Equivalence of DFA and NFA

- NFA's are usually easier to "program" in.

- Surprisingly, for any NFA N there is a DFA D, such that $L(D) = L(N)$, and vice versa.

- This involves the *subset construction*, an important example 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 details of the subset construction:

- $Q_D = \{S : S \subseteq Q_N\}$.

Note: $|Q_D| = 2^{|Q_N|}$, although most states in Q_D are likely to be garbage.

- $F_D = \{S \subseteq Q_N : S \cap F_N \neq \emptyset\}$

- For every $S \subseteq Q_N$ and $a \in \Sigma$,

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

Let's construct δ_D from the NFA on slide 27

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

Note: The states of D correspond to subsets of states of N, but we could have denoted the states of D by, say, $A - F$ just as well.

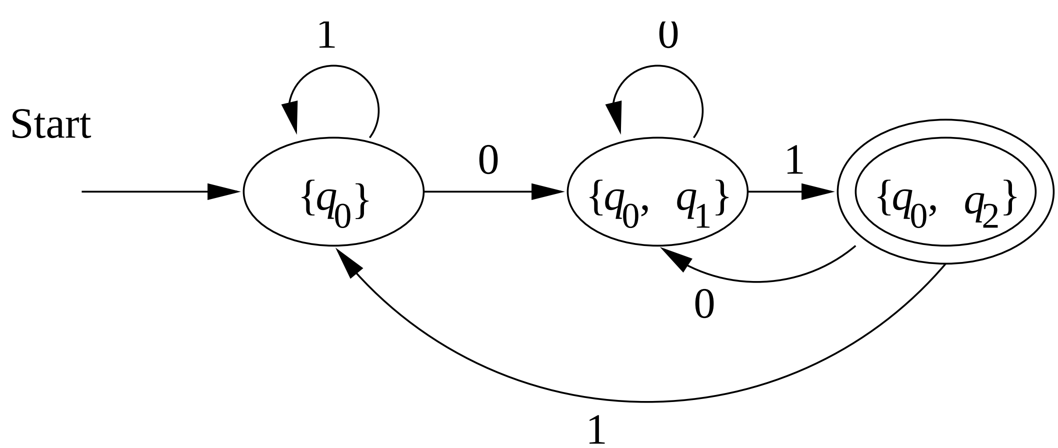
	0	1
A	A	A
$\rightarrow B$	E	B
C	A	D
$\star D$	A	A
E	E	F
$\star F$	E	B
$\star G$	A	D
$\star H$	E	F

We can often avoid the exponential blow-up by constructing the transition table for D only for accessible states S as follows:

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

Induction: If state S is accessible, so are the states in $\bigcup_{a \in \Sigma} \{\delta_D(S, a)\}$

Example: The "subset" DFA with accessible states only.



Theorem 2.11: Let D be the "subset" DFA of an NFA N. Then $L(D) = L(N)$.

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

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

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

Induction:

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

Now (**why?**) it follows that $L(D) = L(N)$.

Theorem 2.12: A language L is accepted by some DFA if and only if L is accepted by some NFA.

Proof: The "if" part is Theorem 2.11.

For the "only if" part we note that any DFA can be converted to an equivalent NFA by modifying the δ_D to δ_N by the rule

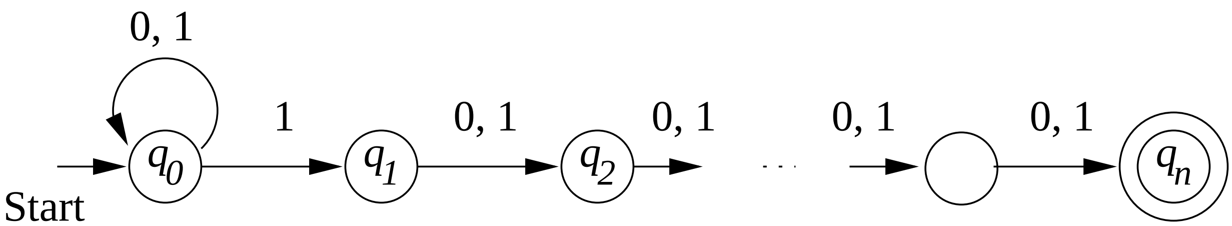
- If $\delta_D(q, a) = p$, then $\delta_N(q, a) = \{p\}$.

By induction on $|w|$ it will be shown in the tutorial that if $\widehat{\delta}_D(q_0, w) = p$, then $\widehat{\delta}_N(q_0, w) = \{p\}$.

The claim of the theorem follows.

Exponential Blow-Up

There is an NFA N with $n+1$ states that has no equivalent DFA with fewer than 2^n states



$$L(N) = \{x1c_2c_3\cdots c_n : x \in \{0,1\}^*, c_i \in \{0,1\}\}$$

Suppose an equivalent DFA D with fewer than 2^n states exists.

D must remember the last n symbols it has read.

There are 2^n bitsequences $a_1a_2\cdots a_n$

$$\begin{aligned} \exists\, q, a_1a_2\cdots a_n, b_1b_2\cdots b_n : q &= \hat{\delta}_D(q_0, a_1a_2\cdots a_n), \\ q &= \hat{\delta}_D(q_0, b_1b_2\cdots b_n), \\ a_1a_2\cdots a_n &\neq b_1b_2\cdots b_n \end{aligned}$$

Case 1:

$1a_2 \cdots a_n$
$0b_2 \cdots b_n$

Then q has to be both an accepting and a nonaccepting state.

