Finite Automata and Formal Languages

TMV026/DIT321– LP4 2011

Ana Bove

Lecture 5

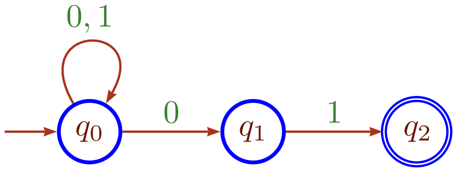
March 29th 2011

Overview of today's lecture:

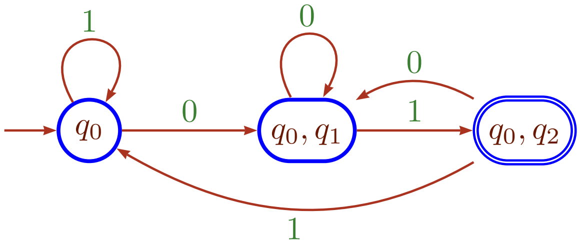
- **Equivalence between DFA and NFA**
- **More on NFA**
- **NFA with ϵ-Transitions**

Example: Subset Construction

Let us apply the subset construction to the NFA



We obtain the following DFA:



By only computing the *accessible* states (from the start state) we are able to keep the total number of states to 3 (and not 8).

Functional Representation of the Subset Construction

Given a (typed modified) δ_N function:

```
delta ::  S -> Q -> [Q]
```

we can define the (typed modified) δ_D function:

```
pDelta :: S -> [Q] -> [Q]
pDelta a qs = concat (map (delta a) qs)
```

or (with the monadic notation)

```
pDelta a qs = qs >>= delta a
```

or

```
pDelta a qs = do p <- qs
                 delta a p
```


Functional Representation of the Subset Construction

```
pFinal :: [Q] -> Bool
pFinal qs = or (map final qs)


pRun :: [S] -> [Q] -> [Q]
pRun [] qs = qs
pRun (a:xs) qs = pRun xs (pDelta a qs)


pAccepts :: [S] -> Bool
pAccepts xs = pFinal (pRun xs [Q0])
```

Testing the Correction of the Subset Construction

```
test :: [S] -> Bool
test xs = run xs Q0 == pRun xs [Q0]     -- run @ slides 22/23 lec 4
```

Informally, let `xs` be `[x1,...,xn]`. Then:

```
run [x1,...,xn] q = delta x1 q >>= run [x2,...,xn]
= delta x1 q >>= (\p -> delta x2 p >>= run [...,xn])
= delta x1 q >>= (\p -> ... >>= (\r -> delta xn r >>= return)...)
= delta x1 q >>= delta x2 >>= .. >>= delta xn

pRun [x1,...,xn] [q] = pDelta xn (... (pDelta x1 [q])...)
= [q] >>= delta x1 >>= ... >>>= delta xn
= delta x1 q >>= delta x2 >>= .. >>= delta xn
```


Towards the Correction of the Subset Construction

Formally we have that

Proposition: $\forall x.\forall q.\ \hat{\delta}_N(q, x) = \hat{\delta}_D(\{q\}, x)$.

Proof: By induction on x. Basis case is trivial.

The inductive step is:

$$\begin{array}{rcll}
\hat{\delta}_N(q, ax) & = & \bigcup_{p \in \delta_N(q,a)} \hat{\delta}_N(p, x) & \text{by definition of } \hat{\delta}_N \\
 & = & \bigcup_{p \in \delta_N(q,a)} \hat{\delta}_D(\{p\}, x) & \text{by IH} \\
 & = & \hat{\delta}_D(\delta_N(q, a), x) & \text{see lemma below} \\
 & = & \hat{\delta}_D(\delta_D(\{q\}, a), x) & \text{remark on slide 27 lecture 4} \\
 & = & \hat{\delta}_D(\{q\}, ax) & \text{by definition of } \hat{\delta}_D
\end{array}$$

Lemma: *For all words x and set of states S, $\hat{\delta}_D(S, x) = \bigcup_{p \in S} \hat{\delta}_D(\{p\}, x)$.*

Correction of the Subset Construction

Theorem: *Given a NFA N, if D is the DFA constructed from N by the subset construction then $\mathcal{L}(N) = \mathcal{L}(D)$.*

Proof: $x \in \mathcal{L}(N)$ iff $\hat{\delta}_N(q_0, x) \cap F_N \neq \emptyset$ iff $\hat{\delta}_N(q_0, x) \in F_D$.

