



# Formal Definition of a Nondeterministic Finite Automaton

Teodor Rus

`rus@cs.uiowa.edu`

The University of Iowa, Department of Computer Science

# Review

- The formal definition of an NFA is similar to that of a DFA. Both have states, an alphabet, transition function, one start state, and a set of accept states
- They differ in one essential way, the transition function:
  1. In a DFA the transition function is defined by:  $\delta : Q \times \Sigma \rightarrow Q$ ;
  2. In an NFA the transition function is defined by:  
$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$$
where  $\mathcal{P}(Q)$  is the collection of subsets of  $Q$  called the power set of  $Q$ .

# Notation

For any alphabet  $\Sigma$ ,  $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

# NFA formal definition (Def. 1.37)

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

1.  $Q$  is a finite set of states
2.  $\Sigma$  is a finite alphabet
3.  $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$  is the transition function
4.  $q_0 \in Q$  is the start state
5.  $F \subseteq Q$  is the set of accept states

# Example NFA (Example 1.38)

Recall the NFA  $N_1$ , Figure 1:

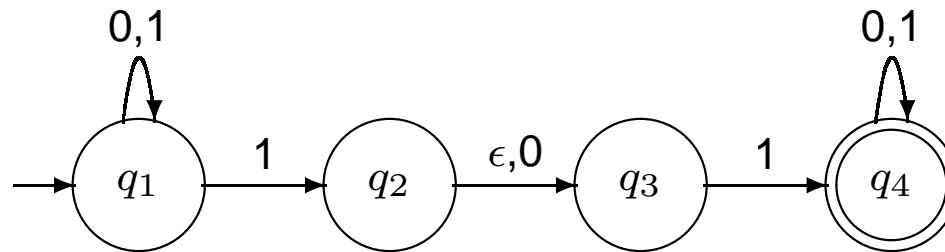


Figure 1: Nondeterministic Finite Automaton  $N_1$

# The formal definition of $N_1$

$N_1 = (Q, \Sigma, \delta, q_1, F)$  where:

1.  $Q = \{q_1, q_2, q_3, q_4\}$
2.  $\Sigma = \{0, 1\}$
3.  $\delta$  is given in the table:

$\delta$	0	1	$\epsilon$
$q_1$	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
$q_2$	$\{q_3\}$	$\emptyset$	$\{q_3\}$
$q_3$	$\emptyset$	$\{q_4\}$	$\emptyset$
$q_4$	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4.  $q_1$  is the start set;
5.  $F = \{q_4\}$

# Computation performed by an NFA

Let  $N = (Q, \Sigma, \delta, q_0, F)$  be an NFA and  $w$  a string over  $\Sigma$ . We say that  $N$  accepts  $w$  if we can write  $w$  as  $w' = y_1 y_2 \dots y_m$ ,  $y_i \in \Sigma_\epsilon$ ,  $1 \leq i \leq m$ , and a sequence of states  $r_0, r_1, \dots, r_m$  exists in  $Q$  such that:

1.  $r_0 = q_0$
2.  $r_{i+1} \in \delta(r_i, y_{i+1})$ , for  $i = 0, 1, \dots, m - 1$
3.  $r_m \in F$

**Note:** the difference between  $w$  and  $w'$ : allow  $\epsilon$  to take positions in  $w$ , that is, **allow  $N$  to perform transitions without reading  $w$ .**

# Interpretation

Condition 1 says that the machine starts its computation in the start state.

Condition 2 says that state  $r_{i+1}$  is one of the allowable new states when  $N$  is in state  $r_i$  and reads  $y_{i+1}$ ; **Note:**  $\delta(r_i, y_{i+1})$  is a set and  $y_{i+1}$  can be  $\epsilon$ .

Condition 3 says that the machine accepts the input if the last state is in the set of accept states.

# An important property

*Any NFA  $N$  can be converted to an equivalent NFA  $N'$  that has a single accept state*

**Proof:** by construction. Let  $N = (Q, \Sigma, \delta, q_0, F)$  be any NFA. Then  $N' = (Q', \Sigma', \delta', q'_0, F')$  where:

1.  $Q' = Q \cup \{q_a\}$  where  $q_a$  is a new state;
2.  $\Sigma' = \Sigma$ ;
3.  $q'_0 = q_0$ ; and 5.  $F' = \{q_a\}$ ;
4.  $\delta'(q_a, a) = \emptyset$  for each  $a \in \Sigma$  (i.e., **no transitions from  $q_a$** ) and:

$$\delta'(q, a) = \begin{cases} \delta(q, a), & \text{if } a \neq \epsilon \text{ or } q \notin F; \\ \delta(q, a) \cup \{q_a\}, & \text{if } a = \epsilon \text{ and } q \in F. \end{cases}$$

# Equivalence of NFA and DFA

- DFAs and NFAs recognize the same class of languages;
- This equivalence is both surprising and useful:
  1. It is surprising because NFAs appears to have more power than DFA, so we might expect that NFA recognizes more languages;
  2. It is useful because describing an NFA for a given language sometimes is much easier than describing a DFA.