Case 2:

$a_1 \cdots a_{i-1} 1 a_{i+1} \cdots a_n$
$b_1 \cdots b_{i-1} 0 b_{i+1} \cdots b_n$

Now $\hat{\delta}_D(q_0, a_1 \cdots a_{i-1} 1 a_{i+1} \cdots a_n 0^{i-1}) =$
$\hat{\delta}_D(q_0, b_1 \cdots b_{i-1} 0 b_{i+1} \cdots b_n 0^{i-1})$

and $\hat{\delta}_D(q_0, a_1 \cdots a_{i-1} 1 a_{i+1} \cdots a_n 0^{i-1}) \in F_D$

$\hat{\delta}_D(q_0, b_1 \cdots b_{i-1} 0 b_{i+1} \cdots b_n 0^{i-1}) \notin F_D$

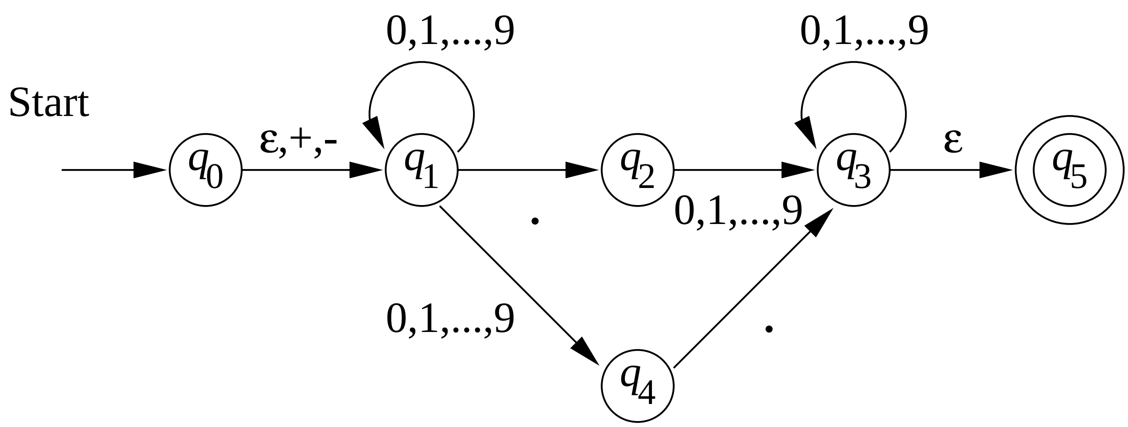
FA's with Epsilon-Transitions

An ϵ-NFA accepting decimal numbers consisting of:

1. An optional $+$ or - sign
2. A string of digits
3. a decimal point
4. another string of digits

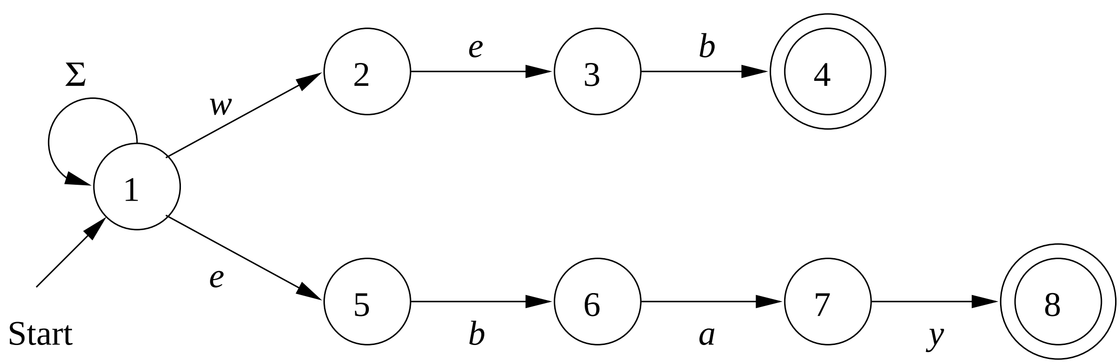
E.g. -12.5
+10.00
5.
-.6

One of the strings (2) are (4) are optional



Example:

ϵ-NFA accepting the set of keywords {ebay, web}



We can have an ε-moves for each keyword.

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.

Example: The ϵ-NFA from the previous slide

$$E = (\{q_0, q_1, \ldots, q_5\}, \{., +, -, 0, 1, \ldots, 9\}\ \delta, q_0, \{q_5\})$$

where the transition table for δ is

	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	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
$\star q_5$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$\boxed{\text{ECLOSE}}$ or ε-closure

We close a state by adding all states reachable by a sequence $\epsilon\epsilon\cdots\epsilon$

Inductive definition of ECLOSE(q)

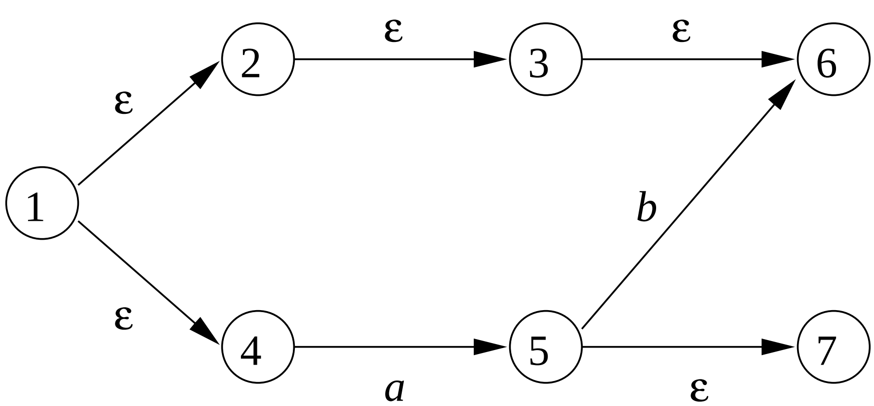
Basis:

$q \in$ ECLOSE(q)

Induction:

$p \in$ ECLOSE(q) and $r \in \delta(p,\epsilon) \;\; \Rightarrow$
$r \in$ ECLOSE(q)

Example of ϵ-closure



For instance,

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

- Inductive definition of $\widehat{\delta}$ for ϵ-NFA's

Basis:

$$\widehat{\delta}(q,\epsilon) = \text{ECLOSE}(q)$$

Induction:

$$\widehat{\delta}(q,xa) = \bigcup_{p \in \delta(\widehat{\delta}(q,x),a)} \text{ECLOSE}(p)$$

where $\delta(\hat{\delta}(q,x),a) = \bigcup_{r \,\varepsilon\, \hat{\delta}(q,x)} \delta(r,a)$

Let's compute on the blackboard in class $\widehat{\delta}(q_0, 5.6)$ for the NFA on slide 43

$\hat{\delta}(q_0,\varepsilon) = \text{ECLOSE}(q_0) = \{q_0,q_1\}$

$\hat{\delta}(q_0,5) = \text{ECLOSE}(\{q_1,q_4\}) = \{q_1,q_4\}$, because $\delta(q_0,5) \cup \delta(q_1,5) = \{q_1,q_4\}$

