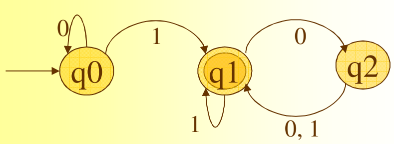
CHAPTER 1
Regular Languages

Contents

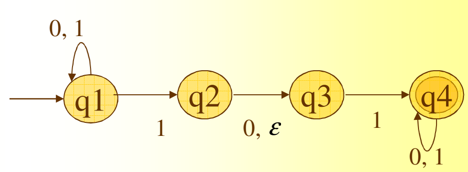
- Finite Automata (FA or DFA)
    - definitions, examples, designing, regular operations
- **Non-deterministic Finite Automata (NFA)**
    - **definitions, equivalence of NFAs and DFAs, closure under regular operations**
- Regular expressions
    - definitions, equivalence with finite automata
- Non-regular Languages
    - the pumping lemma for regular languages

Non-determinism

- So far in our discussion, every step of a computations follows in a unique way from the preceding step.
    - When the machine is in a given state and reads the next input symbol, we know what the next state will be – it is determined. We call this **deterministic** computation.
- In a **non-deterministic** machine, several choices may exist for the next state at any point.



A deterministic FA (**DFA**) M_1

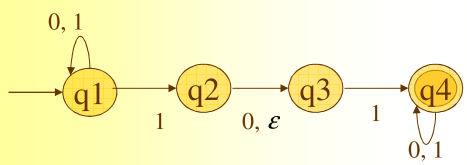


A non-deterministic FA (**NFA**) N_1

- Non-determinism is a generalization of determinism, so every DFA is automatically a NFA.
- The difference: - every state of a DFA has exactly one exiting arrow for each symbol
    - in a NFA a state may have 0, 1, or many exiting arrows for each symbol
    - a NFA may have arrows with the label ε

How does an NFA compute?

- After reading the symbol, the machine splits into multiple copies of itself and follows all the possibilities in parallel.
- Each copy takes one of the possible ways to proceed and continues as before.
- If there are subsequent choices, the machine splits again.
- If the next input symbol doesn't appear on any of the arrows exiting the state occupied by a copy of the machine, that copy dies.
- If any one of these copies of the machine is in an accept state at the end of the input, the NFA accepts the input string.
- If a state with an ε symbol on an exiting arrow is encountered, the machine (w/o reading any input) splits into multiple copies, one following each of the exiting arrow with ε and one staying at current state.

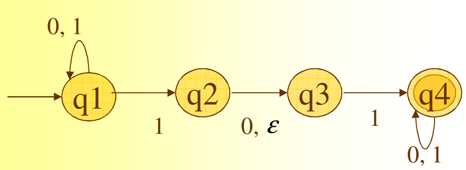


Non-determinism may be viewed as a kind of parallel computation wherein several "processes" can be running concurrently.

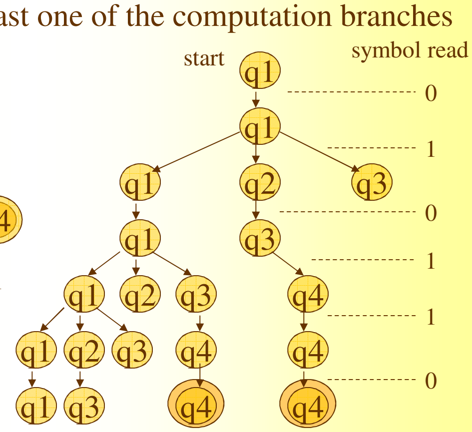
A non-deterministic FA (**NFA**) N_1 (Run for inputs 11, 101)

Tree of Possibilities

- A way to think of a non-deterministic computation is as a **tree of possibilities.**
  - The root corresponds to the start of the computation
  - Every branching point in the tree corresponds to a point in the computation at which the machine has multiple choices
  - The machine accepts if at least one of the computation branches ends in an accept state.

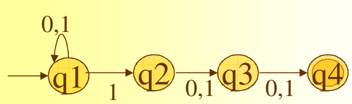


NFA N_1 (input is 010110)

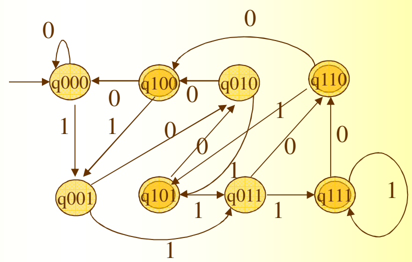


NFA vs. DFA

- NFAs are useful in several aspects.
  - Every NFA can be converted into an equivalent DFA (construction later).
  - Constructing NFAs is sometimes easier than directly constructing DFAs.
  - An NFA may be much smaller than its deterministic counterpart.
  - Its functioning may be easier to understand.
  - We will use non-determinism in more powerful computational models.
- Example.



