Finite Automata and Formal Languages

TMV027/DIT321– LP4 2013

Lecture 6
Ana Bove

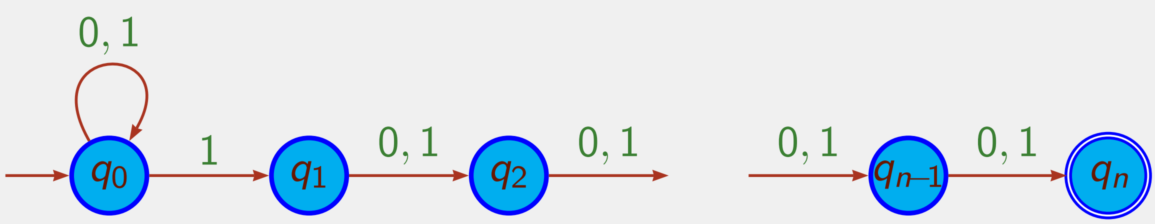
April 16th 2013

Overview of today's lecture:

- More on NFA;
- NFA with ϵ-Transitions;
- Equivalence between DFA and ϵ-NFA;
- Regular expresssions.

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 function $h : \Sigma^n \rightarrow Q$ such that $h(z) = \hat{\delta}(q_0, z)$.

h 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

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

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

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

Recall that for a DFA, $\hat{\delta}(q, zw) = \hat{\delta}(\hat{\delta}(q, z), w)$ 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* $t_2 \in \hat{\delta}_2(p_2, x)$.

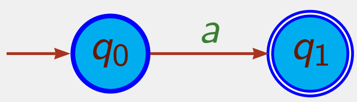
Proof: By induction on x.

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

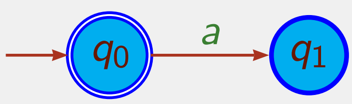
Complement for NFA

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:



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

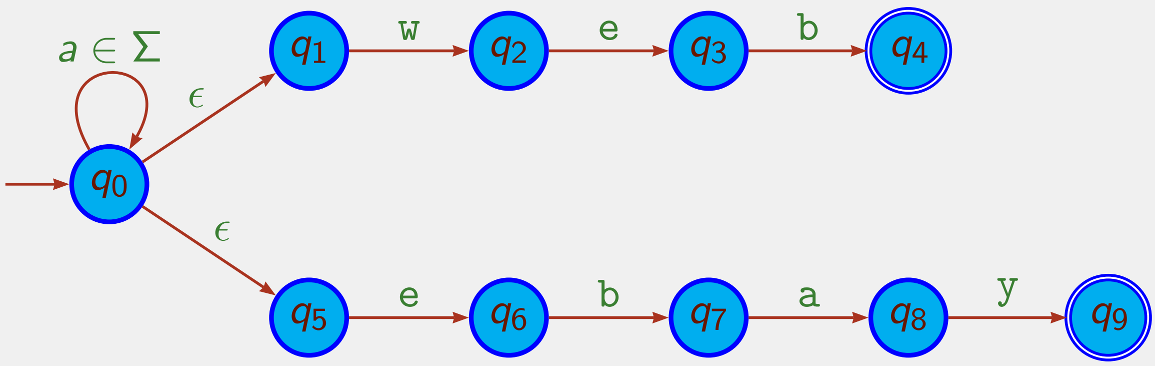


$\mathcal{L}(N') = \{\epsilon\} \neq \Sigma^* - \{a\}$

NFA with ϵ-Transitions

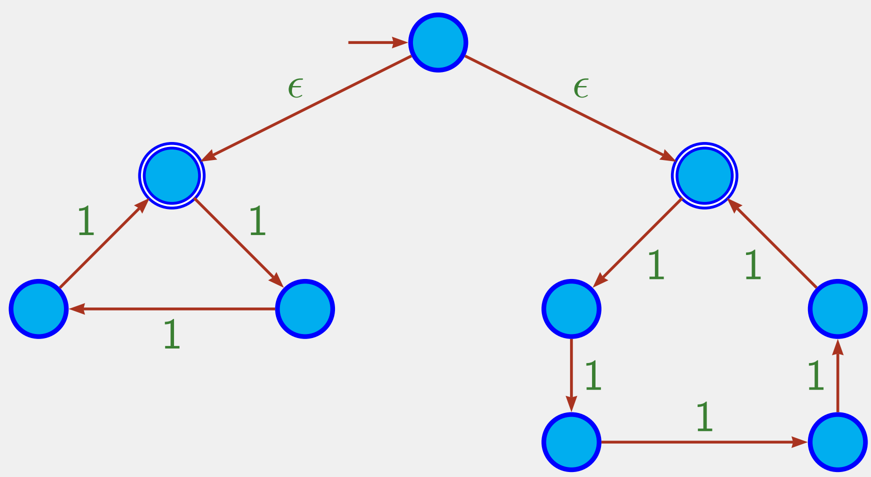
Another useful extension of automata that does not add more power is the possibility to allow ϵ-transitions, that is, transitions from one state to another *without* reading any input symbol.