$\hat{\delta}(q_0,5.) = \text{ECLOSE}(\{q_2,q_3\}) = \{q_2,q_3,q_5\}$

$\hat{\delta}(q_0,5.6) = \text{ECLOSE}(\{q_3\}) = \{q_3,q_5\}$

Given an ϵ-NFA

$$E = (Q_E, \Sigma, \delta_E, q_0, F_E)$$

we will construct a DFA

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

such that

$$L(D) = L(E)$$

Details of the construction:

- $Q_D = \{S : S \subseteq Q_E \text{ and } S = \text{ECLOSE}(S)\}$

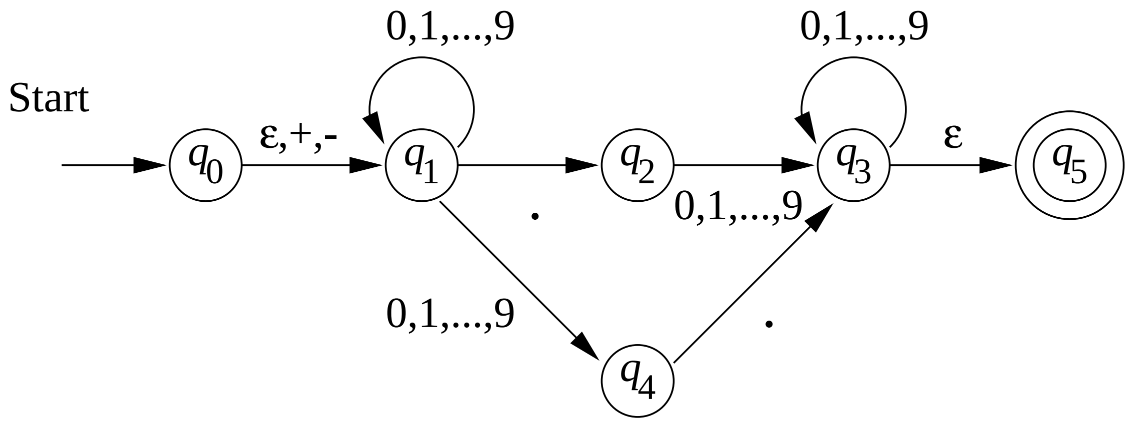
- $q_D = \text{ECLOSE}(q_0)$

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

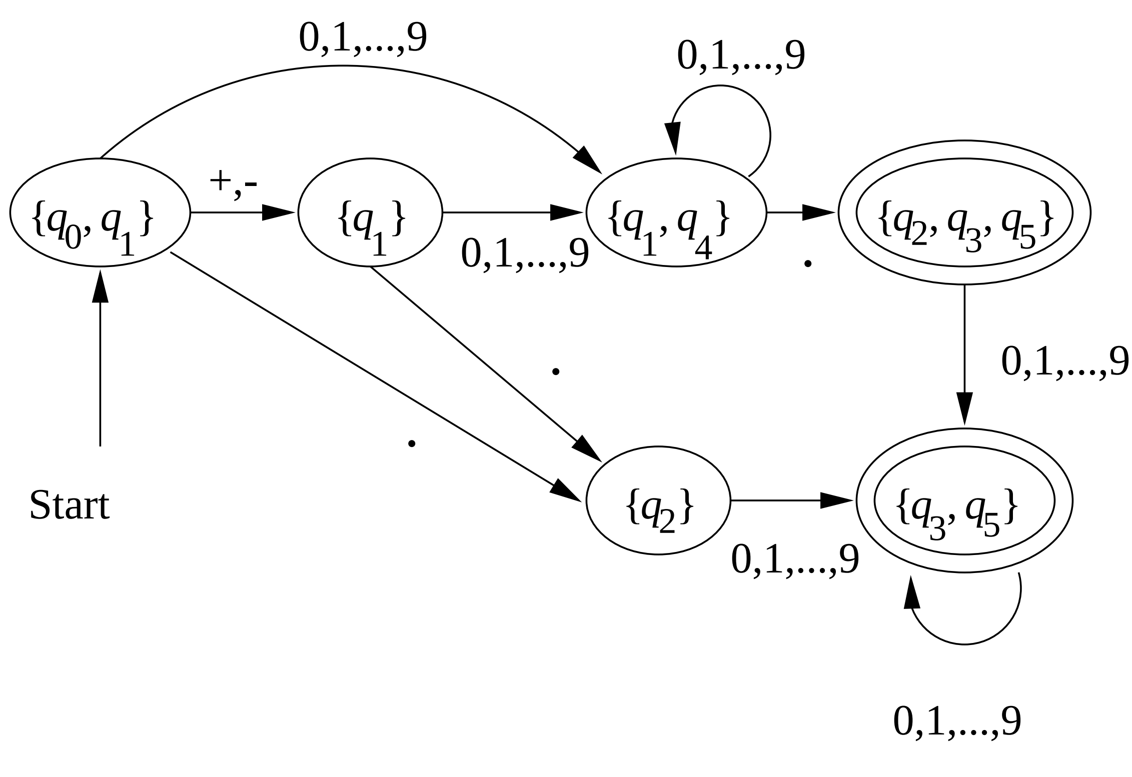
- $\delta_D(S, a) =$

  $\bigcup\{\text{ECLOSE}(p) : p \in \delta_E(t, a) \text{ for some } t \in S\}$

Example: ϵ-NFA E



DFA D corresponding to E



Theorem 2.22: A language L is accepted by some ϵ-NFA E if and only if L is accepted by some DFA.

Proof: We use D constructed as above and show by induction that $\widehat{\delta}_D(q_D, w) = \widehat{\delta}_E(q_0, w)$

Basis: $\widehat{\delta}_E(q_0, \epsilon) = \text{ECLOSE}(q_0) = q_D = \widehat{\delta}(q_D, \epsilon)$

Induction:

$$\hat{\delta}_E(q_0, xa) \stackrel{\text{DEF}}{=} \bigcup_{p \in \delta_E(\hat{\delta}_E(q_0,x),a)} \text{ECLOSE}(p)$$

$$\stackrel{\text{I.H.}}{=} \bigcup_{p \in \delta_E(\hat{\delta}_D(q_D,x),a)} \text{ECLOSE}(p)$$

$$\stackrel{\text{CST}}{=} \delta_D(\hat{\delta}_D(q_D,x),a)$$

$$\stackrel{\text{DEF}}{=} \hat{\delta}_D(q_D, xa)$$

Regular expressions

An FA (NFA or DFA) is a "blueprint" for contructing a machine recognizing a regular language.

