15-453

FORMAL LANGUAGES, AUTOMATA AND COMPUTABILITY

FIRST HOMEWORK IS DUE
Thursday, January 22

NON-DETERMINISM

THURSDAY JAN 18

NON-DETERMINISM AND THE PUMPING LEMMA

THURSDAY JAN 18

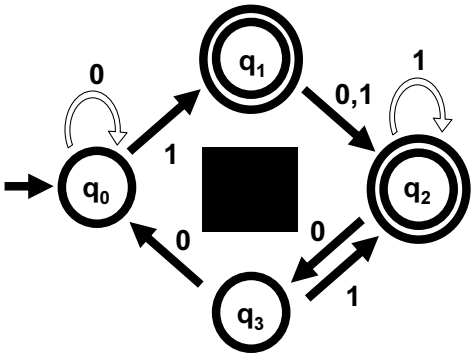
M = (Q, Σ, δ, q_0, F) where **Q = {q_0, q_1, q_2, q_3}**

Σ = {0,1}

δ : Q × Σ → Q transition function*

q_0 ∈ Q is start state

F = {q_1, q_2} ⊆ Q accept states



*

δ	0	1
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_3	q_2
q_3	q_0	q_2

deterministic DFA

A ^ finite automaton ^ is a 5-tuple M = (Q, Σ, δ, q_0, F)

Q is the set of states (finite)

Σ is the alphabet (finite)

δ : Q × Σ → Q is the transition function

q_0 ∈ Q is the start state

F ⊆ Q is the set of accept states

M accepts a string w if the process ends in a double circle

deterministic DFA

A ^ finite automaton ^ is a 5-tuple M = (Q, Σ, δ, q_0, F)

Q is the set of states (finite)

Σ is the alphabet (finite)

δ : Q × Σ → Q is the transition function

q_0 ∈ Q is the start state

F ⊆ Q is the set of accept states

Let w_1, ... , w_n ∈ **Σ** and **w** = w_1... w_n ∈ **Σ***
Then **M accepts w** if there are r_0, r_1, ..., r_n ∈ **Q,** s.t.

1. r_0=**q_0**
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for i = 0, ..., n-1, and
3. r_n ∈ F

deterministic DFA

A ^ finite automaton ^ is a 5-tuple M = (Q, Σ, δ, q_0, F)

Q is the set of states (finite)

Σ is the alphabet (finite)

δ : Q × Σ → Q is the transition function

q_0 ∈ Q is the start state

F ⊆ Q is the set of accept states

L(M) = set of all strings machine M accepts

A language L is regular if it is recognized by a deterministic finite automaton, i.e. if there is a DFA M such that L = L (M).

UNION **THEOREM**

The union of two regular languages is also a regular language

Intersection **THEOREM**

The intersection of two regular languages is also a regular language

Complement **THEOREM**

The complement of a regular languages is also a regular language

In other words,

if L is regular than so is $\neg$L,

where $\neg L = \{ w \in \Sigma^* \mid w \notin L \}$

Proof ?

THE REGULAR OPERATIONS

- **Union: $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$**
- **Intersection: $A \cap B = \{ w \mid w \in A \text{ and } w \in B \}$**
- **Negation: $\neg A = \{ w \in \Sigma^* \mid w \notin A \}$**

Reverse: $A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \}$

Concatenation: $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

Star: $A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \}$

Reverse **THEOREM**

The reverse of a regular languages is also a regular language

REVERSE CLOSURE

Regular languages are closed under reverse

Assume L is a regular language and M recognizes L

We build M^R that accepts L^R

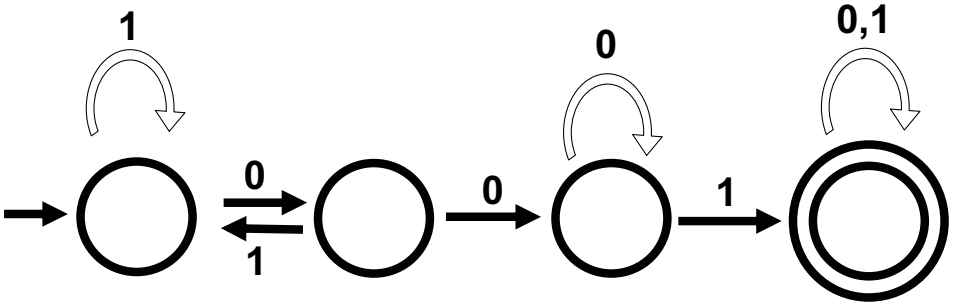
If M accepts w then w describes a directed path in M from start to an accept state

Define M^R as M with the arrows reversed

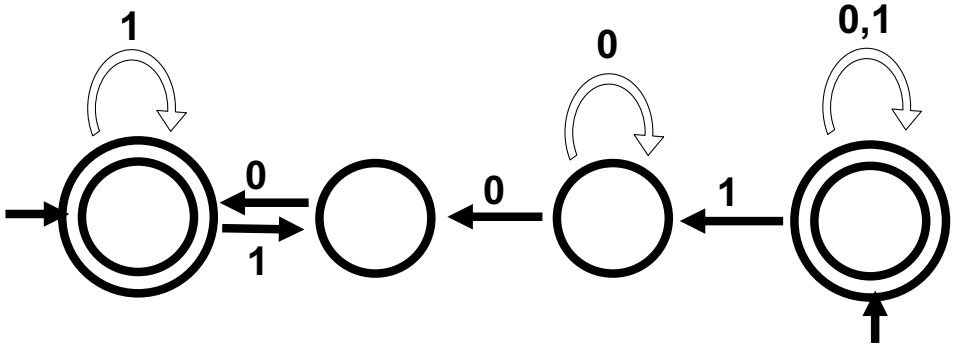
M^R IS NOT ALWAYS A DFA!