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



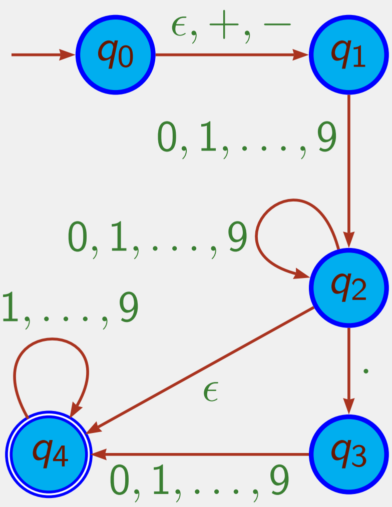
ϵ-NFA Accepting Words of Length Divisible by 3 or by 5

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



ϵ-NFA Accepting Decimal Numbers

Exercise: Define a NFA accepting number with an optional $+$/- symbol and an optional decimal part.



	+,-	.	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

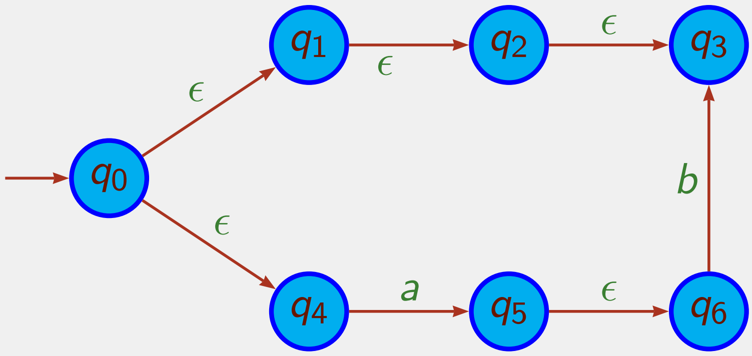
Definition: A *NFA with ϵ-transitions* (ϵ-NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ consisting of:

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 doing nothing or by *only* following paths labelled with 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 as follows:

- If $q \in S$ then $q \in \text{ECLOSE}(S)$;
- If $q \in \text{ECLOSE}(S)$ and $p \in \delta(q, \epsilon)$ then $p \in \text{ECLOSE}(S)$.

Note: Alternative formulation

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

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

Remarks: ϵ-Closures

- Intuitively, $p \in \text{ECLOSE}(S)$ iff there exists $q \in S$ and a sequence of ϵ-transitions such that

- The ϵ-closure of a single state q can be computed as $\text{ECLOSE}(\{q\})$.
- $\text{ECLOSE}(\emptyset) = \emptyset$.
- S is ϵ-closed iff $q \in S$ and $p \in \delta(q, \epsilon)$ implies $p \in S$.
- We can prove that $\text{ECLOSE}(S)$ is the *smallest* subset of Q containing S which is ϵ-closed.

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, $\hat{\delta}(q, a) = \text{ECLOSE}(\Delta(\text{ECLOSE}(\{q\}), a))$.

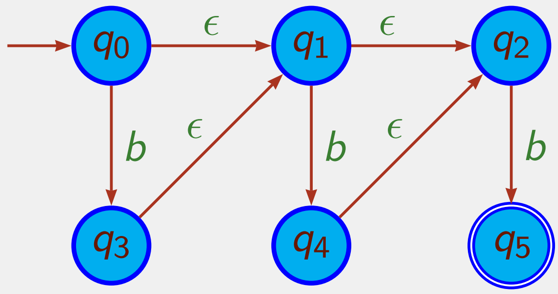
Remark: We can prove by induction on x that all sets $\hat{\delta}(q, x)$ are ϵ-closed.

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.

Exercise: Do it!

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:

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

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.

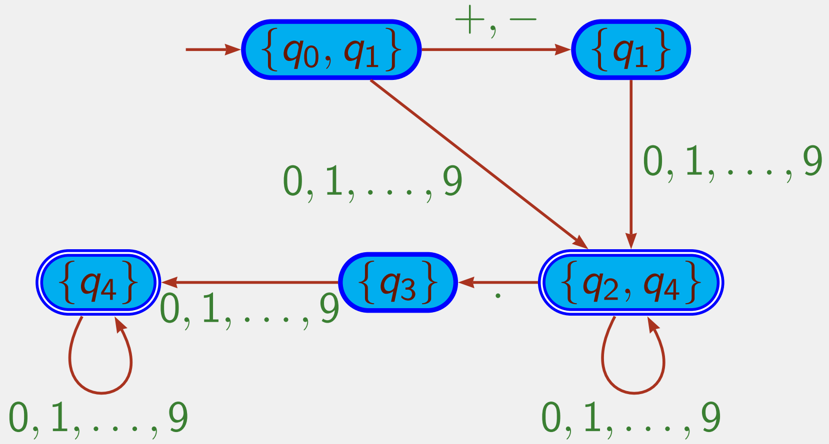
Proposition: $\mathcal{L}(E) = \mathcal{L}(D)$.

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 8.

We obtain the following DFA:



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.

Regular Expressions

Regular expressions (RE) are an "algebraic" way to denote languages.

RE are a simple way to express the strings we want to accept.

They serve as input language for certain systems.

Example: `grep` command in UNIX (K. Thompson) is given a (variation) of a RE as input

We will show that RE are as expressive as DFA and hence, they define all and only the *regular languages*.

Inductive Definition of Regular Expressions

Definition: Given an alphabet Σ, we inductively define the *regular expressions* over Σ as follows:

Base cases:
- The constants $\emptyset$ and ϵ are RE;
- If $a \in \Sigma$ then a is a RE.

Inductive steps: Given the RE R and S, we define the following RE:
- $R + S$ and RS are RE;
- R^* is RE.

The precedence of the operands is the following:

- The closure operator * has the highest precedence;
- Next comes concatenation;
- Finally, comes the operator $+$;
- We use parentheses (,) to change the precedence.

Another Way to Define the Regular Expressions

A nicer way to define the regular expressions is by giving the following BNF (Backus-Naur Form), for $a \in \Sigma$:

$$R ::= \emptyset \mid \epsilon \mid a \mid R + R \mid RR \mid R^*$$

alternatively

$$R, S ::= \emptyset \mid \epsilon \mid a \mid R + S \mid RS \mid R^*$$

Note: BNF is a way to declare the syntax of a language.

It is very useful when describing *context-free grammars* and in particular the syntax of most programming languages.

Functional Representation of Regular Expressions

```
data RExp a = Empty | Epsilon | Atom a |
              Plus (RExp a) (RExp a) |
              Concat (RExp a) (RExp a) |
              Star (RExp a)
```

For example the expression $b + (bc)^*$ is given as

```
Plus (Atom "b") (Star (Concat (Atom "b") (Atom "c")))
```


Language Defined by the Regular Expressions

Given a RE R, it defines the language $\mathcal{L}(R)$.

Definition: The *language* defined by a regular expression is defined by recursion on the expression:

Base cases:
- $\mathcal{L}(\emptyset) = \emptyset$;
- $\mathcal{L}(\epsilon) = \{\epsilon\}$;
- Given $a \in \Sigma$, $\mathcal{L}(a) = \{a\}$.

Recursive cases:
- $\mathcal{L}(R + S) = \mathcal{L}(R) \cup \mathcal{L}(S)$;
- $\mathcal{L}(RS) = \mathcal{L}(R)\mathcal{L}(S)$;
- $\mathcal{L}(R^*) = \mathcal{L}(R)^*$.

Note: $x \in \mathcal{L}(R)$ iff x is generated/accepted by R.

Notation: We write $x \in R$ or $x \in \mathcal{L}(R)$ indistinctly.

Overview of Next Lecture

Sections 3.2, 3.4:

- More on RE;
- Equivalence between FA and RE;
- Algebraic laws for regular expressions.