Finite Automata and Formal Languages

TMV026/DIT321– LP4 2012

Lecture 6
Ana Bove

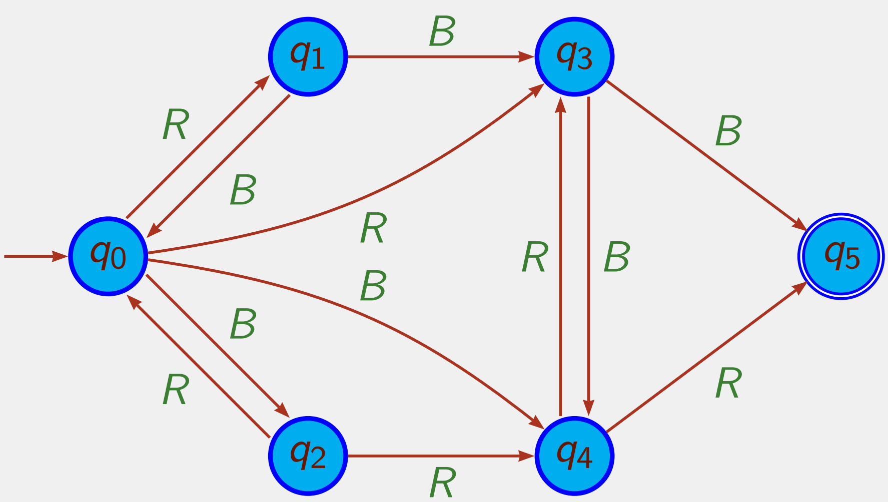
March 26th 2012

Overview of today's lecture:

- Equivalence between DFA and NFA
- NFA with ϵ-Transitions

Example: NFA Representation of Gilbreath's Principle

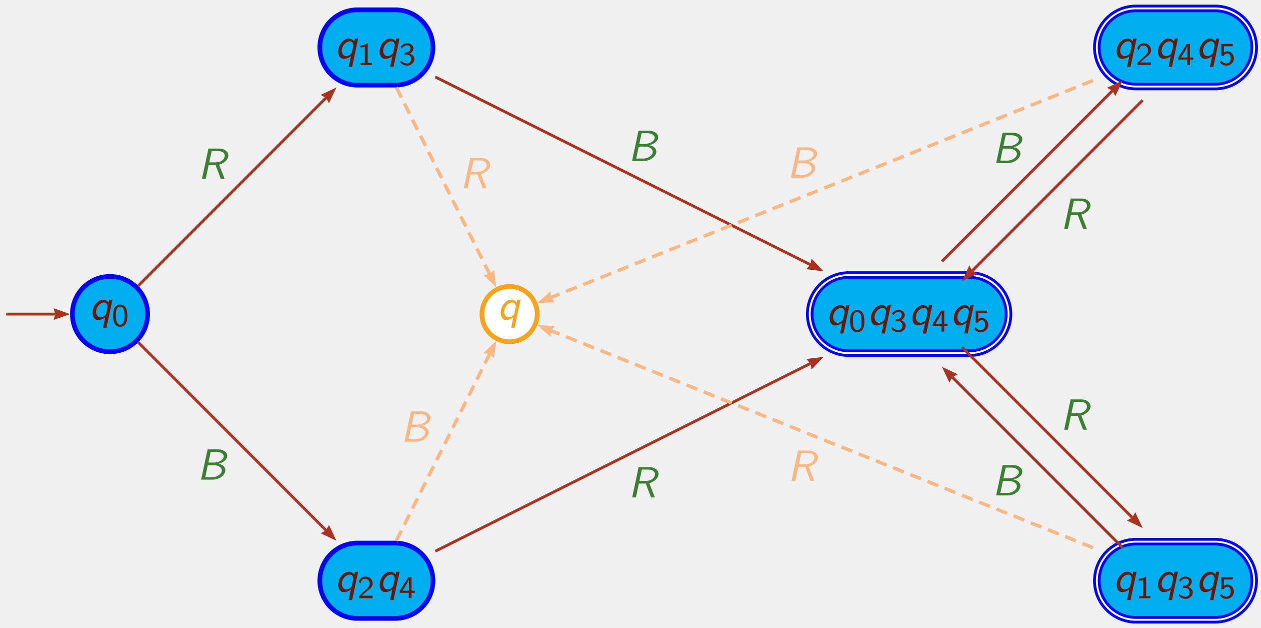
This is a model of Gilbreath's principle when we shuffle 2 non-empty alternating decks of cards, one starting with a red card and one starting with a black one. Let $\Sigma = \{B, R\}$ represent a black or red card respectively.