It may have many start states

Some states may have too many outgoing edges, or none



NON-DETERMINISM



What happens with 100?

We will say that the machine accepts if there is some way to make it reach an accept state

IBM JOURNAL APRIL 1959

Turing Award winning paper

M. O. Rabin*
D. Scott†

Finite Automata and Their Decision Problems‡

Abstract: Finite automata are considered in this paper as instruments for classifying finite tapes. Each one-tape automaton defines a set of tapes, a two-tape automaton defines a set of pairs of tapes, et cetera. The structure of the defined sets is studied. Various generalizations of the notion of an automaton are introduced and their relation to the classical automata is determined. Some decision problems concerning automata are shown to be solvable by effective algorithms; others turn out to be unsolvable by algorithms.

Introduction

Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an *a priori* upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.

In the last few years the idea of a *finite automaton* has appeared in the literature. These are machines having a method of viewing automata but have retained throughout a machine-like formalism that permits direct comparison with Turing machines. A neat form of the definition of automata has been used by Burks and Wang[1] and by E. F. Moore,[4] and our point of view is closer to theirs than it is to the formalism of nerve-nets. However, we have adopted an even simpler form of the definition by doing away with a complicated output function and having our machines simply give "yes" or "no" answers. This was also used by Myhill, but our generalizations to the "nondeterministic," "two-way," and "many-tape"

...construction of $\mathfrak{A}$ and we shall ...tail.

...ces of words $S_1=(a_1,a_2,\ldots,a_n)$ ... $b_n)$ then $P(a_1,a_2,\ldots,a_n)\cap P(b_1,$ ...d only if the Post correspondence ...$_2$ has a solution. Since the corre- ... not effectively solvable *it* follows ...ther

$T_2(\mathfrak{A}(b_1,\ldots,b_n))\neq\phi$

...ible.

...tape automata

...-way, two-tape automata we find ... constructive decision processes is ...sible to decide, by a constructive ...icable to all automata, whether a ...chine accepts any tapes. To prove ...ourse, necessary to give the explicit ...y machine. We shall not give the ... are long and not very much dif... ...l definitions needed for two-way ...he main point is that, as with the ...omaton, the table of moves of a ...utomaton sometimes requires the ...om the scanned square. However, ...should clarify the method.

...that there is no constructive deci-

Theorem 19. *There is no effective method of deciding whether the set of tapes definable by a two-tape, two-way automaton is empty or not.*

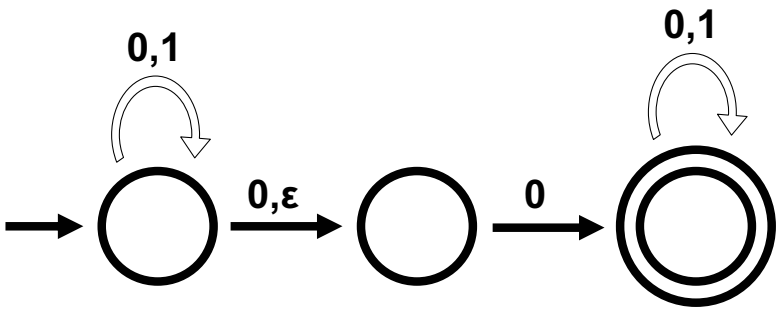
An argument similar to the above one will show that the class of sets of pairs of tapes definable by two-way, two-tape automata is closed under Boolean operations. In view of Theorem 17, this implies that there are sets definable by two-way automata which are not definable by any one-way automaton; thus no analogue to Theorem 15 holds.

References

1. A. W. Burks and Hao Wang, "The logic of automata," *Journal of the Association for Computing Machinery*, **4**, 193-218 and 279-297 (1957).
2. S. C. Kleene, "Representation of events in nerve nets and finite automata," *Automata Studies*, Princeton, pp. 3-41, (1956).
3. W. S. McCulloch and E. Pitts, "A logical calculus of the ideas imminent in nervous activity," *Bulletin of Mathematical Biophysics*, **5**, 115-133 (1943).
4. E. F. Moore, "Gedanken-experiments on sequential machines," *Automata Studies*, Princeton, pp. 129-153 (1956).
5. A. Nerode, "Linear automaton transformations," *Proceedings of the American Mathematical Society*, **9**, 541-544 (1958).
6. E. Post, "A variant of a recursively unsolvable problem," *Bulletin of the American Mathematical Society*, **52**, 264-268 (1946).
7. J. C. Shepherdson, "The reduction of two-way automata to one-way automata," *IBM Journal*, **3**, 198-200 (1959).