By the previous proposition, this is equivalent to $\hat{\delta}_D(\{q_0\}, x) \in F_D$.

Since $\{q_0\}$ is the starting state in D the above is equivalent to $x \in \mathcal{L}(D)$.

Equivalence between DFA and NFA

Theorem: *A language $\mathcal{L}$ is accepted by some DFA iff $\mathcal{L}$ is accepted by some NFA.*

Proof: The "if" part is the result of the previous theorem (correctness of subset construction).

For the "only if" part we need to transform the DFA into a NFA.

Intuitively, each DFA can be seen as a NFA where there exists only one choice at each stage.

Formally, given $D = (Q, \Sigma, \delta_D, q_0, F)$ we define $N = (Q, \Sigma, \delta_N, q_0, F)$ such that, if $\delta_D(q, a) = p$ then $\delta_N(q, a) = \{p\}$.

It only remains to show (by induction on x) that if $\hat{\delta}_D(q_0, x) = p$ then $\hat{\delta}_N(q_0, x) = \{p\}$.

Application: Text Search

Suppose we are given a set of words, called *keywords*, and we want to find occurrences of any of these words in a text.

An useful way to proceed is to design a NFA that enters in an accepting state when it has recognised one of the keywords.

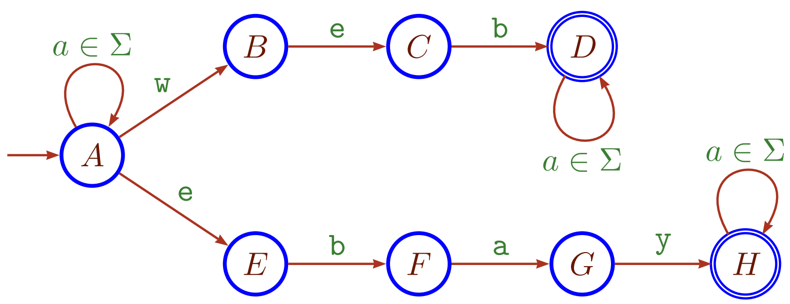
Then we could implement the NFA, or we could transform it to a DFA and get a deterministic (efficient) program.

Since we have proved the subset construction correct, we know the DFA will be correct (if the NFA is!).

This is a good example of a derivation of a *program* (the DFA) from a *specification* (the NFA).

Application: Text Search

The following (easy to write) NFA searches for the keyword `web` and `ebay`:



If one applies the subset construction one obtains the DFA of page 71 in the book.

Observe that the obtained DFA has the same number of states as the NFA.

Functional Representation: Text Search

```
data Q = A | B | C | D | E | F| G | H

delta :: Char -> Q -> [Q]
delta 'w' A = [A,B]
delta 'e' A = [A,E]
delta  _  A = [A]
delta 'e' B = [C]
delta 'b' C = [D]
delta 'b' E = [F]
delta 'a' F = [G]
delta 'y' G = [H]
delta  _  D = [D]
delta  _  H = [H]
delta  _  _ = []
```


Functional Representation: Text Search (cont.)