q_0 starts with B and R
q_1 both start with B
q_2 both start with R
q_3 starts with B and ϵ
q_4 starts with R and ϵ
q_5 both ϵ

What does the principle say? Let us build the corresponding DFA.

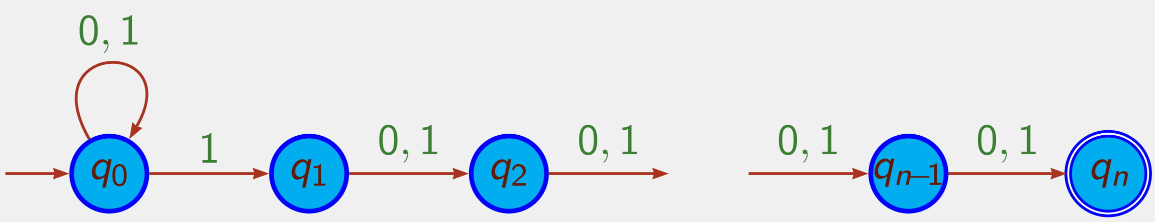
Example: DFA Representation of Gilbreath's Principle



What does the principle say?

A Bad Case for the Subset Construction

Proposition: *Any DFA recognising the same language as the NFA below has at least 2^n states:*



This NFA recognises strings over $\{0, 1\}$ such that the nth symbol from the end is a 1.

Proof: Let $\mathcal{L}_n = \{x1u \mid x \in \Sigma^*, u \in \Sigma^{n-1}\}$ and $D = (Q, \Sigma, \delta, q_0, F)$ a DFA.
We want to show that if $|Q| < 2^n$ then $\mathcal{L}(D) \neq \mathcal{L}_n$.

A Bad Case for the Subset Construction (Cont.)

Lemma: *If $\Sigma = \{0, 1\}$ and $|Q| < 2^n$ then there exists $x, y \in \Sigma^*$ and $u, v \in \Sigma^{n-1}$ such that $\hat{\delta}(q_0, x0u) = \hat{\delta}(q_0, y1v)$.*

Proof: Let us define a map $\Sigma^n \to Q$ such that $z \mapsto \hat{\delta}(q_0, z)$.

This map cannot be *injective* because $|Q| < 2^n = |\Sigma^n|$.

Hence, we have $a_1 \dots a_n \neq b_1 \dots b_n$ such that

$$\hat{\delta}(q_0, a_1 \dots a_n) = \hat{\delta}(q_0, b_1 \dots b_n)$$

Let us assume that $a_i = 0$ and $b_i = 1$.

Let $x = a_1 \dots a_{i-1}$, $y = b_1 \dots b_{i-1}$ and let $u = a_{i+1} \dots a_n 0^{i-1}$ and $v = b_{i+1} \dots b_n 0^{i-1}$.

Recall that for a DFA, $\hat{\delta}(q, zw) = \hat{\delta}(\hat{\delta}(q, z), w)$ (slide 7, lecture 4) and hence:

$$\hat{\delta}(q_0, x0u) = \hat{\delta}(q_0, a_1 \dots a_n 0^{i-1}) = \hat{\delta}(\hat{\delta}(q_0, a_1 \dots a_n), 0^{i-1}) =$$
$$\hat{\delta}(\hat{\delta}(q_0, b_1 \dots b_n), 0^{i-1}) = \hat{\delta}(q_0, b_1 \dots b_n 0^{i-1}) = \hat{\delta}(q_0, y1v)$$

A Bad Case for the Subset Construction (Cont.)

Lemma: *If $|Q| < 2^n$ then $\mathcal{L}(D) \neq \mathcal{L}_n$.*

Proof: Assume $\mathcal{L}(D) = \mathcal{L}_n$.