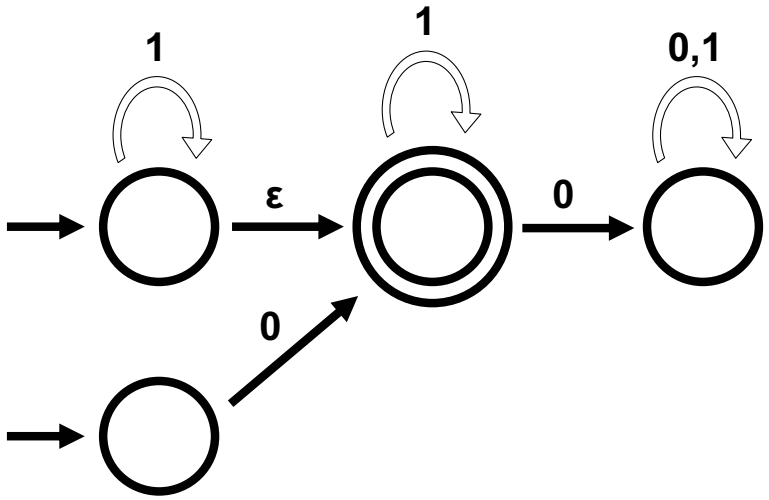
Revised manuscript received August 8, 1958

EXAMPLE



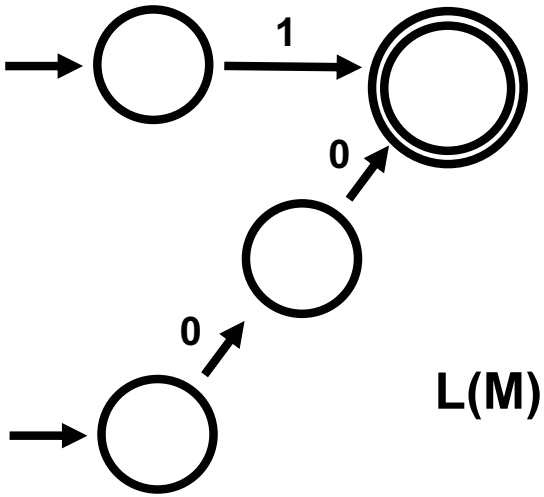
At each state, possibly zero, one or many out arrows for each $\sigma \in \Sigma$ or with label ε

EXAMPLE



Possibly many start states

EXAMPLE



L(M)={1,00}

A non-deterministic finite automaton (NFA) is a 5-tuple $N = (Q, \Sigma, \delta, Q_0, F)$

Q is the set of states

Σ is the alphabet

$\delta : Q \times \Sigma_\varepsilon \rightarrow 2^Q$ is the transition function

$Q_0 \subseteq Q$ is the set of start states

$F \subseteq Q$ is the set of accept states

2^Q is the set of subsets of Q and $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

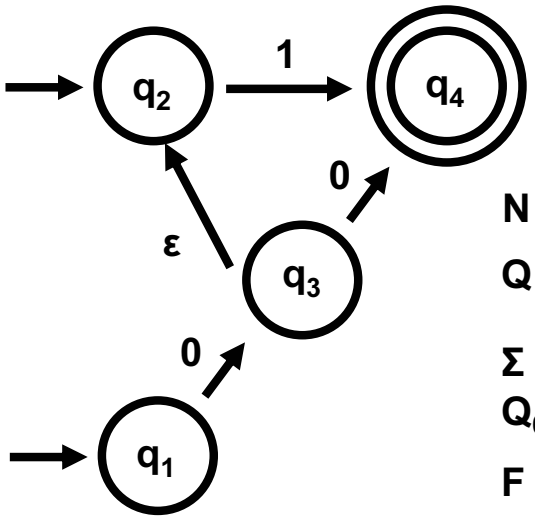
Let $w \in \Sigma^*$ and suppose w can be written as $w_1 \ldots w_n$ where $w_i \in \Sigma_\varepsilon$ (ε is viewed as representing the empty string)

Then N accepts w if there are $r_0, r_1, \ldots, r_n \in Q$ such that

1. $r_0 \in Q_0$
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for i = 0, ..., n-1, and
3. $r_n \in F$

L(N) = the language recognized by N
= set of all strings machine N accepts

A language L is recognized by an NFA N if L = L (N).



$N = (Q, \Sigma, \delta, Q_0, F)$

$Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0,1\}$

$Q_0 = \{q_1, q_2\}$

$F = \{q_4\} \subseteq Q$

$\delta(q_2,1) = \{q_4\}$

$\delta(q_3,1) = \varnothing$

$\delta(q_1,0) = \{q_3\}$

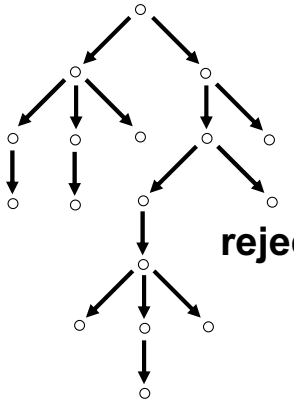
$00 \in L(N)$?