A *regular expression* is a "user-friendly," declarative way of describing a regular language.

Example: $\mathbf{01^* + 10^*}$

Regular expressions are used in e.g.

1. UNIX `grep` command

   grep PATTERN FILE

2. UNIX Lex (Lexical analyzer generator) and Flex (Fast Lex) tools.

Operations on languages

Union:

$L \cup M = \{w : w \in L \text{ or } w \in M\}$

Concatenation:

$L.M = \{w : w = xy, x \in L, y \in M\}$

Powers:

$L^0 = \{\epsilon\},\ L^1 = L,\ L^{k+1} = L.L^k$

Kleene Closure:

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

Question: What are $\emptyset^0$, $\emptyset^i$, and $\emptyset^*$

Question: What is $\{0^2,0^3\}^*$?

Building regex's

Inductive definition of regex's:

Basis: $\boldsymbol{\epsilon}$ is a regex and $\boldsymbol{\emptyset}$ is a regex.
$L(\boldsymbol{\epsilon}) = \{\epsilon\}$, and $L(\boldsymbol{\emptyset}) = \emptyset$.

If $a \in \Sigma$, then $\boldsymbol{a}$ is a regex.
$L(\boldsymbol{a}) = \{a\}$.

Induction:

If E is a regex's, then (E) is a regex.
$L((E)) = L(E)$.

If E and F are regex's, then $E + F$ is a regex.
$L(E + F) = L(E) \cup L(F)$.

If E and F are regex's, then $E.F$ is a regex.
$L(E.F) = L(E).L(F)$.

If E is a regex's, then $E^\star$ is a regex.
$L(E^\star) = (L(E))^*$.

Example: Regex for

$L = \{w \in \{0,1\}^* :$ 0 and 1 alternate in $w\}$

$$\mathbf{(01)^* + (10)^* + 0(10)^* + 1(01)^*}$$

or, equivalently,

$$\mathbf{(\boldsymbol{\epsilon} + 1)(01)^*(\boldsymbol{\epsilon} + 0)}$$

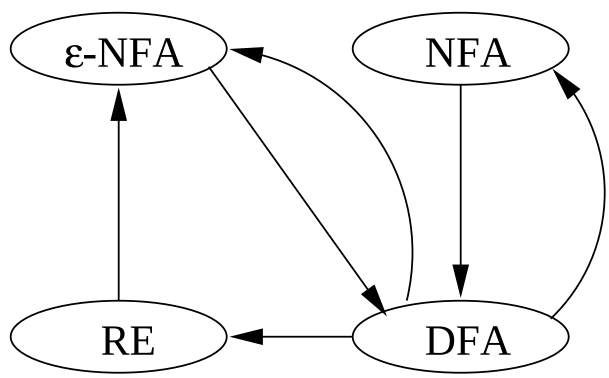
Order of precedence for operators:

1. Star
2. Dot
3. Plus

Example: $\mathbf{01^* + 1}$ is grouped $\mathbf{(0(1^*)) + 1}$

Equivalence of FA's and regex's

We have already shown that DFA's, NFA's, and ϵ-NFA's all are equivalent.



To show FA's equivalent to regex's we need to establish that

1. For every DFA A we can find (construct, in this case) a regex R, s.t. $L(R) = L(A)$.

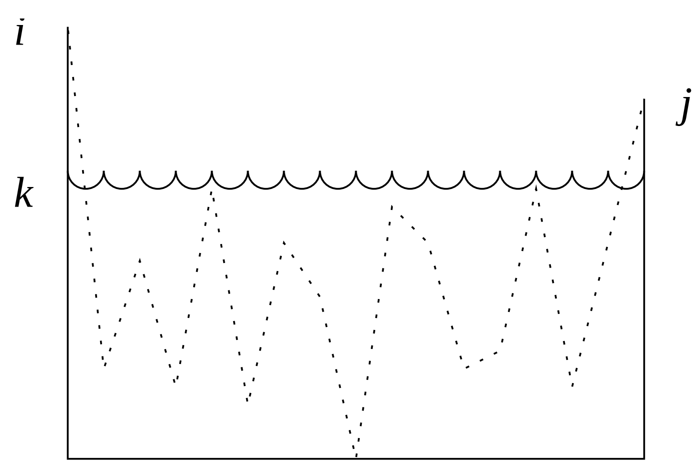
2. For every regex R there is an ϵ-NFA A, s.t. $L(A) = L(R)$.

Theorem 3.4: For every DFA $A = (Q, \Sigma, \delta, q_0, F)$ there is a regex R, s.t. $L(R) = L(A)$.

Proof: Let the states of A be $\{1, 2, \ldots, n\}$, with 1 being the start state.

- Let $R_{ij}^{(k)}$ be a regex describing the set of labels of all paths in A from state i to state j going through intermediate states $\{1, \ldots, k\}$ only.

Note that, i and j don't have to be in {1, ...,k}.



$R_{ij}^{(k)}$ will be defined inductively. Note that

$$L\left(\bigoplus_{j \in F} {R_{1j}}^{(n)}\right) = L(A)$$

Basis: $k = 0$, i.e. no intermediate states.

- *Case 1:* $i \neq j$ $\boxed{\text{i.e., arc i -> j}}$

$$R_{ij}^{(0)} = \bigoplus_{\{a \in \Sigma : \delta(i,a) = j\}} \boldsymbol{a}$$

- *Case 2:* $i = j$ $\boxed{\text{i.e., arc i -> i or } \varepsilon}$

$$R_{ii}^{(0)} = \left(\bigoplus_{\{a \in \Sigma : \delta(i,a) = i\}} \boldsymbol{a}\right) + \boldsymbol{\epsilon}$$