Let $x, y \in \Sigma^*$ and $u, v \in \Sigma^{n-1}$ as in previous lemma.

Then we must have that $y1v \in \mathcal{L}(D)$ but $x0u \notin \mathcal{L}(D)$,
That is, $\hat{\delta}(q_0, y1v) \in F$ but $\hat{\delta}(q_0, x0u) \notin F$.

However, this contradicts the previous lemma that says that $\hat{\delta}(q_0, x0u) = \hat{\delta}(q_0, y1v)$.

Product Construction for NFA

Definition: Given 2 NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ over the same alphabet Σ, we define the product $N_1 \times N_2 = (Q, \Sigma, \delta, q_0, F)$ as follows:

- $Q = Q_1 \times Q_2$;
- $\delta((p_1, p_2), a) = \delta_1(p_1, a) \times \delta_2(p_2, a)$;
- $q_0 = (q_1, q_2)$;
- $F = F_1 \times F_2$.

Lemma: $(t_1, t_2) \in \hat{\delta}((p_1, p_2), x)$ *iff* $t_1 \in \hat{\delta}_1(p_1, x)$ *and* $t_2 \in \hat{\delta}_2(p_2, x)$

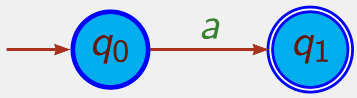
Proof: By induction on x.

Proposition: $\mathcal{L}(N_1 \times N_2) = \mathcal{L}(N_1) \cap \mathcal{L}(N_2)$.

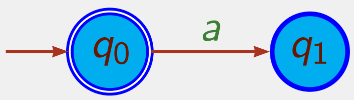
Complement for NFA

OBS: Given NFA $N = (Q, \Sigma, \delta, q, F)$ and $N' = (Q, \Sigma, \delta, q, Q - F)$ we do *not* have in general that $\mathcal{L}(N') = \Sigma^* - \mathcal{L}(N)$.

Example: Let $\Sigma = \{a\}$ and N and N' as follows:



$\mathcal{L}(N) = \{a\}$

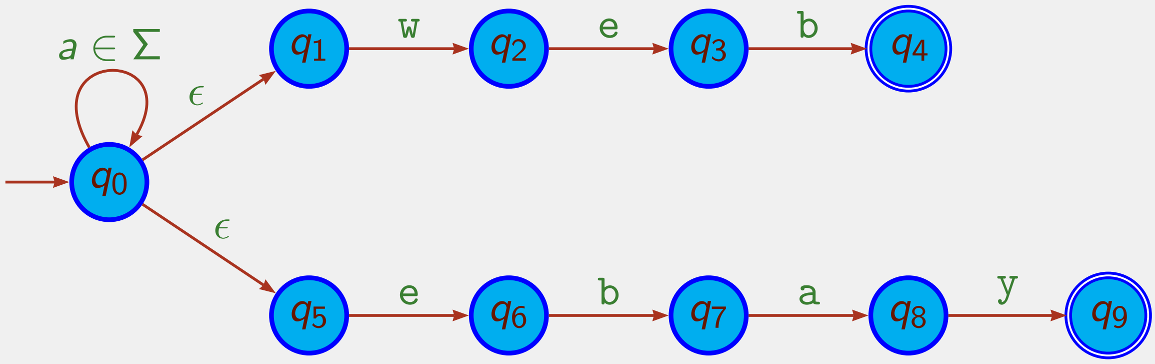


$\mathcal{L}(N') = \{\epsilon\} \neq \Sigma^* - \{a\}$

NFA with ϵ-Transitions

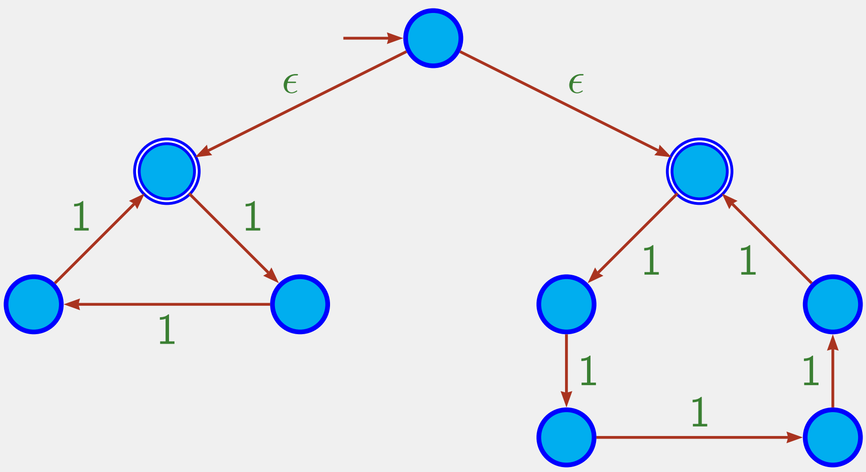
Another useful extension of automata that does not add more power is the possibility to allow ϵ-transitions, that is, transitions from one state to another *without* reading any input symbol.

Example: The following ϵ-NFA searches for the keyword `web` and `ebay`:



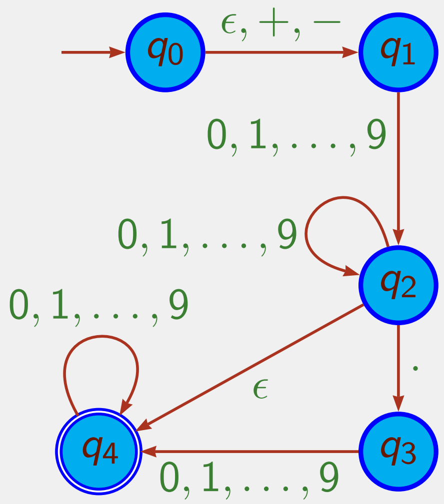
ϵ-NFA Accepting Words of Length Divisible by 3 or by 5

Example: Let $\Sigma = \{1\}$.



ϵ-NFA Accepting Decimal Numbers

Example: A NFA accepting number with an optional +/- symbol and an optional decimal part can be the following:



	+,-	.	0,1,...,9	ϵ
$\rightarrow q_0$	$\{q_1\}$	$\emptyset$	$\emptyset$	$\{q_1\}$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$	$\emptyset$
q_2	$\emptyset$	$\{q_3\}$	$\{q_2\}$	$\{q_4\}$
q_3	$\emptyset$	$\emptyset$	$\{q_4\}$	$\emptyset$
$*q_4$	$\emptyset$	$\emptyset$	$\{q_4\}$	$\emptyset$

The uses of ϵ-transitions represent the *optional* symbol +/- and the *optional* decimal part.

NFA with ϵ-Transitions

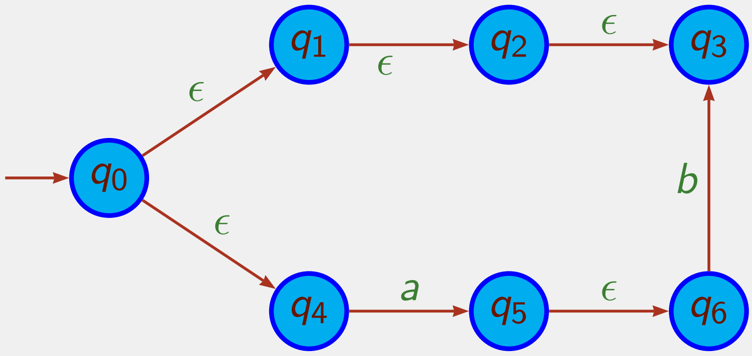
Definition: A *NFA with ϵ-transitions* (ϵ-NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ consisting of:

1. A finite set Q of *states*;
2. A finite set Σ of *symbols* (alphabet);
3. A *transition function* $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}ow(Q)$ ("partial" function that takes as argument a state and a symbol or the ϵ-transition, and returns a *set of states*);
4. A *start state* $q_0 \in Q$;
5. A set $F \subseteq Q$ of *final* or *accepting* states.

ϵ-Closures

Informally, the *ϵ-closure* of a state q is the set of states we can reach by *only* following paths labelled with ϵ.
We recursively follow all transitions out of a state q that are labelled ϵ.

Example: For the automaton



the ϵ-closure of q_0 is $\{q_0, q_1, q_2, q_3, q_4\}$.