$01 \in L(N)$?

Deterministic Computation

accept or reject

Non-Deterministic Computation

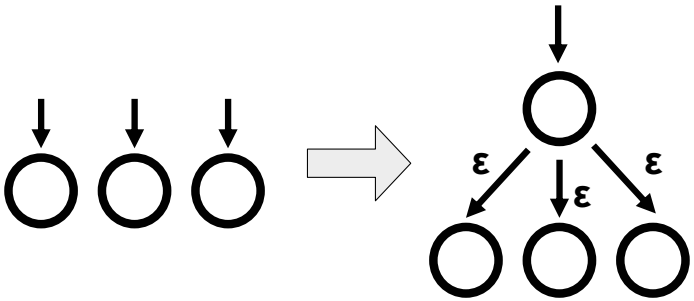


accept

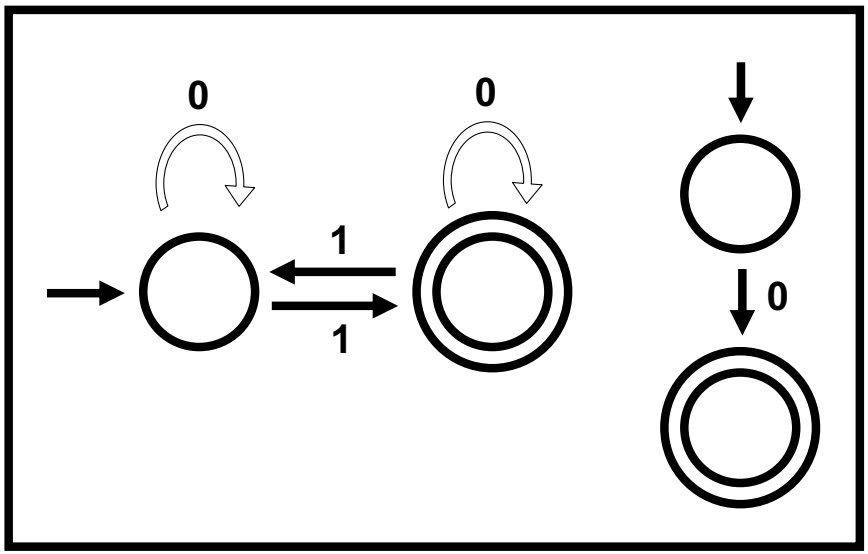
MULTIPLE START STATES

We allow multiple start states for NFAs, and Sipser allows only one

Can easily convert NFA with many start states into one with a single start state:

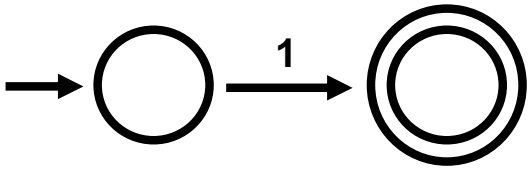


UNION THEOREM FOR NFAs

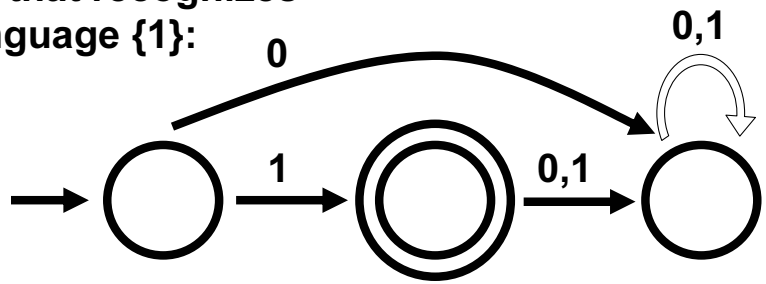


NFAs ARE SIMPLER THAN DFAs

An NFA that recognizes the language {1}:



A DFA that recognizes the language {1}:



Theorem: Every NFA has an equivalent* DFA

Corollary: A language is regular iff it is recognized by an NFA

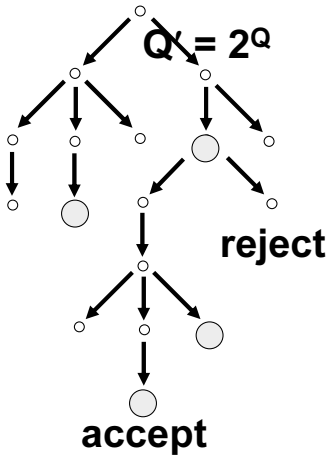
Corollary: L is regular iff L^R is regular

*** N is equivalent to M if L(N) = L (M)**

FROM NFA TO DFA

Input: N = (Q, Σ, δ, Q_0, F)

Output: M = (Q′, Σ, δ′, q_0′, F′)



To learn if NFA accepts, we could do the computation in parallel, maintaining the set of states where all threads are

Idea:

$Q' = 2^Q$

FROM NFA TO DFA

Input: N = (Q, Σ, δ, Q_0, F)

Output: M = (Q′, Σ, δ′, q_0′, F′)