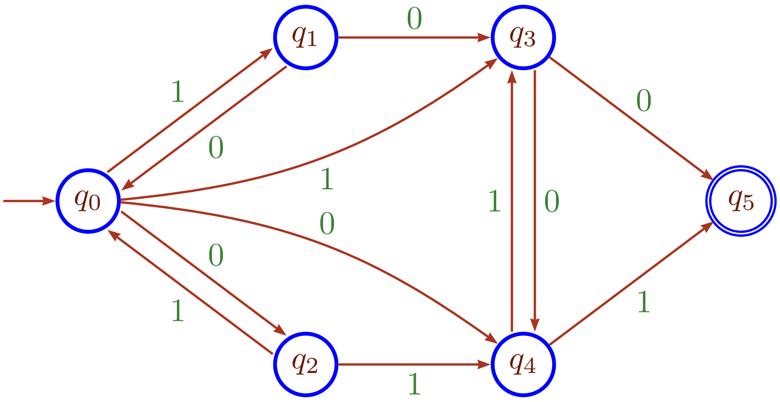
```
final :: Q -> Bool
final D = True
final H = True
final _ = False

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

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

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 $\Sigma = \{0, 1\}$ represent a black or red card respectively.



q_0 starts with 0 and 1

q_1 both start with 0

q_2 both start with 1

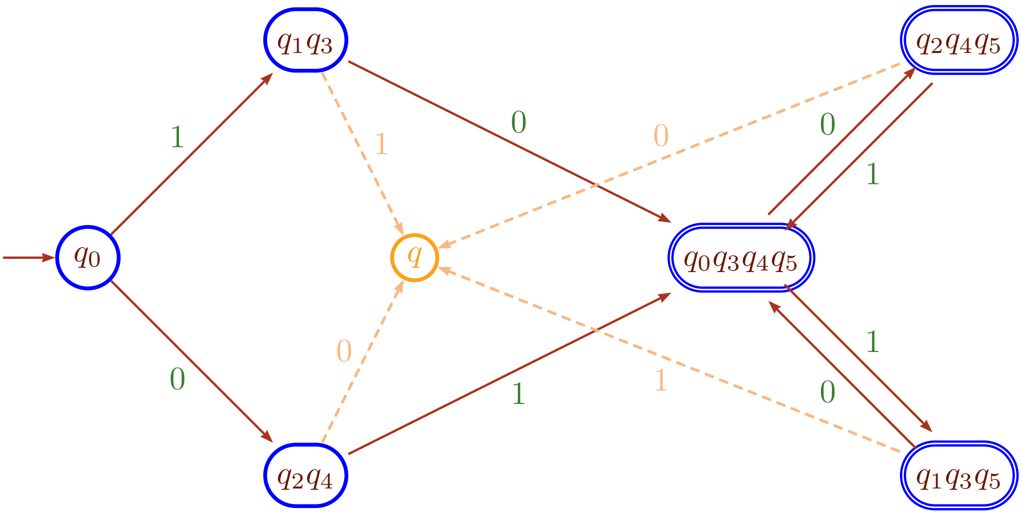
q_3 starts with 0 and ϵ

q_4 starts with 1 and ϵ

q_5 both ϵ

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

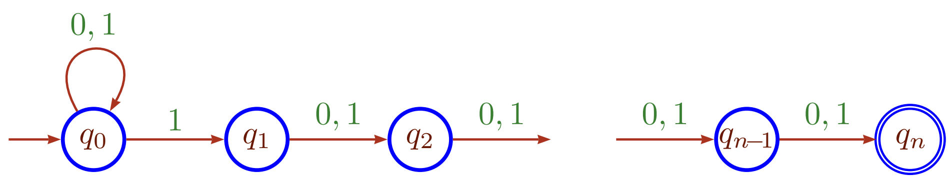
Example: DFA Representation of Gilbreath's Principle



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 $|Q| < 2^n$ 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 24, lecture 3) 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.)

Proof: (of the proposition: if $|Q| < 2^n$ then $\mathcal{L}(D) \neq \mathcal{L}_n$).

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:

- $Q = Q_1 \times Q_2$
- $\delta((p_1, p_2), a) = \delta_1(p_1, a) \times \delta_2(p_2, a)$
- $q_0 = (q_1, q_2)$
- $F = \{(p_1, p_2) \mid p_1 \in F_1, p_2 \in F_2\}$

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

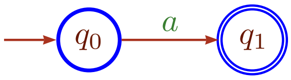
Proof: By induction on x.

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

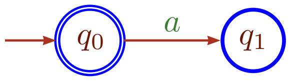
Complement for NFA

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:



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



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

Regular Languages

Recall: A language $\mathcal{L} \subseteq \Sigma^*$ is *regular* iff there exists a DFA D on the alphabet Σ such that $\mathcal{L} = \mathcal{L}(D)$.

Proposition: *A language $\mathcal{L} \subseteq \Sigma^*$ is regular iff there exists a NFA N such that $\mathcal{L} = \mathcal{L}(N)$.*