NFA N_2



DFA M_2

They recognize the same language A={all strings over {0,1} containing a 1 in the third position from the end}

Formal Definition of NFAs

- A non-deterministic finite automaton (NFA) is specified by a 5-tuple (Q,Σ,δ,q_0,F) , where

Q is a finite set of states,
Σ is a finite alphabet,
$\delta : Q \times \Sigma_\varepsilon \rightarrow \boxed{P(Q)}$ is the transition function,
$q_0 \in Q$ is the initial state,
$F \subseteq Q$ is the set of final states.

Is a collection of all subsets of Q

$$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$$

- For NFA N_1 we have

δ :

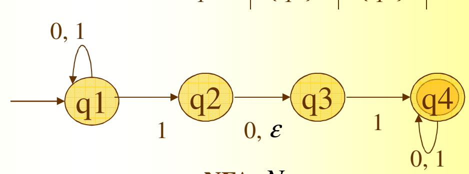
	0	*1*	ε
q1	*{q1}*	*{q1,q2}*	$\varnothing$
q2	*{q3}*	$\varnothing$	*{q3}*
q3	$\varnothing$	*{q4}*	$\varnothing$
q4	*{q4}*	*{q4}*	$\varnothing$

$Q = \{q1, q2, q3, q4\}$

$\Sigma = \{0,1\}$

$q_0 = q1$

$F = \{q4\}$



NFA N_1

Acceptance of Strings and the Language of NFA

- Let *N*= (Q,Σ,δ,q_0,F) be a NFA
- *N* accepts *w* if we can write *w* as $w = w_1, w_2, \ldots, w_n$, where each w_i is a member of Σ_ε and a sequence of states $r_0, r_1, r_2, \ldots, r_n$ exists in *Q* with the following three conditions:

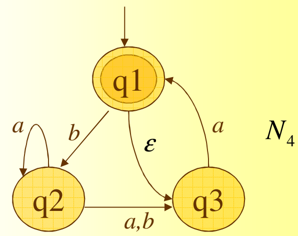
1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for $i = 0, \ldots, n-1$, and
3. $r_n \in F$

- If *L* is a set of strings that *N* accepts, we say that *L* is the *language of N* and write *L=L(N).*
- We say *N recognizes L* or *N accepts L.*
- In this example, N_4 recognizes the strings

  a, baba, baa,

but doesn't accept the strings

  b, bb, babba.



Subset Construction

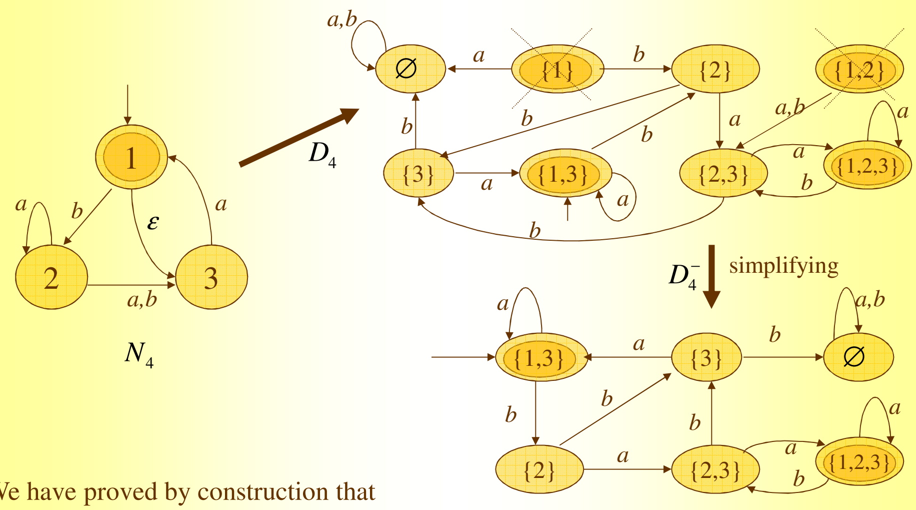
- For every NFA there is an *equivalent* (*accepts the same language*) DFA.
- But the DFA can have exponentially many states.
- Let N= $(Q_N, \Sigma, \delta_N, q_0, F_N)$ be an NFA.
- The equivalent DFA constructed by the *subset construction* is

$$D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D).$$

- For $R \subseteq Q_N$, we define
  E(R)={ q: q can be reached from *R* by traveling along *0* or more ε arrows*}*.
- Then,

1. $Q_D = P(Q_N)$, (= the set of subsets of Q_N),
2. For $R \in Q_D$ and $a \in \Sigma$ let $\delta_D(R,a) = E(\bigcup_{r \in R} \delta_N(r,a))$,
3. $q_{0D} = E(\{q_0\})$,
4. $F_D = \{R \in Q_D : R$ contains an accept state of $N\}$.

Example And The Theorem



We have proved by construction that

Theorem. Every NFA has an equivalent DFA.

Corollary. A language is regular if and only if some NFA recognizes it.