$Q' = 2^Q$

$\delta' : Q' \times \Sigma \rightarrow Q'$

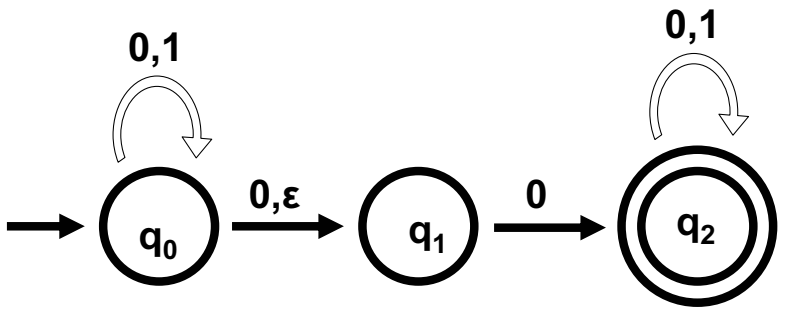
$\delta'(R,\sigma) = \bigcup_{r \in R} \varepsilon(\delta(r,\sigma))$ *

$q_0' = \varepsilon(Q_0)$

$F' = \{ R \in Q' \mid f \in R \text{ for some } f \in F \}$

* For **R** ⊆ **Q,** the **ε-closure of R, ε(R)** = **{q** that can be reached from some r ∈ **R** by traveling along zero or more **ε** arrows},

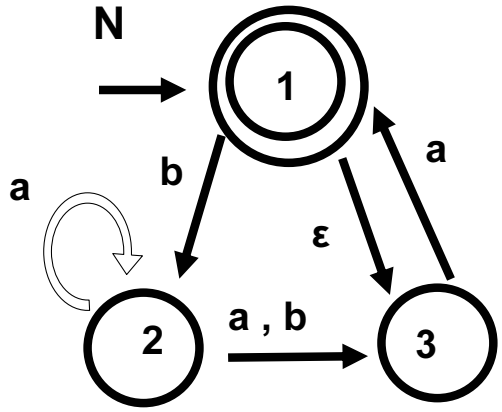
EXAMPLE



$\varepsilon(\{q_0\}) = \{q_0, q_1\}$

Given: NFA N = ({1,2,3}, {a.b}, δ , {1}, {1})

Construct: equivalent DFA M

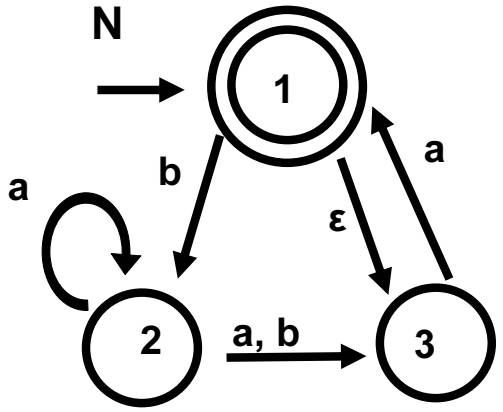


ε({1}) = {1,3}

$N = (Q, \Sigma, \delta, Q_0, F)$

Given: NFA N = ({1,2,3}, {a,b}, δ , {1}, {1})

Construct: equivalent DFA M = (Q′, Σ, δ′, q_0′, F′)



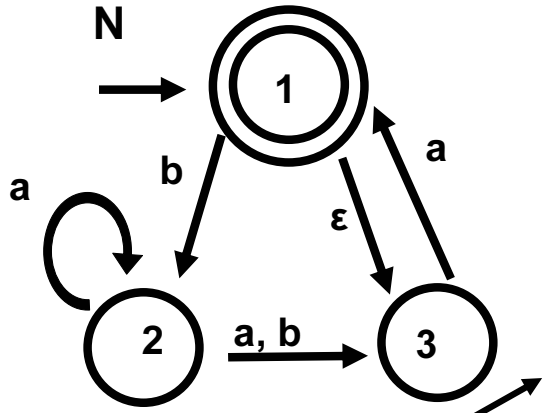
ε({1}) = {1,3}

δ′	a	b
∅		
{1}		
{2}		
{3}		

$N = (Q, \Sigma, \delta, Q_0, F)$

Given: NFA N = ({1,2,3}, {a,b}, δ , {1}, {1})

Construct: equivalent DFA M = (Q′, Σ, δ′, q_0′, F′)



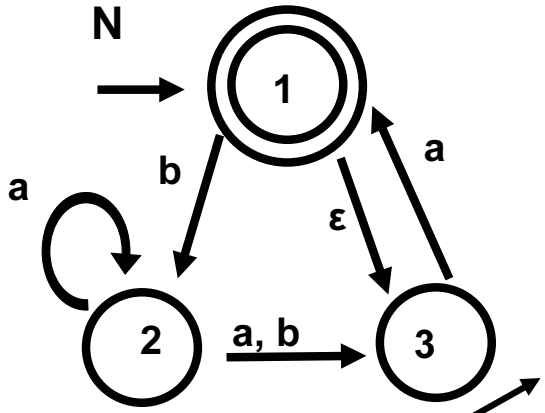
$q_0' = \varepsilon(\{1\}) = \{1,3\}$

δ′	a	b
∅		
{1}		
{2}		
{3}		
{1,2}		
{1,3}		
{2,3}		
{1,2,3}		

$N = (Q, \Sigma, \delta, Q_0, F)$

Given: NFA N = ({1,2,3}, {a,b}, δ , {1}, {1})

