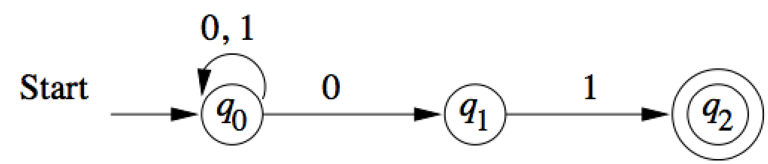
Equivalence between NFA and DFA

- For many languages, it is easier to construct an NFA than a DFA.
- Surprisingly, however, for any NFA N there is a DFA D, such that $L(D) = L(N)$, and vice versa.
- Finding such equivalent DFA involves an algorithm for subset construction, an important example on how an automaton B can be generically constructed from another automaton A.
- Given an NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$, we will construct a DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $L(D) = L(N)$

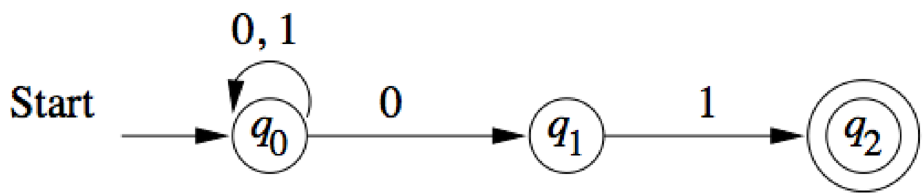
The subset construction method



- $Q_D = \{S : S \subseteq Q_N\}$.
  Note:
  - Q_D consists of all possible subsets of Q_N
  - $|Q_D| = 2^{|Q_N|}$, although most states in Q_D are likely to be garbage.
- $\Sigma_D = \Sigma_N$
- $q_{0D} = q_{0N}$
- $F_D = \{S \subseteq Q_N : S \cap F_N \neq \emptyset\}$

	0	1
$\emptyset$		
$\rightarrow \{q_0\}$		
$\{q_1\}$		
$\star\{q_2\}$		
$\{q_0, q_1\}$		
$\star\{q_0, q_2\}$		
$\star\{q_1, q_2\}$		
$\star\{q_0, q_1, q_2\}$		

Subset construction: transition function



δ_D: For every $S \subseteq Q_N$ and $a \in \Sigma$, $\delta_D(S,a) = \bigcup_{p \in S} \delta_N(p,a)$

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$\star\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\star\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\star\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$\star\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

	0	1
A	A	A
$\rightarrow B$	E	B
C	A	D
$\star D$	A	A
E	E	F
$\star F$	E	B
$\star G$	A	D
$\star H$	E	F

A more direct subset construction

- We can avoid generating all possible subsets of states by performing "lazy evaluation" on the subsets.
- To convert a NFA N into a DFA D the idea is to construct the transition table for D only for **accessible** states S in N, as follows:

Basis: $S = \{q_0\}$ is accessible

Induction: If state S is accessible, then for each input symbol a we compute the states

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$

Induction, in pseudo code:

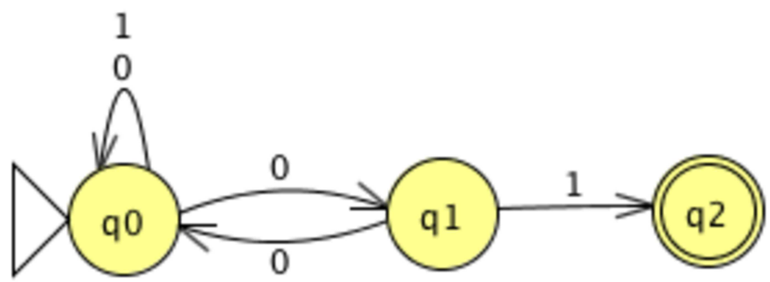
$acc := \{\ \};$

for each $p \in S$

$\quad acc := acc \cup \delta_N(p, a);$

$\delta_D(S, a) = acc$

Subset construction by "lazy evaluation": example



Obtaining the DFA for the NFA above using "lazy evaluation" of states sets.
(on the board)

A DFA obtained from a NFA recognizes the same language

- If $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ is the DFA constructed from NFA $N = (Q_n, \Sigma, \delta_N, q_0, F_N)$ by subset construction, then $L(D) = L(N)$.

Proof: First we show on an induction on $|w|$ that

$$\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$$

Basis: $w = \epsilon$. The claim follows from def.

Proof (cont.)