We have seen that for any NFA there exists an equivalent DFA. Hence

A language is regular if and only if some NFA recognizes it.

We will show today that regular languages are closed under regular operations.

Regular Operations (again)

- Let L1 and L2 be languages. We defined the regular operations ***union, concatenation,*** and ***star*** as follows.
  - ***Union:*** $L1 \cup L2 = \{w : w \in L1 \;\; or \;\; w \in L2\}.$
  - ***Concatenation:*** $L1 \circ L2 = \{wv : w \in L1 \;\; and \;\; v \in L2\}.$
  - ***Star:*** $L1^* = \{w_1 w_2 ... w_k : k \geq 0 \;\; and \;\; each \;\; w_i \in L1\}.$
- Example: Let the alphabet Σ be the standard 26 letters {a,b,…,z}.
  - If L1={good, bad} and L2= {boy, girl}, then

$L1 \cup L2 =$ {good, bad, boy, girl}.

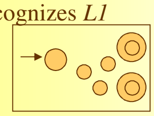
$L1 \circ L2 =$ {goodboy, badboy, goodgirl, badgirl}.

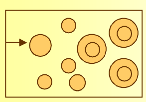
$L1^* =$ { ε , good, bad, goodgood, badgood, badbad, goodbad, goodgoodgood, goodgoodbad, goodbadbad, …}

Th.1 The class of regular languages is closed under the union operation.

- We have regular languages *L1* and *L2* and want to prove that $L1 \cup L2$ is regular.
- The idea is to take two NFAs *N1* and *N2* for *L1* and *L2*, and combine them into one new NFA *N*.
- *N* must accept its input if either *N1* or *N2* accepts this input
- *N* will have a new state that branches to the start states of the old machines *N1, N2* with ε arrows
- In this way *N* non-deterministically guesses which of the two machines accepts the input
- If one of them accepts the input then *N* will accept it, too

$N1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes *L1*





$N2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes *L2*

$Q = \{q_0\} \cup Q_1 \cup Q_2$

$F = F_1 \cup F_2$

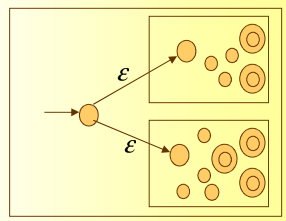
q_0 *is the start state*

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \varnothing & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

$q \in Q$

$a \in \Sigma_\varepsilon$

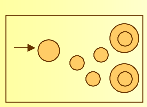
$N = (Q, \Sigma, \delta, q_0, F)$ recognizes $L1 \cup L2$



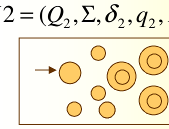
Th.2 The class of regular languages is closed under the concatenation operation.

- We have regular languages *L1* and *L2* and want to prove that $L1 \circ L2$ is regular.
- The idea is to take two NFAs *N1* and *N2* for *L1* and *L2*, and combine them into a new NFA *N*.
- *N* accepts when the input can be split into two parts, the first accepted by *N1* and the second by *N2*
- We can think of *N* as non-deterministically guessing where to make the split

$N1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes *L1*



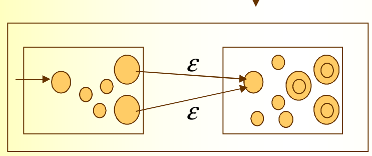
$N2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes *L2*



$Q = Q_1 \cup Q_2$

F_2 *is the set of final states*

q_1 *is the start state*



$N = (Q, \Sigma, \delta, q_1, F_2)$ recognizes $L1 \circ L2$

$q \in Q$

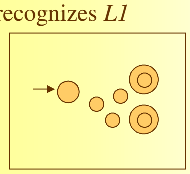
$a \in \Sigma_\varepsilon$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1, q \notin F_1 \\ \delta_1(q,a) & q \in F_1, a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1, a = \varepsilon \\ \delta_2(q,a) & q \in Q_1 \end{cases}$$

Th.3 The class of regular languages is closed under the star operation.

- We have regular language *L1* and want to prove that *L1** is regular.
- We take an NFA *N1* for *L1*, and modify it to recognize *L1**.
- The resulting NFA *N* accepts its input if it can be broken into several pieces and *N1* accepts each piece.
- *N* is like *N1* with additional ε arrows returning to the start state from the accept state.
- In addition we must modify *N* so that it accepts ε, which always is a member of *L1**.

$N1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes *L1*



$Q = \{q_0\} \cup Q_1$

$F = \{q_0\} \cup F_1$

q_0 *is the start state*

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1,\ q \notin F_1 \\ \delta_1(q,a) & q \in F_1, a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1,\ a = \varepsilon \\ \{q_1\} & q = q_0,\ a = \varepsilon \\ \varnothing & q = q_0, a \neq \varepsilon \end{cases}$$

$q \in Q$

$a \in \Sigma_\varepsilon$

$N = (Q, \Sigma, \delta, q_0, F)$ recognizes *L1**