ϵ-Closures

Definition: Formally, we define the *ϵ-closure* of a set of states with the following 2 rules:

$$\frac{q \in S}{q \in \text{ECLOSE}(S)} \qquad \frac{q \in \text{ECLOSE}(S) \qquad p \in \delta(q, \epsilon)}{p \in \text{ECLOSE}(S)}$$

Definition: We say that S is *ϵ-closed* iff $S = \text{ECLOSE}(S)$.

ϵ-Closures: Remarks

- The ϵ-closure of a single state q can be computed as ECLOSE($\{q\}$).
- ECLOSE($\emptyset$) $= \emptyset$.
- S is ϵ-closed iff $q \in S$ and $p \in \delta(q, \epsilon)$ implies $p \in S$.
- Intuitively, $p \in$ ECLOSE(S) iff there exists $q \in S$ and a sequence of ϵ-transitions such that

$$q \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} \quad \xrightarrow{\epsilon} q_n \xrightarrow{\epsilon} p$$

- We can prove that ECLOSE(S) is the *smallest* subset of Q containing S which is ϵ-closed.

Functional Representation of ϵ-Closures

```
import List(union)

e_jump :: Q -> [Q]
e_jump Q0 = [Q1,Q4]
e_jump Q1 = [Q2]
e_jump Q2 = [Q3]
e_jump Q5 = [Q6]
e_jump _ = []

isSub :: [Q] -> [Q] -> Bool
isSub ps qs = and (map (\x -> elem x qs) ps)

closure :: [Q] -> [Q]
closure qs = let qs' = qs >>= e_jump
             in if isSub qs' qs then qs
                                else closure (union qs qs')
```

Extending the Transition Function to Strings

Definition: Given an ϵ-NFA $E = (Q, \Sigma, \delta, q_0, F)$ we define

$$\begin{aligned}
&\hat{\delta} : Q \times \Sigma^* \to [Q] \\
&\hat{\delta}(q, \epsilon) = \text{ECLOSE}(\{q\}) \\
&\hat{\delta}(q, ax) = \bigcup_{p \in \Delta(\text{ECLOSE}(\{q\}), a)} \hat{\delta}(p, x) \\
&\qquad \text{where } \Delta(S, a) = \cup_{p \in S} \delta(p, a)
\end{aligned}$$

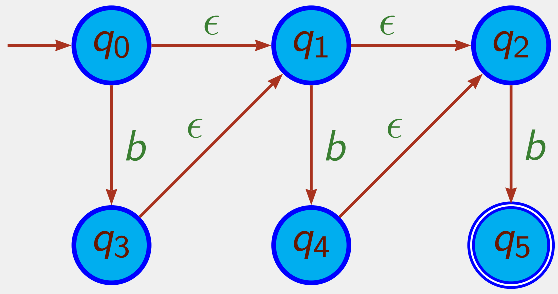
Remark: By definition we have that
$\hat{\delta}(q, a) = \text{ECLOSE}(\Delta(\text{ECLOSE}(\{q\}), a))$.

Remark: We can prove by induction on x that all sets $\hat{\delta}(q, x)$ are ϵ-closed.
This result uses that the union of ϵ-closed sets is also a ϵ-closed set.

Language Accepted by a ϵ-NFA

Definition: The *language* accepted by the ϵ-NFA $(Q, \Sigma, \delta, q_0, F)$ is the set $\mathcal{L} = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$.

Example: Let $\Sigma = \{b\}$.



The automaton accepts the language $\{b, bb, bbb\}$.

Note: Yet again, we could write a program that simulates a ϵ-NFA and let the program tell us whether a certain string is accepted or not.