# Theorem 1.39

Every nondeterministic finite automaton (NFA) has an equivalent deterministic finite automaton (DFA).

**Proof idea:** Convert NFA recognizing a language into a DFA that recognizes the same language by simulating the NFA.

# More on proof idea

The approach is to pretend to be a DFA simulating an NFA. Questions to be answered:

- How would you simulate an NFA if you were pretending to be a DFA?
- What does one need to keep track of as the input is processed?

# Remember

1. In the examples of NFA computation we kept track of various branches of the computation by **placing tokens on each state that could be activated** at given points in the input;
2. Tokens were updated by moving, adding, and removing them according to the way NFA operates;
3. In the tree computation algorithm this was done by **systematically generating a family of computation trees on levels** as NFA consumes the input.

# Conclusion

all we needed to keep track of was:

- **the set of active states**  
i.e., with tokens on them, or, equivalently;
- **set of active sub-tree roots**  
as the construction of computation tree proceeds.

# Observations

- An NFA with  $k$  states has  $2^k$  subsets of states;
- Each subset correspond to one of the possibilities that DFA must remember. So, the DFA simulating an NFA with  $k$  states has  $2^k$  states;
- The only thing we need now is to figure out which are the start state, the accept states, and transition function.

# The formal proof

Let  $N = (Q, \Sigma, \delta, q_0, F)$  be the NFA recognizing the language  $A$ . We construct the DFA  $M$  recognizing  $A$ . Before doing the full construction, consider first the easier case when  $N$  has no  $\epsilon$  transitions.

# Constructing $M = (Q', \Sigma, \delta', q'_0, F')$

1.  $Q' = \mathcal{P}(Q)$ ; every state of  $M$  is a set of states of  $N$ ;
2. For  $R \in Q'$  and  $a \in \Sigma$  let
$$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$$
**Formally:**  $\delta'(R, a) = \cup_{r \in R} \delta(r, a)$ ;
3.  $q'_0 = \{q_0\}$ ;  $M$  starts computing in the same state as  $N$ ;
4.  $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$   
that is, the machine  $M$  accepts if one of the possible states that  $N$  could be in at this point is an accept state.

# Consider the $\epsilon$ transitions

**Notation:** for any  $R \in Q'$  define  $E(R)$  to be the collection of states that can be reached from  $R$  by going only along  $\epsilon$  transitions, including the members of  $R$  themselves.

**Formally:**

$$E(R) = R \cup \{q \in Q \mid \exists r \in R \wedge \delta(r, \epsilon) = q\}.$$

**Note:** any  $q \in E(R)$  can be reached from  $R$  by traveling along 0 or more  $\epsilon$  arrows.

# Modifications

- Modify  $\delta'$  to place additional tokens on all states that can be reached by going along  $\epsilon$  arrows after every step.
- That is, replace  $\delta(r, a)$  by  $E(\delta(r, a))$  in definition of  $\delta'$ , i.e.,  
$$\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\}$$
- Modify the start state of  $M$  to put tokens initially on all states that can be reached from  $q_0$  along  $\epsilon$  transitions, i.e.,  $q'_0 = E(\{q_0\})$ .

# Facts

1.  $M$  thus constructed obviously simulates  $N$ ;
2. At every step in its computation  $M$  clearly enters a state that corresponds to the subset of states that  $N$  could be in at that point.

# Conclusion

Since every NFA can be converted into a DFA by the procedure discussed above, NFA provides an alternative way of characterizing regular sets.

# Corollary 1.40

A language is regular iff some NFA recognizes it.

**Proof:**

- **IF:** if a language  $A$  is recognized by an NFA then  $A$  is recognized by the DFA equivalent, hence,  $A$  is regular.
- **Only IF:** if a language  $A$  is regular, it means that it is recognized by a DFA. But any DFA is also an NFA hence, the language is recognized by an NFA.

# Example conversion NFA to DFA

Consider the NFA  $N_4$  described in Figure 2:

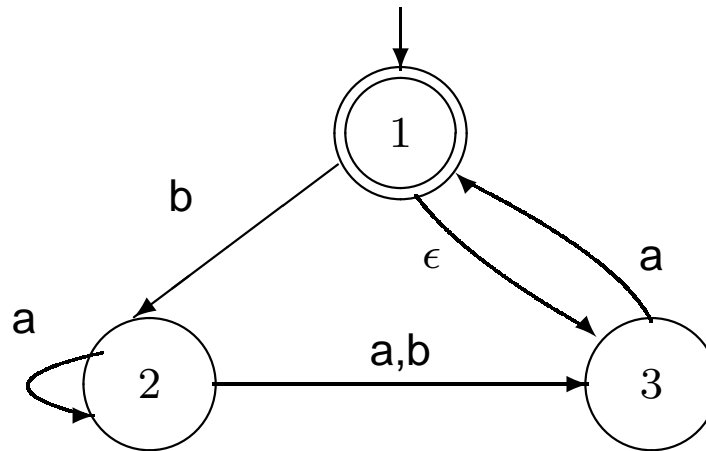


Figure 2: The NFA  $N_4$

# Formal description of $N_4$

$N_4 = (\{1, 2, 3\}, \{a, b\}, \delta, 1, \{1\})$  where  $\delta$  is given in the table:

