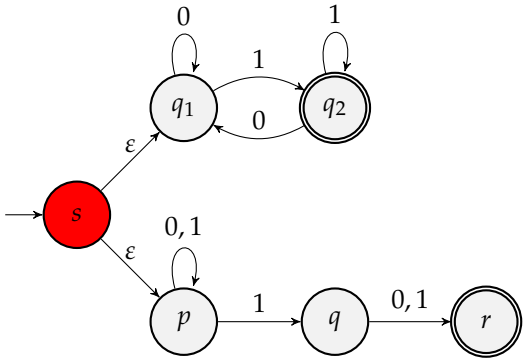

Figure 15: $\underline{0}1$ (the NFA begins in its initial state s).

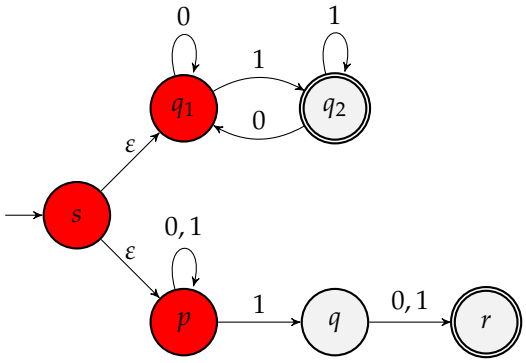

Figure 16: $\underline{0}1$ (before reading 0 the NFA is allowed to use ε-transitions to q_1 and p).

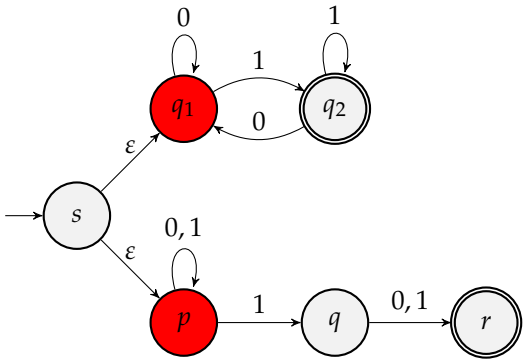

Figure 17: $0\underline{1}$ (the NFA has read 0 and goes from q_1 to q_1 and from p to p, but is no longer in s since there is no transition to s from another state by 0).

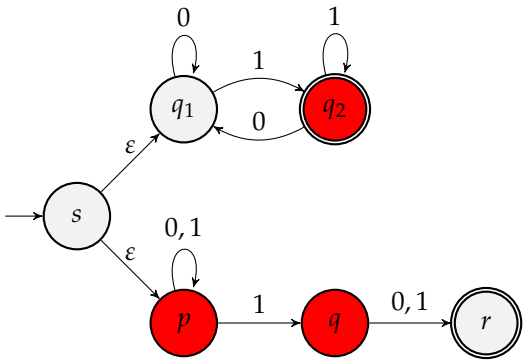

Figure 18: 01 (the NFA has read 1, proceeds from q_1 to q_2, from p to q, and accepts since q_2 is an accepting state).