Induction:

$$R_{ij}^{(k)}$$

$$=$$

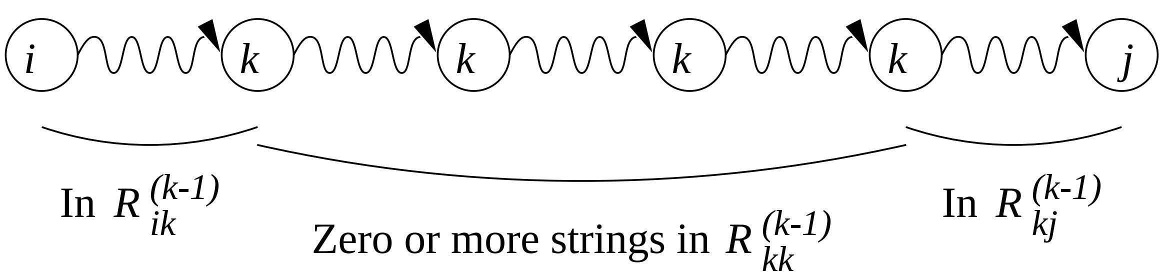
$$R_{ij}^{(k-1)}$$

does not go through k

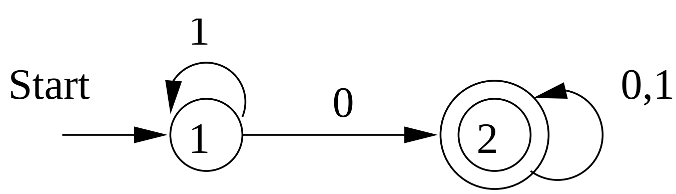
$$+$$

$$R_{ik}^{(k-1)}\left(R_{kk}^{(k-1)}\right)^{*}R_{kj}^{(k-1)}$$

goes through k
at least once



Example: Let's find R for A, where
$L(A) = \{x0y : x \in \{1\}^* \text{ and } y \in \{0,1\}^*\}$



$R_{11}^{(0)}$	$\boldsymbol{\epsilon + 1}$
$R_{12}^{(0)}$	$\mathbf{0}$
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\boldsymbol{\epsilon + 0 + 1}$

We will need the following *simplification rules:*

- $(\epsilon + R)^* = R^*$ (ε+R)R* = R*
- $R + RS^* = RS^*$ ε+R+R* = R*
- $\emptyset R = R\emptyset = \emptyset$ (Annihilation)
- $\emptyset + R = R + \emptyset = R$ (Identity)

$R_{11}^{(0)}$	$\boldsymbol{\epsilon} + \mathbf{1}$
$R_{12}^{(0)}$	$\mathbf{0}$
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1}$

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)}\left(R_{11}^{(0)}\right)^* R_{1j}^{(0)}$$

	By direct substitution	Simplified
$R_{11}^{(1)}$	$\boldsymbol{\epsilon} + \mathbf{1} + (\boldsymbol{\epsilon} + \mathbf{1})(\boldsymbol{\epsilon} + \mathbf{1})^*(\boldsymbol{\epsilon} + \mathbf{1})$	$\mathbf{1}^*$
$R_{12}^{(1)}$	$\mathbf{0} + (\boldsymbol{\epsilon} + \mathbf{1})(\boldsymbol{\epsilon} + \mathbf{1})^*\mathbf{0}$	$\mathbf{1}^*\mathbf{0}$
$R_{21}^{(1)}$	$\emptyset + \emptyset(\boldsymbol{\epsilon} + \mathbf{1})^*(\boldsymbol{\epsilon} + \mathbf{1})$	$\emptyset$
$R_{22}^{(1)}$	$\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1} + \emptyset(\boldsymbol{\epsilon} + \mathbf{1})^*\mathbf{0}$	$\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1}$

	Simplified
$R_{11}^{(1)}$	$\mathbf{1}^*$
$R_{12}^{(1)}$	$\mathbf{1}^*\mathbf{0}$
$R_{21}^{(1)}$	$\emptyset$
$R_{22}^{(1)}$	$\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1}$

$$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)}\left(R_{22}^{(1)}\right)^* R_{2j}^{(1)}$$

	By direct substitution
$R_{11}^{(2)}$	$\mathbf{1}^* + \mathbf{1}^*\mathbf{0}(\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1})^*\emptyset$
$R_{12}^{(2)}$	$\mathbf{1}^*\mathbf{0} + \mathbf{1}^*\mathbf{0}(\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1})^*(\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1})$
$R_{21}^{(2)}$	$\emptyset + (\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1})(\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1})^*\emptyset$
$R_{22}^{(2)}$	$\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1} + (\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1})(\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1})^*(\boldsymbol{\epsilon} + \mathbf{0} + \mathbf{1})$

	By direct substitution
$R_{11}^{(2)}$	$\mathbf{1^* + 1^*0(\epsilon + 0 + 1)^*\emptyset}$
$R_{12}^{(2)}$	$\mathbf{1^*0 + 1^*0(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)}$
$R_{21}^{(2)}$	$\mathbf{\emptyset + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*\emptyset}$
$R_{22}^{(2)}$	$\mathbf{\epsilon + 0 + 1 + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)}$

	Simplified
$R_{11}^{(2)}$	$\mathbf{1^*}$
$R_{12}^{(2)}$	$\mathbf{1^*0(0 + 1)^*}$
$R_{21}^{(2)}$	$\mathbf{\emptyset}$
$R_{22}^{(2)}$	$\mathbf{(0 + 1)^*}$