$\delta$	a	b	$\epsilon$
1	$\emptyset$	$\{2\}$	$\{3\}$
2	$\{2, 3\}$	$\{3\}$	$\emptyset$
3	$\{1\}$	$\emptyset$	$\emptyset$

# Construction of $D_4 = (Q', \{a, b\}, \delta', q'_0, F')$

- Since  $N_4$ 's set of states is  $Q = \{1, 2, 3\}$   
 $Q' = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$
- The start state of  $Q'$ : the set of states reachable from 1 (the start state of  $N_4$ ) traveling by  $\epsilon$ :  $E(\{1\}) = \{1, 3\}$
- The accept state of  $Q'$ : are those states of  $Q'$  that contain the accept states of  $Q$ : i.e.,  
 $F' = \{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$

# Transition function

$\delta' : \mathcal{P}(Q) \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ , where:

$\forall q' \in R \subseteq \mathcal{P}(Q)$  and  $x \in \Sigma$  we have:

$\delta'(q', x) \in R' \subseteq \mathcal{P}(Q)$  where  $R' = \cup_{r \in R} E(\delta(r, a))$ ;

# Constructing $D_4$ 's transitions

- State  $\emptyset$ : goes to  $\emptyset$  on both  $a, b$ , i.e.,  
 $\delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset$
- State  $\{1\}$ : goes to  $\emptyset$  on  $a$  and to  $\{2\}$  on  $b$
- State  $\{2\}$ : goes to  $\{2, 3\}$  on  $a$  and to  $\{3\}$  on  $b$
- State  $\{1, 2\}$ : goes to  $\{2, 3\}$  on  $a$  and to  $\{2, 3\}$  on  $b$
- And so on, for each state

# Putting all together

Figure 3 shows the resulting DFA  $D_4$  equivalent to NFA  $N_4$

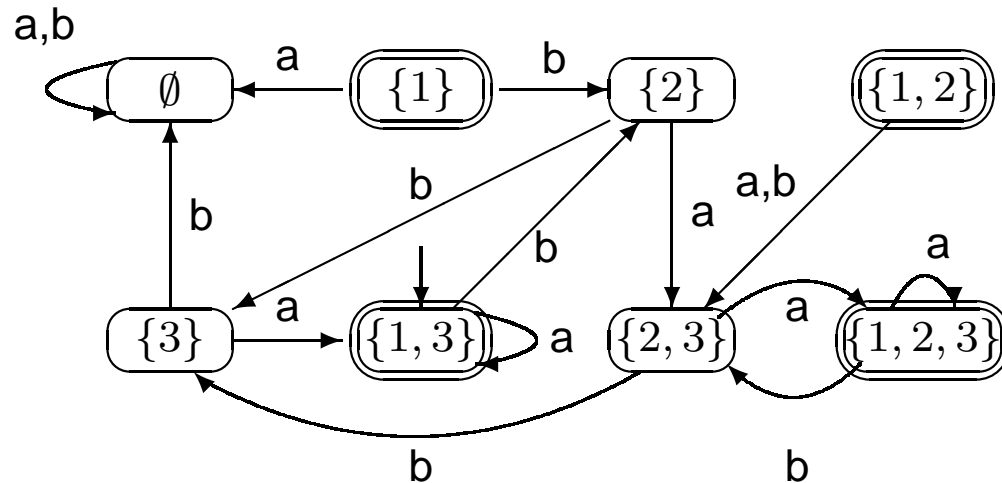


Figure 3: DFA  $D_4$  equivalent to NFA  $N_4$

# Simplifying $D_4$

The automaton  $D_4$  may be simplified by observing that no arrows point at the states  $\{1\}$  and  $\{1, 2\}$ . Removing these states we obtain the automaton in Figure 4

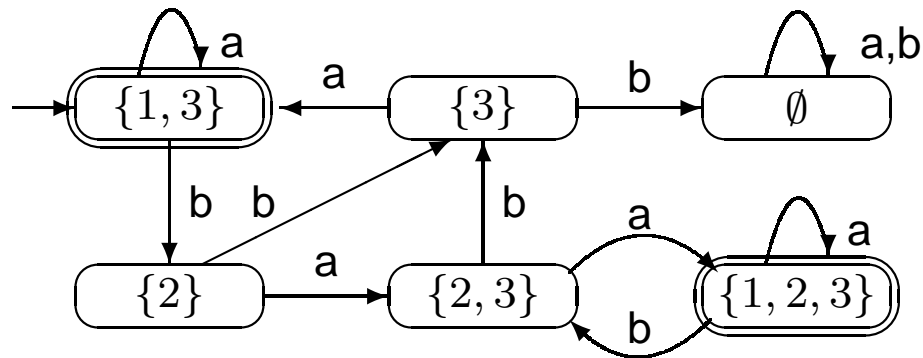


Figure 4: The simplified version of  $D_4$