Functional Representation of an ϵ-NFA

Let us implement the ϵ-NFA that recognises numbers (slide 10).

```
data Q = Q0 | Q1 | Q2 | Q3 | Q4  deriving (Eq,Show)

final :: Q -> Bool
final Q4 = True
final _ = False

e_jump :: Q -> [Q]
e_jump Q0 = [Q1]
e_jump Q2 = [Q4]
e_jump _ = []

isSub :: [Q] -> [Q] -> Bool

closure :: [Q] -> [Q]
```


Functional Representation of an ϵ-NFA (cont.)

```
delta :: Char -> Q -> [Q]
delta a Q0 | elem a "+-" = [Q1]
delta a Q1 | elem a "0123456789" = [Q2]
delta a Q2 | elem a "0123456789" = [Q2]
delta '.' Q2 = [Q3]
delta a Q3 | elem a "0123456789" = [Q4]
delta a Q4 | elem a "0123456789" = [Q4]
delta _ _ = []

run :: String -> Q -> [Q]
run [] q = closure [q]
run (a:xs) q = closure [q] >>= delta a >>= run xs

accepts :: String -> Bool
accepts xs = or (map final (run xs Q0))
```

Eliminating ϵ-Transitions

Definition: Given an ϵ-NFA $E = (Q_E, \Sigma, \delta_E, q_E, F_E)$ we define a DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ as follows:

- $Q_D = \{\mathrm{ECLOSE}(S) \mid S \in \mathcal{P}ow(Q_E)\}$
- $\delta_D(S, a) = \mathrm{ECLOSE}(\Delta(S, a))$ with $\Delta(S, a) = \cup_{p \in S} \delta(p, a)$
- $q_D = \mathrm{ECLOSE}(\{q_E\})$
- $F_D = \{S \in Q_D \mid S \cap F_E \neq \emptyset\}$

Note: This construction is similar to the subset construction but now we need to ϵ-close after each step.

Eliminating ϵ-Transitions

Let E be an ϵ-NFA and D the corresponding DFA.

Lemma: $\forall x \in \Sigma^*.\ \hat{\delta}_E(q_E, x) = \hat{\delta}_D(q_D, x)$.

Proof: By induction on x.

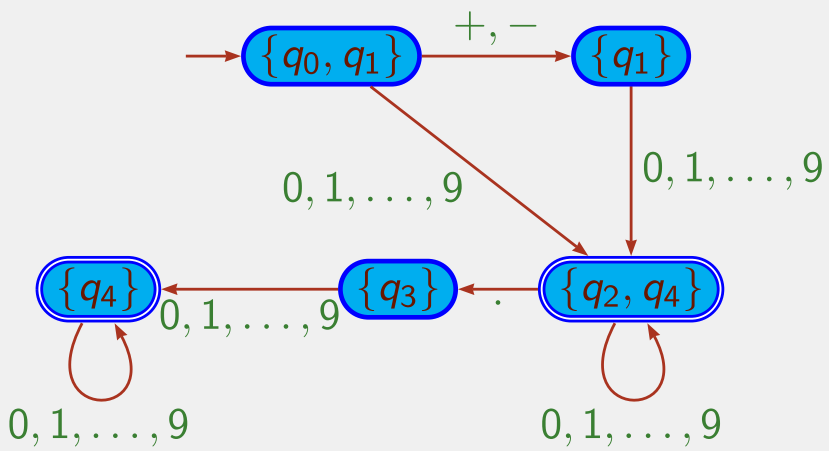
Proposition: $\mathcal{L}(E) = \mathcal{L}(D)$.

Proof: $x \in \mathcal{L}(E)$ iff $\hat{\delta}_E(q_E, x) \cap F_E \neq \emptyset$ iff $\hat{\delta}_E(q_E, x) \in F_D$ iff (by previous lemma) $\hat{\delta}_D(q_D, x) \in F_D$ iff $x \in \mathcal{L}(D)$.

Example: Eliminating ϵ-Transitions

Let us eliminate the ϵ-transitions in ϵ-NFA that recognises numbers in slide 10.

We obtain the following DFA:



Functional Representation of Eliminating ϵ-Transitions

```
pDelta :: Char -> [Q] -> [Q]
pDelta a qs = closure (qs >>= delta a)

pRun :: [Char] -> [Q] -> [Q]
pRun [] qs = qs
pRun (a:x) qs = pRun x (pDelta a qs)

run  :: String -> Q -> [Q]
run xs q = pRun xs (closure [q])

accepts :: String -> Bool
accepts xs = or (map final (run xs Q0))
```

Finite Automata and Regular Languages

We have shown that DFA, NFA and ϵ-NFA are equivalent in the sense that we can transform one to the other.

Hence, a language is *regular* iff there exists a finite automaton (DFA, NFA or ϵ-NFA) that accepts the language.