The final regex for A is

$$R_{12}^{(2)} = \mathbf{1^*0(0 + 1)^*}$$

Observations

There are n^3 expressions $R_{ij}^{(k)}$

Each inductive step grows the expression 4-fold

$R_{ij}^{(n)}$ could have size 4^n

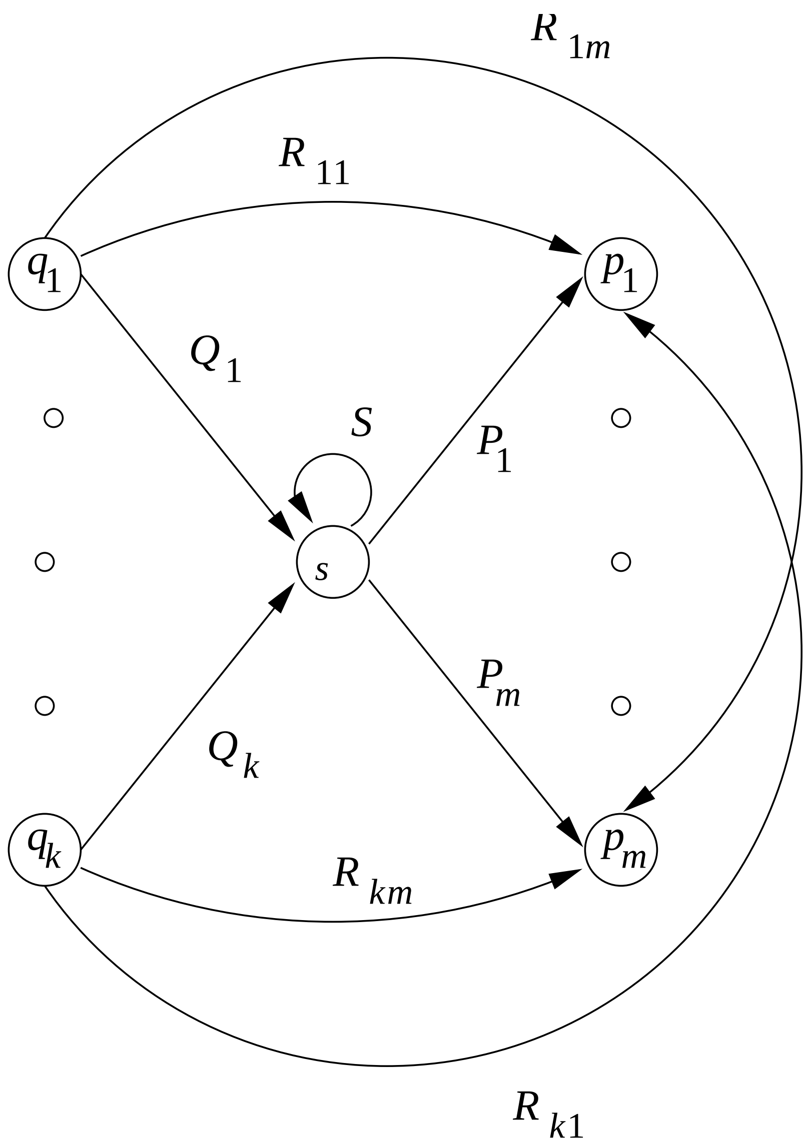
For all $\{i, j\} \subseteq \{1, \ldots, n\}$, $R_{ij}^{(k)}$ uses $R_{kk}^{(k-1)}$
so we have to write n^2 times the regex $R_{kk}^{(k-1)}$

but most of them can be removed by annihilation!

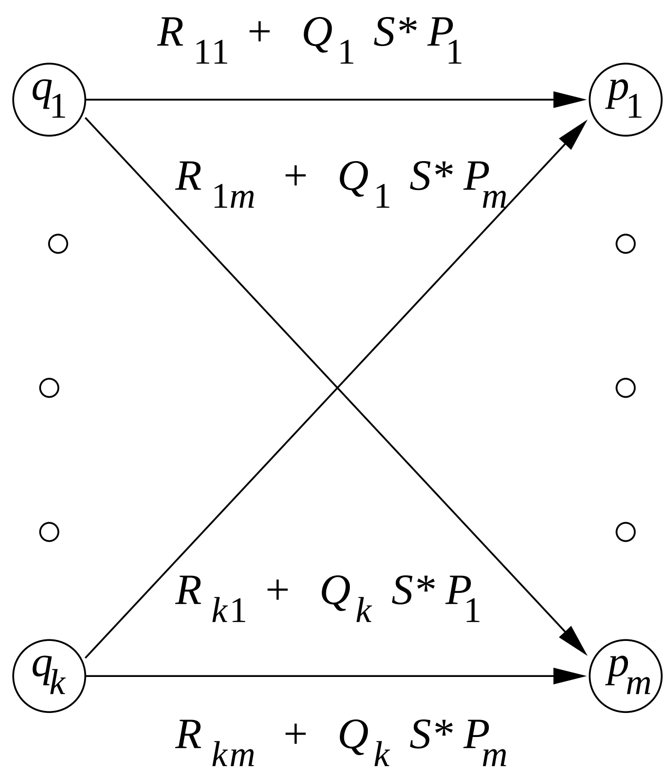
We need a more efficient approach:
the state elimination technique

The state elimination technique

Let's label the edges with regex's instead of symbols

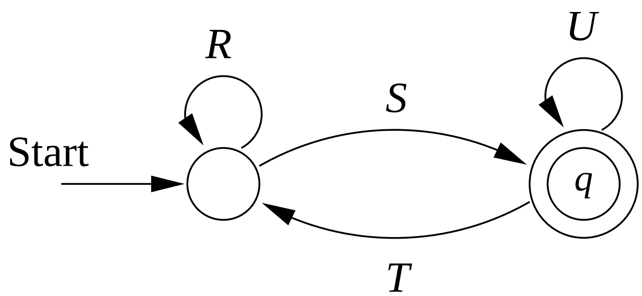


Now, let's eliminate state s.



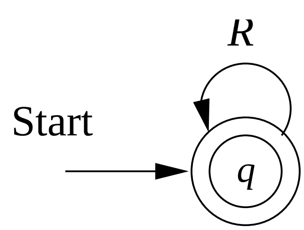
For each accepting state q eliminate from the original automaton all states exept q_0 and q.