Construct: equivalent DFA M = (Q′, Σ, δ′, q_0′, F′)



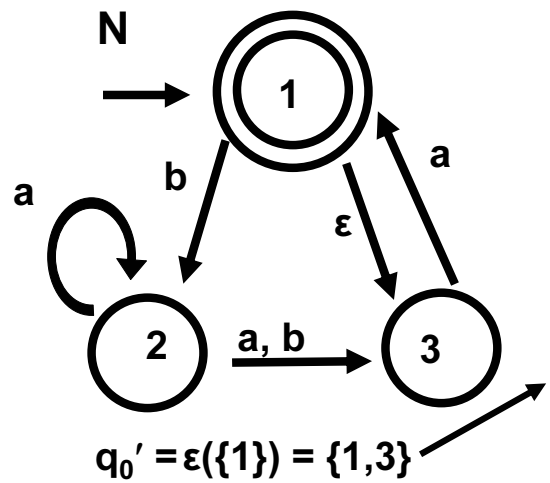
$q_0' = \varepsilon(\{1\}) = \{1,3\}$

δ′	a	b
∅		
{1}		
{2}		
{3}		
{1,2}		
{1,3}		
{2,3}		
{1,2,3}		

$N = (Q, \Sigma, \delta, Q_0, F)$

Given: NFA $N = (\{1,2,3\}, \{a,b\}, \delta , \{1\}, \{1\})$

Construct: equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$

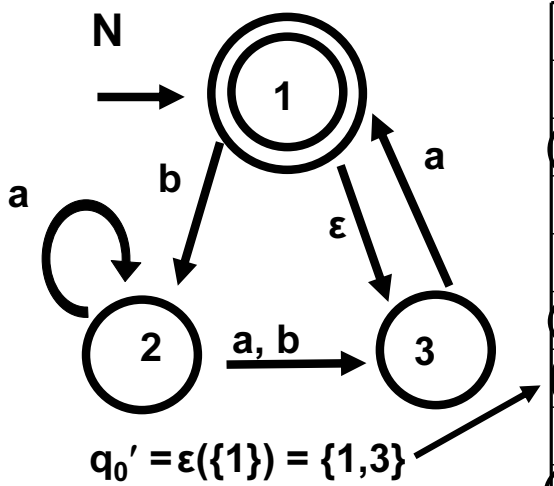


δ'	a	b
∅	∅	∅
{1}		
{2}		
{3}		
{1,2}		
{1,3}		
{2,3}		
{1,2,3}		

$N = (Q, \Sigma, \delta, Q_0, F)$

Given: NFA $N = (\{1,2,3\}, \{a,b\}, \delta , \{1\}, \{1\})$

Construct: equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$

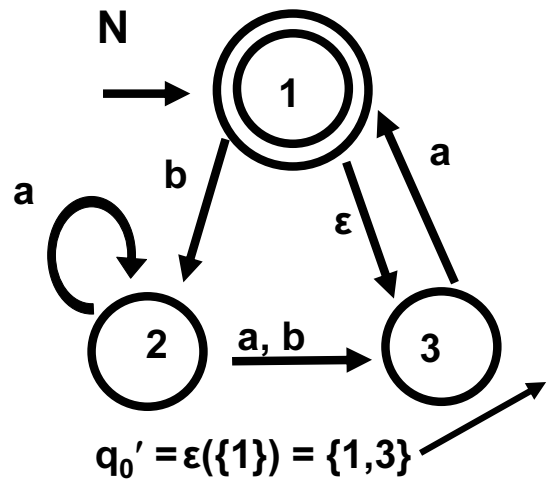


δ'	a	b
∅	∅	∅
{1}	∅	{2}
{2}		
{3}		
{1,2}		
{1,3}		
{2,3}		
{1,2,3}		

$N = (Q, \Sigma, \delta, Q_0, F)$

Given: NFA $N = (\{1,2,3\}, \{a,b\}, \delta , \{1\}, \{1\})$

Construct: equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$

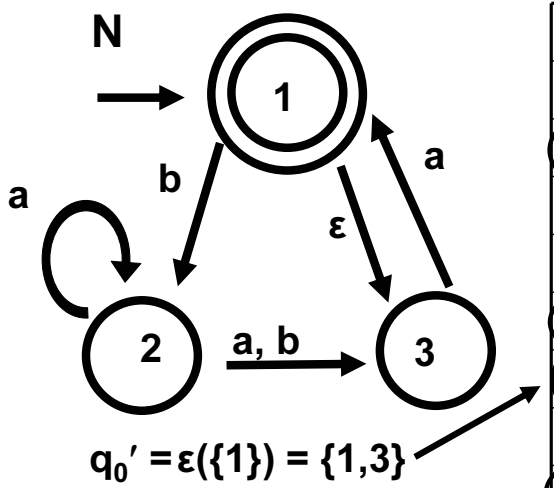


δ'	a	b
∅	∅	∅
{1}	∅	{2}
{2}	{2,3}	{3}
{3}		
{1,2}		
{1,3}		
{2,3}		
{1,2,3}		

$N = (Q, \Sigma, \delta, Q_0, F)$

Given: NFA $N = (\{1,2,3\}, \{a,b\}, \delta , \{1\}, \{1\})$

Construct: equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$

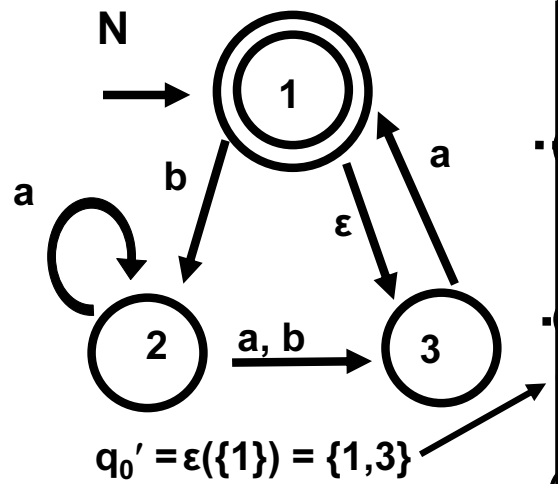


δ'	a	b
∅	∅	∅
{1}	∅	{2}
{2}	{2,3}	{3}
{3}	{1,3}	∅
{1,2}	{2,3}	{2,3}
{1,3}	{1,3}	{2}
{2,3}	{1,2,3}	{3}
{1,2,3}	{1,2,3}	{2,3}

$N = (Q, \Sigma, \delta, Q_0, F)$

Given: NFA $N = (\{1,2,3\}, \{a,b\}, \delta , \{1\}, \{1\})$