Induction:

$$\begin{aligned}
\hat{\delta}_D(\{q_0\}, xa) &\stackrel{\text{def}}{=} \delta_D(\hat{\delta}_D(\{q_0\}, x), a) \\
&\stackrel{\text{i.h.}}{=} \delta_D(\hat{\delta}_N(q_0, x), a) \\
&\stackrel{\text{cst}}{=} \bigcup_{p \in \hat{\delta}_N(q_0, x)} \delta_N(p, a) \\
&\stackrel{\text{def}}{=} \hat{\delta}_N(q_0, xa)
\end{aligned}$$

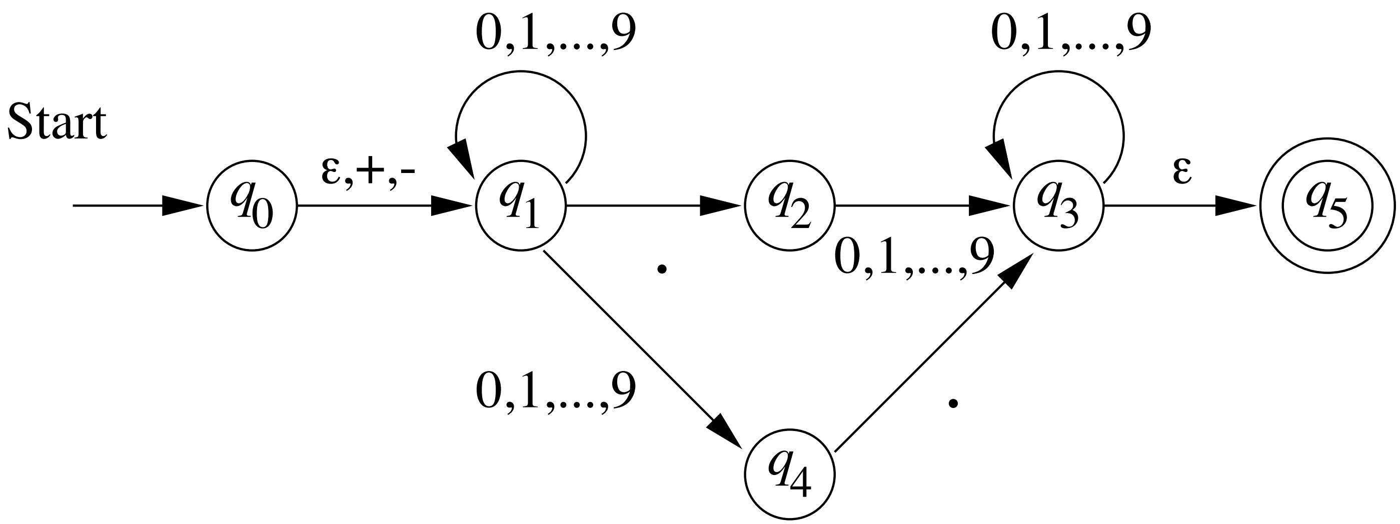
- Since both D and N accept w if and only if $\hat{\delta}_D(\{q_0\}, w)$ or $\hat{\delta}_N(q_0, w)$ contain a state in F_N, it follows that $L(D) = L(N)$.

Automata with spontaneous moves

- Such moves are depicted in the state-transition diagram as an arc labelled ϵ
- Such arcs are quite handy for assembling automata recognizing a regular composition of finite-state languages.
- Moreover, ϵ-NFA are useful in proving the equivalence between the language accepted by a FA (a machine-like description of a language) and by a *regular expression* (an algebraic description of a language)

Example of ϵ-NFA

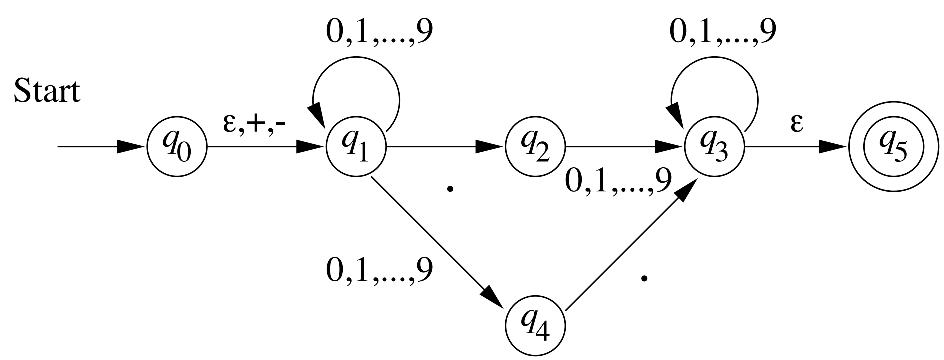
An ϵ-NFA that recognizes an optional + or - sign, a string of digits, a decimal point, and another string of digits.