For each $q \in F$ we'll be left with an A_q that looks like



that corresponds to the regex $E_q = (R + SU^*T)^*SU^*$

or with A_q looking like



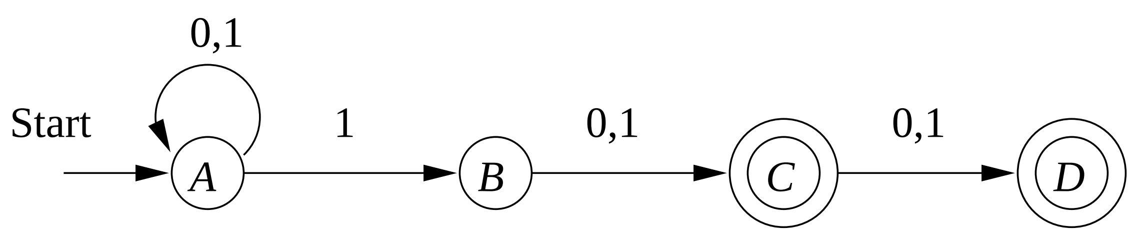
corresponding to the regex $E_q = R^*$

- The final expression is

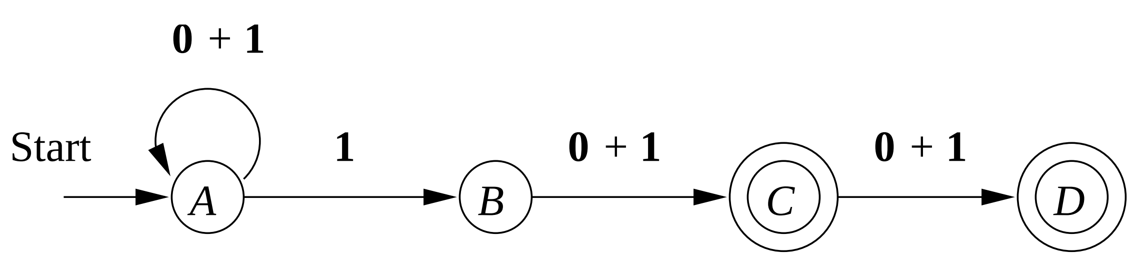
$$\bigoplus_{q \in F} E_q$$

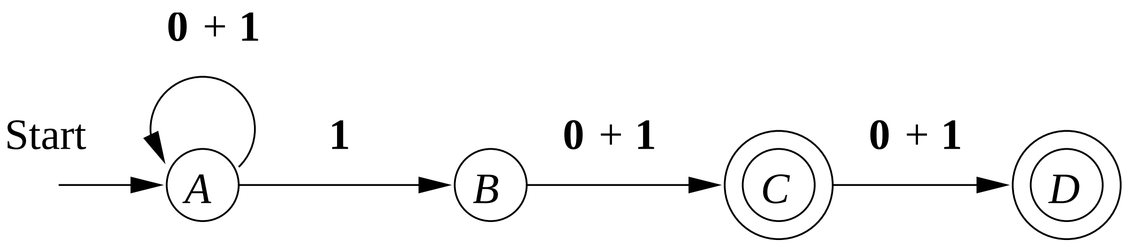
Note that the algorithm also works for NFAs.

Example: $\mathcal{A}$, where $L(\mathcal{A}) = \{W : w = x1b,$ or $w = x1bc,\ \ x \in \{0,1\}^*, \{b,c\} \subseteq \{0,1\}\}$

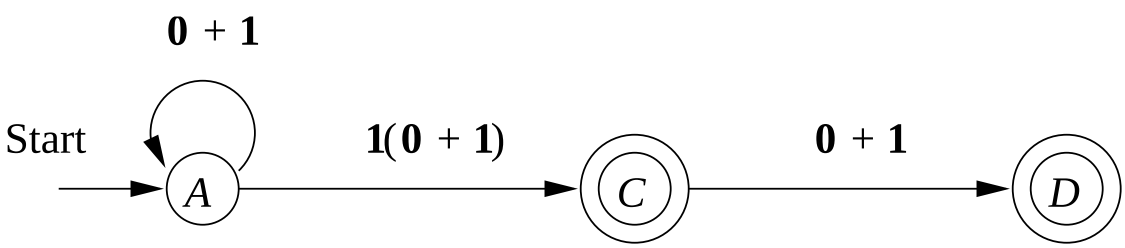


We turn this into an automaton with regex labels

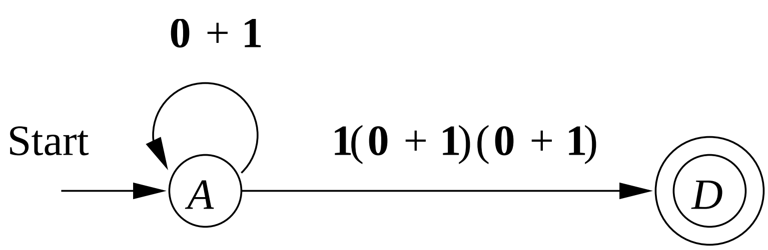




Let's eliminate state B

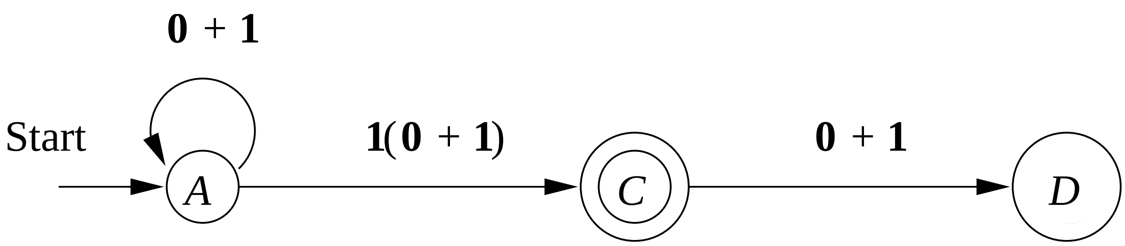


Then we eliminate state C and obtain $\mathcal{A}_D$

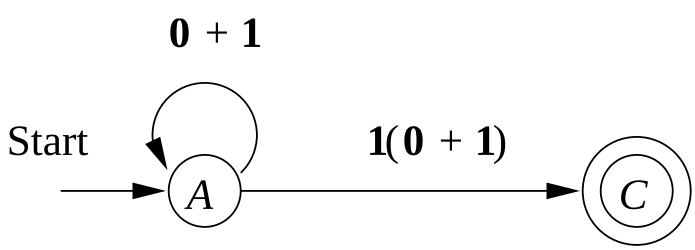


with regex $(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})$

From



we can eliminate D to obtain $\mathcal{A}_C$



with regex $(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$

- The final expression is the sum of the previous two regex's:

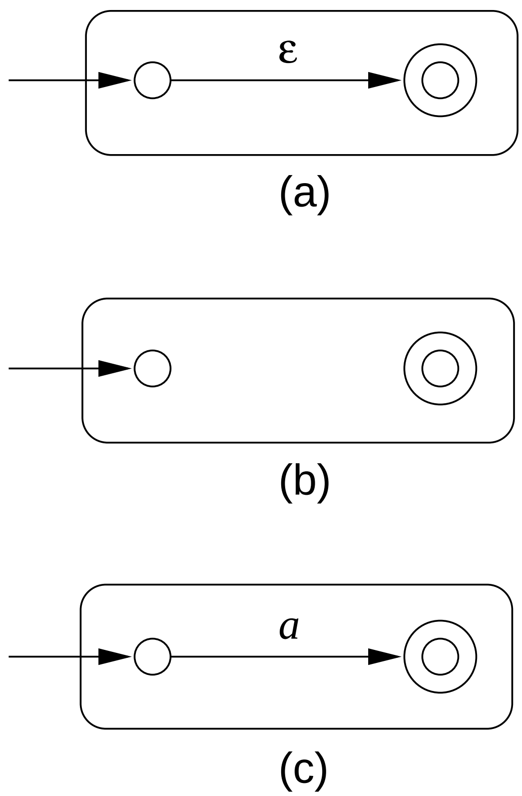
$$(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})+(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$$

From regex's to ϵ-NFA's

Theorem 3.7: For every regex R we can construct an ϵ-NFA A, s.t. $L(A) = L(R)$.