Construct: equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$



$q_0' = \varepsilon(\{1\}) = \{1,3\}$

δ'	a	b
∅	∅	∅
{1}	∅	{2}
{2}	{2,3}	{3}
{3}	{1,3}	∅
{1,2}	{2,3}	{2,3}
{1,3}	{1,3}	{2}
{2,3}	{1,2,3}	{3}
{1,2,3}	{1,2,3}	{2,3}

FROM NFA TO DFA

Input: $N = (Q, \Sigma, \delta, Q_0, F)$

Output: $M = (Q', \Sigma, \delta', q_0', F')$

$Q' = 2^Q$

$\delta' : Q' \times \Sigma \rightarrow Q'$

$\delta'(R,\sigma) = \bigcup_{r \in R} \varepsilon(\delta(r,\sigma))$ *

$q_0' = \varepsilon(Q_0)$

$F' = \{ R \in Q' \mid f \in R \text{ for some } f \in F \}$

* For $R \subseteq Q$, the **ε-closure of R**, **ε(R)** = {**q** that can be reached from **R** by traveling along zero or more **ε** arrows},

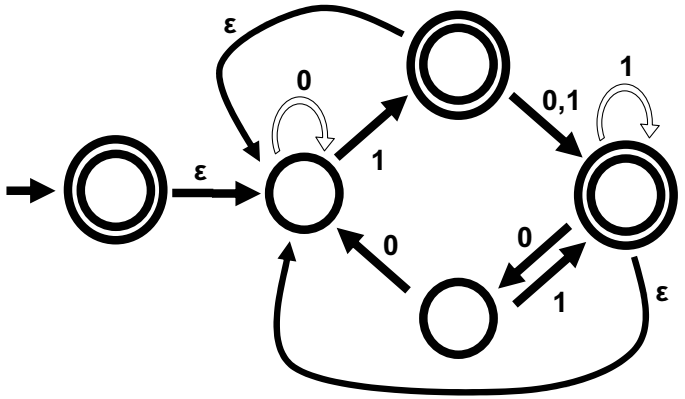
REGULAR LANGUAGES CLOSED UNDER CONCATENATION

Given DFAs M_1 and M_2, construct NFA by connecting all accept states in M_1 to start states in M_2

REGULAR LANGUAGES CLOSED UNDER STAR

Let L be a regular language and M be a DFA for L

We construct an NFA N that recognizes L*



Formally:

Input: $M = (Q, \Sigma, \delta, q_1, F)$

Output: $N = (Q', \Sigma, \delta', \{q_0\}, F')$

$$Q' = Q \cup \{q_0\}$$

$$F' = F \cup \{q_0\}$$

$$\delta'(q,a) = \begin{cases} \{\delta(q,a)\} & \text{if } q \in Q \text{ and } a \neq \varepsilon \\ \{q_1\} & \text{if } q \in F \text{ and } a = \varepsilon \\ \{q_1\} & \text{if } q = q_0 \text{ and } a = \varepsilon \\ \varnothing & \text{if } q = q_0 \text{ and } a \neq \varepsilon \\ \varnothing & \text{else} \end{cases}$$