ϵ-NFA formally

- An ϵ-NFA is a quintuple $(Q, \Sigma, \delta, q_0, F)$, where δ is a function from $Q \times \Sigma \cup \{\epsilon\}$ to the powerset of Q.
- Like other automata, an ϵ-NFA can also be represented by the transition table.

	ϵ	+,-	.	$0,\ldots,9$
$\rightarrow q_0$	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_3	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_4	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
$\star q_5$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

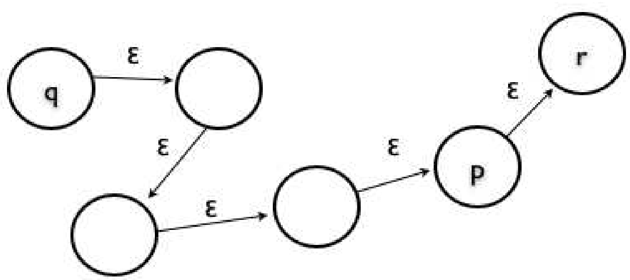


ECLOSE

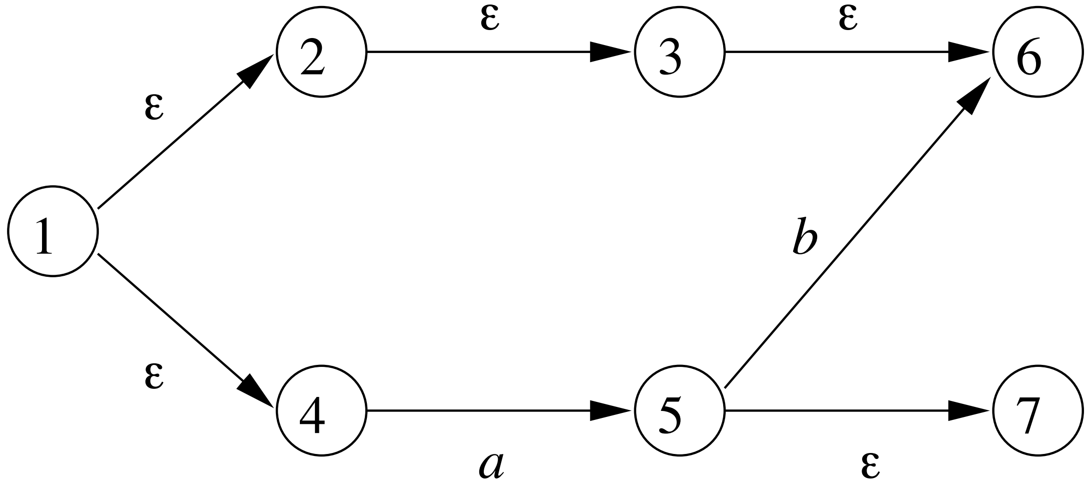
- $ECLOSE(q)$ yields all states from state q along any path whose arcs are labled with ϵ.
- If A is a set of states of an FA, then $ECLOSE(A) = \bigcup_{p \in A} ECLOSE(p)$
- Inductive definition of $ECLOSE(q)$, where q is a state:

  Basis: $q \in ECLOSE(q)$
  Induction:

  if $(p \in ECLOSE(q)$ and $r \in \delta(p, \epsilon))$ then $r \in ECLOSE(q)$



ECLOSE: example



For instance,

$$\text{ECLOSE}(1) = \{1, 2, 3, 4, 6\}$$

Extended transition function for ϵ-NFA

Basis: $\hat{\delta}(q, \epsilon) = ECLOSE(q)$

Induction:

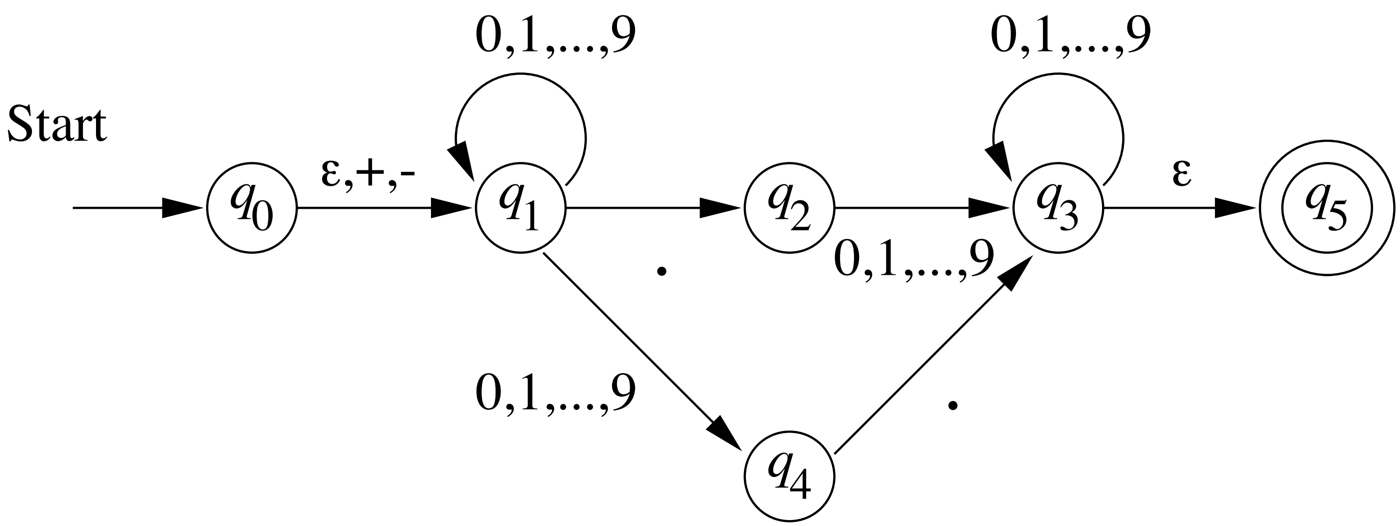
$$\hat{\delta}(q, xa) = ECLOSE\left(\bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a) \right)$$

Induction, in pseudo code: Let $\hat{\delta}(q, x) = \{p_1, p_2, \cdots, p_k\}$

$r := \{\ \}$;
for each $p_i \in \{p_1, p_2, \cdots, p_k\}$
    $r := r \cup \delta(p_i, a)$
$\hat{\delta}(p, xa) = ECLOSE(r)$;

Example

Let's compute $\hat{\delta}(q_0, .5)$ for the DFA below



Equivalent DFA for an ϵ-NFA

- To convert an ϵ-NFA $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ into a DFA $D = (Q_D, \Sigma, \delta_D, q_D, F)$ the construction we use is similar to the subset construction.
- The difference is that we must incorporate the ϵ-transitions via the ϵ-closure.
- We construct the transition table for D as follows:

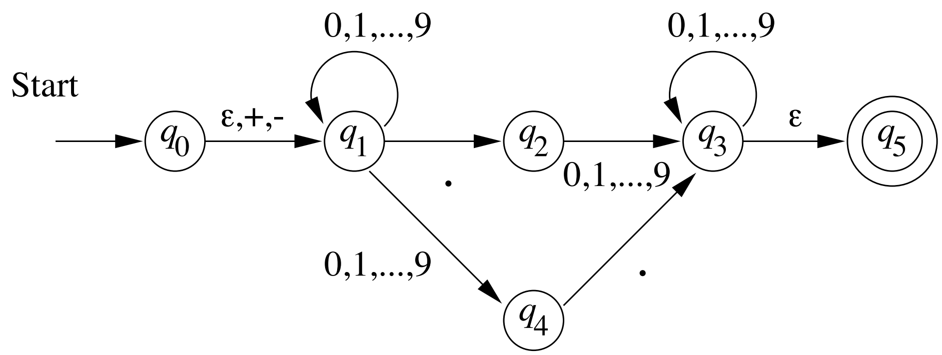
  Basis: $q_D = ECLOSE(q_0)$ is accessible in D

  Induction: If state S is accessible, then for each input symbol a we compute the states