Proof: By structural induction:

Basis: Automata for ϵ, $\emptyset$, and $\boldsymbol{a}$.



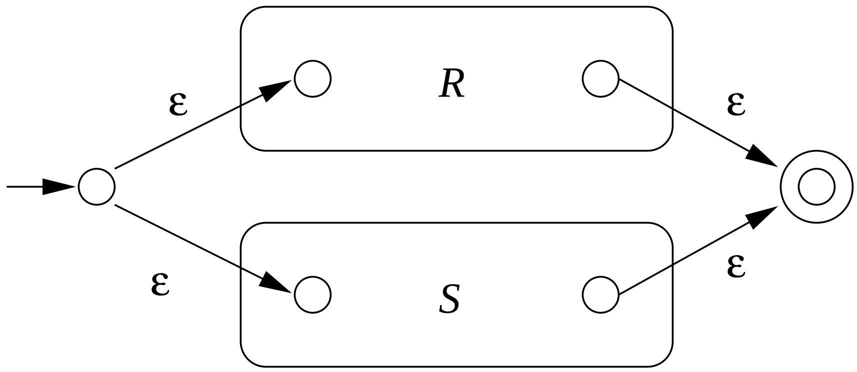
ε-NFAs with properties:

* unique start and final states

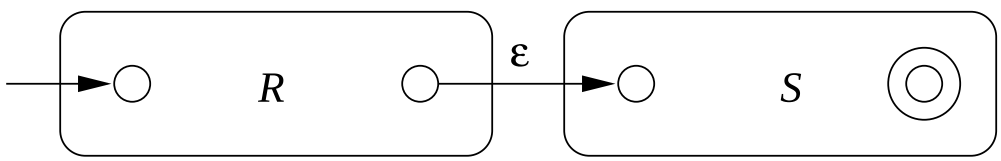
* no arcs into the start state

* no arcs out of the final state

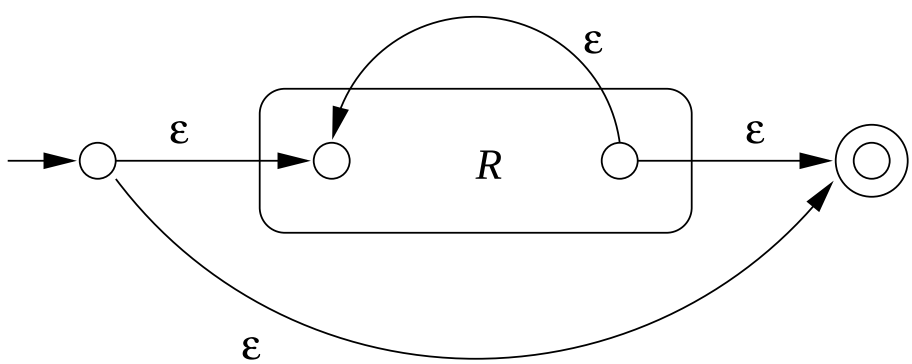
Induction: Automata for $R + S$, RS, and R^*



(a)

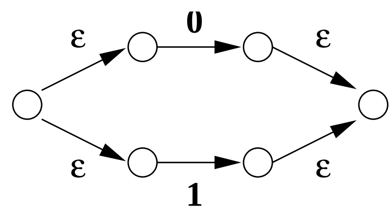


(b)

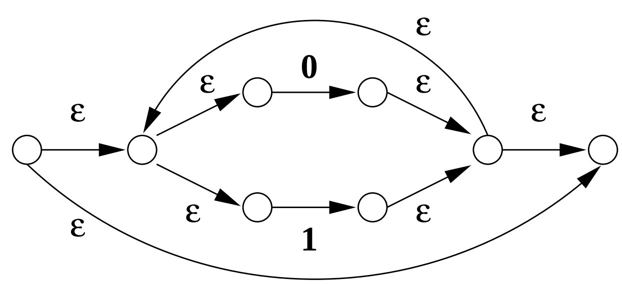


(c)

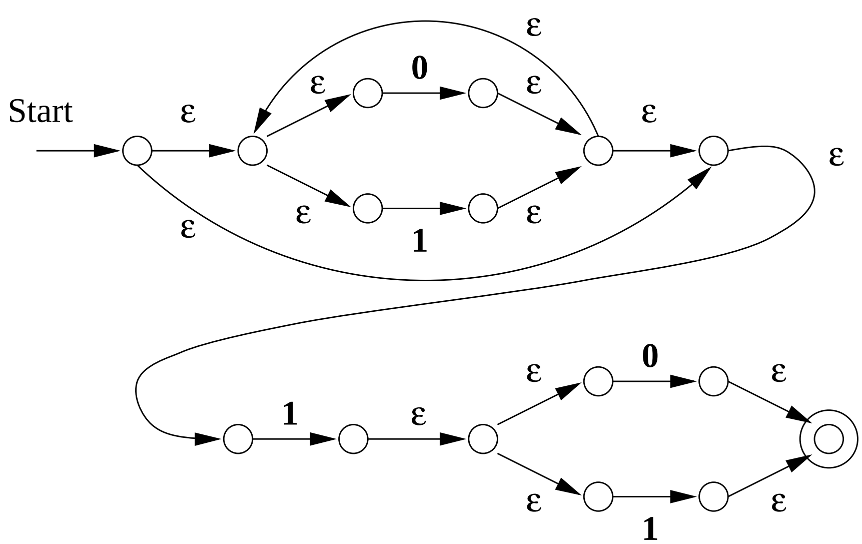
Example: We convert $(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$



(a)



(b)



(c)