L(N) = L*

Assume $w = w_1 \ldots w_k$ is in L*, where $w_1, \ldots, w_k \in L$

We show N accepts w by induction on k

Base Cases:

- ✓ **k = 0**
- ✓ **k = 1**

Inductive Step:

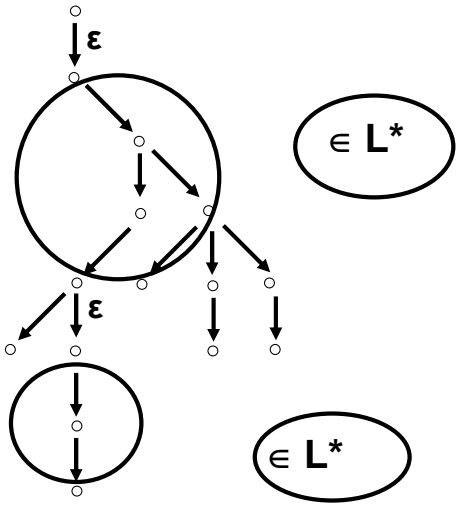
Assume N accepts all strings $v = v_1 \ldots v_k \in L^*$, $v_i \in L$, and let $u = u_1 \ldots u_k u_{k+1} \in L^*$, $u_j \in L$,

Since N accepts $u_1 \ldots u_k$ and M accepts u_{k+1}, N must accept u

Assume w is accepted by N, we show $w \in L^*$

If $w = \varepsilon$, then $w \in L^*$

If $w \neq \varepsilon$

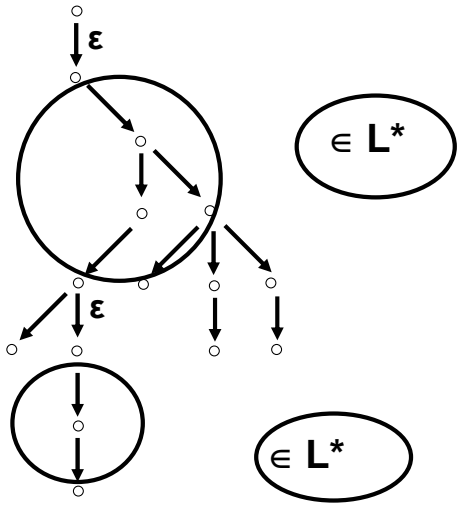


accept

Assume w is accepted by N, we show $w \in L^*$

If $w = \varepsilon$, then $w \in L^*$

If $w \neq \varepsilon$



accept

REGULAR LANGUAGES ARE COLSED UNDER **REGULAR OPERATIONS**

- **Union: $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$**
- **Intersection: $A \cap B = \{ w \mid w \in A \text{ and } w \in B \}$**
- **Negation: $\neg A = \{ w \in \Sigma^* \mid w \notin A \}$**
- **Reverse: $A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \}$**
- **Concatenation: $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$**
- **Star: $A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \}$**

SOME LANGUAGES **ARE NOT** REGULAR

$B = \{0^n 1^n \mid n \geq 0\}$ is NOT regular!

WHICH OF THESE ARE REGULAR

C = { w | w has equal number of 1s and 0s}

NOT REGULAR

D = { w | w has equal number of occurrences of 01 and 10}

REGULAR!!!

THE PUMPING LEMMA

Let L be a regular language with $|L| = \infty$

Then there exists a positive integer P such that

if $w \in L$ and $|w| \geq P$

then w = xyz, where:

1. **$|y| > 0$**
2. **$|xy| \leq P$**
3. **$xy^iz \in L$ for any $i \geq 0$**

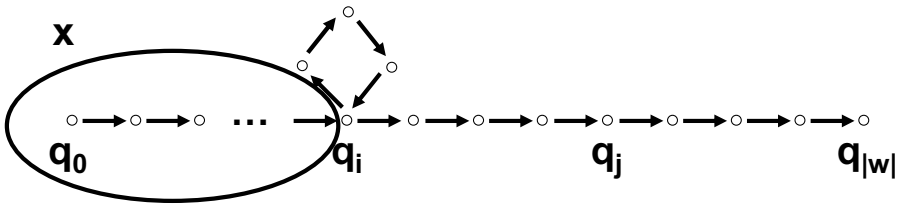
Let M be a DFA that recognizes L

Let P be the number of states in M

Assume w ∈ L is such that |w| ≥ P

We show w = xyz

1. **|y| > 0**
2. **|xy| ≤ P**
3. **$xy^iz \in L$ for any i ≥ 0**



There must be j > i such that $q_i = q_j$

USING THE PUMPING LEMMA

Use the pumping lemma to prove that
$B = \{0^n1^n \mid n \geq 0\}$ is not regular

Hint: Assume B is regular, and try pumping $s = 0^P1^P$

If B is regular, s can be split into s = xyz,
where for any i ≥ 0, xy^iz is also in B

If y is all 0s: xyyz has more 0s than 1s

If y is all 1s: xyyz has more 1s than 0s

If y has both 1s and 0s:

xyyz will have some 1s before some 0s

For next time

Read Chapter 1.2 of the book.

Also, get started on the homework ASAP!!