Proof: If $\mathcal{L}$ is regular then $\mathcal{L} = \mathcal{L}(D)$ for some DFA D. To any DFA D we can associate a NFA N_D such that $\mathcal{L}(D) = \mathcal{L}(N_D)$.

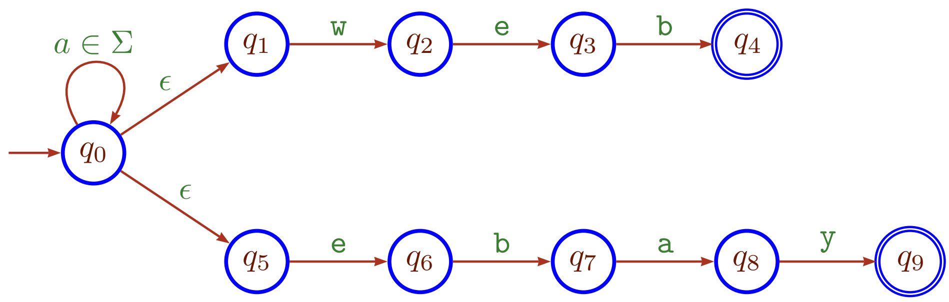
If $D = (Q, \Sigma, \delta, q_0, F)$ we simply take $N_D = (Q, \Sigma, \delta', q_0, F)$ with $\delta'(q, a) = \{\delta(q, a)\}$. Notice that $\delta' \in Q \times \Sigma \rightarrow \mathcal{P}ow(Q)$.

In the other direction, if $\mathcal{L} = \mathcal{L}(N)$ for some NFA N then, the subset construction gives a DFA D such that $\mathcal{L}(N) = \mathcal{L}(D)$ so $\mathcal{L}$ is regular.

NFA with ϵ-Transitions

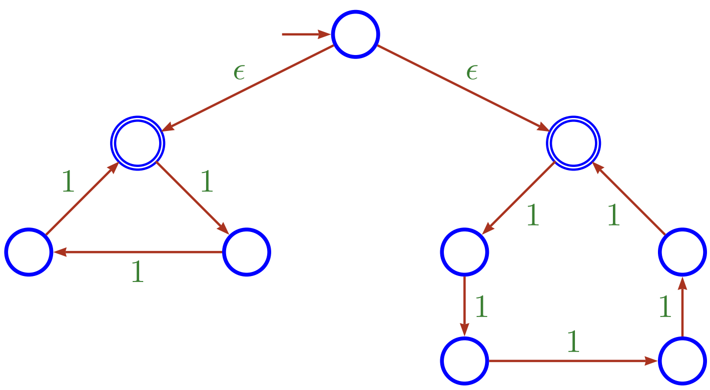
Another useful extension of automata that does not add more power is the possibility to allow ϵ-transitions, that is, transitions from one state to another *without* reading any input symbol.

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



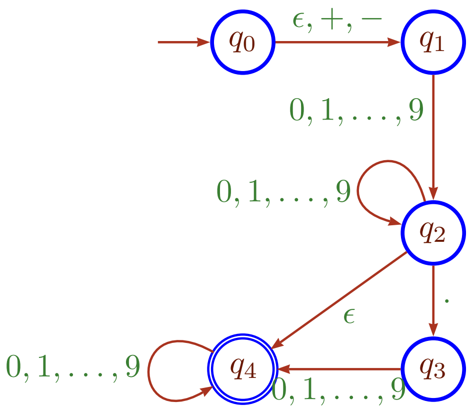
ϵ-NFA Accepting Words of Length Divisible by 3 or by 5

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

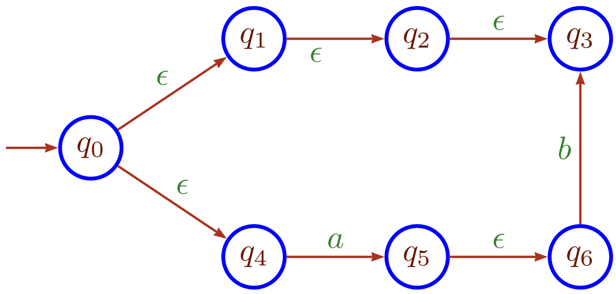
Definition: A *NFA with ϵ-transitions* (ϵ-NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ consisting of:

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.

Example: For the automaton



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

Informally, we recursively follow all transitions out of a state q that are labelled ϵ.

ϵ-Closures

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

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

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

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

$$q_1 \in \delta(q, \epsilon) \quad q_2 \in \delta(q_1, \epsilon) \quad \cdots \quad p \in \delta(q_n, \epsilon)$$

- 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')
```