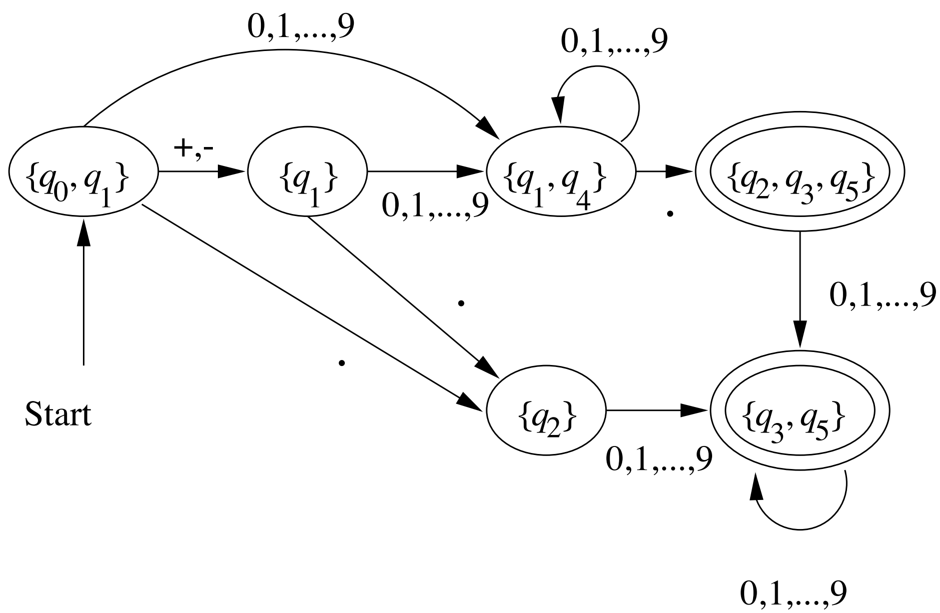
  $$\delta_D(S, a) = ECLOSE(\bigcup_{p \in S} \delta_E(p, a))$$

- $F_D = \{S : S \in Q_D \text{ and } S \cap F_E \neq \emptyset\}$

Equivalent DFA for an ϵ-NFA

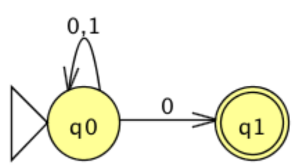


DFA D corresponding to E



Regular Expressions

- A FA (DFA, NFA, or ϵ-NFA) provides a "procedural" description for a machine that recognizes a regular language.
- A regular expression provides a "declarative" description for a regular language.
- E.g.: The set of all binary strings that end with a 0.



$$(0+1)^*0$$

- Now, instead of focusing on how regular languages are computed, we focus on the problem of describing finite or repeating patterns.

Where can we find applications for this?

Since REs provide a declarative way to express strings we want to accept, it can serve as input to systems that process strings. For example:

- The first step of compiling a program.
- Filtering email for spam.
- Sorting email into appropriate mailbox based on keywords
- Searching a complex directory structure by specifying patterns (e.g., UNIX-like `grep` command).

Operations on regular languages

Before describing the notation for RE, let's learn the operations on languages that the operators of RE represent.
Let L and M be languages, e.g., $L = \{01, 11\}$, $M = \{00, 10, 11\}$.

- Union: $L \cup M = \{w : w \in L$ or $w \in M\}$
- Concatenation:
  $L.M$ or just $LM = \{w : w = xy, x \in L, y \in M\}$
- Power: $L^0 = \{\epsilon\}, L^1 = L, L^{k+1} = L.L^k$
- Kleene Closure: $L^* = \bigcup_{i=0}^{\infty} L^i$

Note: $\emptyset^* = \{\epsilon\}$. Rationale: $\emptyset^0 = \{\epsilon\}$ and $\emptyset^i$ for $i \geq 1$ is empty.

w3.1-3

REs and the languages they define

Inductive definition of REs and the languages they define.

- **Basis:**
  - ϵ and $\emptyset$ are REs. $L(\epsilon) = \{\epsilon\}, L(\emptyset) = \emptyset$
  - If $a \in \Sigma$, then a is a RE. $L(a) = \{a\}$
- **Induction:**
  - If E is a RE, then (E) is a RE. $L((E)) = L(E)$
  - If E and F are REs, then $E + F$ is a RE. $L(E + F) = L(E) \cup L(F)$.
  - If E and F are REs, then $E.F$ is a RE. $L(E.F) = L(E).L(F)$.
  - If E is a RE, then E^* is a RE. $L(E^*) = (L(E))^*$.

RE: examples

- $L(01) = \{01\}$
- $L(01 + 0) = \{01, 0\}$
- $L(0(1 + 0)) = \{01, 00\}$. Note order of precedence of operators:
- $L(0^*) = \{\epsilon, 0, 00, 000, \cdots\}$.
- $L((0 + 10)^*)$ = all binary strings without consecutive 1s that end in 0.
- $L((0 + 10)^*(\epsilon + 1)) =$ all binary strings without two consecutive 1s.

- Order of precedence of operators: $* > . > +$
- Example: $01^* + 1$ is grouped $(0(1)^*) + 1$

Exercise

Provide a regular expression that defines the language of the DFA below: