Finite Automata

Finite Automaton (FA)

- Informally, a state diagram that comprehensively captures all possible states and transitions that a machine can take while responding to a stream or sequence of input symbols

  E.g. Recognizer for “Regular Languages”

δ	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
$*q_2$	q_2	q_2

- Deterministic Finite Automata (DFA)
  - The machine can exist in only one state at any given time
- Non-deterministic Finite Automata (NFA)
  - The machine can exist in multiple states at the same time

Deterministic Finite Automata - Definition

- A Deterministic Finite Automaton (DFA) consists of:
  - Q ==> a finite set of states
  - $\sum$ ==> a finite set of input symbols (alphabet)
  - q_0 ==> a start state
  - F ==> set of accepting states
  - δ ==> a transition function, which is a mapping between Q x $\sum$ ==> Q
- A DFA is defined by the 5-tuple:
  - $\{Q, \sum, q_0, F, \delta\}$

What does a DFA do on reading an input string?

- <u>Input:</u> a word w in $\sum^*$
- <u>Question:</u> Is w acceptable by the DFA?
- <u>Steps:</u>
  - Start at the "start state" q_0
  - For every input symbol in the sequence w do
    - Compute the next state from the current state, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed, the current state is one of the accepting states (F) then *accept w;*
  - Otherwise, *reject w.*

Regular Languages

Let L(A) be a language *recognized* by a DFA A.

Then L(A) is called a "*Regular Language*".

The Chomsky Hierachy

- A containment hierarchy of classes of formal languages

Example #1

- Build a DFA for the following language:
  - L = {w | w is a binary string that contains 01 as a substring}
- Steps for building a DFA to recognize L:
  - $\sum = \{0,1\}$
  - Decide on the states: Q
  - Designate start state and final state(s)
  - δ: Decide on the transitions:
- "Final" states == same as "accepting states"
- Other states == same as "non-accepting states"

DFA for strings containing 01

- What makes this DFA deterministic?

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

symbols

δ	0	1
q_0	q_1	q_0
q_1	q_1	q_2
$*q_2$	q_2	q_2

states

- What if the language allows empty strings?

Question-1

- Build a DFA for the following language:

  L = { w | w is a binary string that has even number of 1s and even number of 0s}

Extension of transitions (δ) to Paths ($\hat{\delta}$)

- $\hat{\delta}$ *(q,w)* = *destination state* from state *q* on input <u>string</u> *w*

- $\hat{\delta}$ *(q,wa)* = δ ($\hat{\delta}$*(q,w), a)*

Consider the finite automata/state machine whose transition function is given by the following transition table. Here, $Q=\{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0,1\}$, $F=\{q_0\}$. Give the entire sequence of 110101

State	Input	
	0	1
→ q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

$$
\begin{aligned}
\delta(q_0, \overset{\downarrow}{1}10101) &= \delta(q_1, \overset{\downarrow}{1}0101) \\
&= \delta(q_0, \overset{\downarrow}{0}101) \\
&= \delta(q_2, \overset{\downarrow}{1}01) \\
&= \delta(q_3, \overset{\downarrow}{0}1) \\
&= \delta(q_1, \overset{\downarrow}{1}) \\
&= q_0
\end{aligned}
$$

Hence,

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_0$$

Language of a DFA

A DFA A accepts string *w* if there is a path from q_0 to an accepting (or final) state that is labeled by *w*

- *i.e., L(A) = all strings that lead to an accepting state from* q_0

Non-deterministic Finite Automata (NFA)

- A Non-deterministic Finite Automaton (NFA)
  - is of course "non-deterministic"
    - Implying that the machine can exist in more than one state at the same time
    - Transitions could be non-deterministic

• Each transition function therefore maps to a set of states

Non-deterministic Finite Automata (NFA)

- A Non-deterministic Finite Automaton (NFA) consists of:
  - Q ==> a finite set of states
  - $\sum$ ==> a finite set of input symbols (alphabet)
  - q_0 ==> a start state
  - F ==> set of accepting states
  - δ ==> a transition function, which is a mapping between Q x $\sum$ ==> subset of Q
- An NFA is also defined by the 5-tuple:
  - $\{Q, \sum, q_0, F, \delta\}$

How to use an NFA?

- <u>Input:</u> a word w in $\sum^*$
- <u>Question:</u> Is w acceptable by the NFA?
- <u>Steps:</u>
  - Start at the "start state" q_0
  - For every input symbol in the sequence w do
    - Determine all possible next states from all current states, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed <u>and</u> if at least one of the current states is a final state then *accept w;*
  - Otherwise, *reject w.*

NFA for strings containing 01

Why is this non-deterministic?

What will happen if at state q_1 an input of 0 is received?

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

DFA

	0	1
q_0	q_0	q_0

symbols

states

δ	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Φ	$\{q_2\}$
$*q_2$	$\{q_2\}$	$\{q_2\}$

Question-2

- Build an NFA for the following language:

  L = { w | w ends in 01}

- Other examples
  - Keyword recognizer (e.g., if, then, else, while, for, include, etc.)
  - Strings where the first symbol is present somewhere later on at least once

Advantages & Caveats for NFA

- Great for modeling regular expressions words
  - String processing - e.g., grep, lexical analyzer

- Could a non-deterministic state machine be implemented in practice?
  - Yes but in a limited way
  - Probabilistic models could be viewed as extensions of non-deterministic state machines
    (e.g., toss of a coin, a roll of dice)
    - They are not the same though
  - A parallel computer could exist in multiple “states” at the same time

NFA to a DFA

But, DFAs and NFAs are equivalent in their power to capture langauges !!

Differences: DFA vs. NFA

- **DFA**

1. All transitions are deterministic
   - Each transition leads to exactly one state
2. Accepts input if the last state visited is in F
3. Sometimes harder to construct because of the number of states
4. Practical implementation is feasible

- **NFA**

1. Some transitions could be non-deterministic
   - A transition could lead to a subset of states
2. Accepts input if *one of* the last states is in F
3. Generally easier than a DFA to construct
4. Practical implementations limited but emerging (e.g., Micron automata processor)

① Construct a DFA accepting all strings over {a,b} ending with ab

② Construct a NFA accepting all strings over {a, b} ending with ab.

NFA

ab
aab
bab
bbab
aaab
abab

aab

aaaab

aab

	a	b
1	1, 2	1
2	2	3
3	ϕ	ϕ

Equivalence of DFA & NFA

- Theorem:
  - A language L is accepted by a DFA *if and only if* it is accepted by an NFA.

Should be true for any L

- Proof:

1. If part:

   NFA → DFA

   - Prove by showing every NFA can be converted to an equivalent DFA (in the next few slides…)

2. Only-if part is trivial:
   - Every DFA is a special case of an NFA where each state has exactly one transition for every input symbol. Therefore, if L is accepted by a DFA, it is accepted by a corresponding NFA. □

Proof for the if-part

- If-part: A language L is accepted by a DFA if it is accepted by an NFA
- rephrasing…

NFA → DFA

- Given any NFA N, we can construct a DFA D such that L(N)=L(D)

- How to convert an NFA into a DFA?
  - Observation: In an NFA, each transition maps to a *subset* of states
  - Idea: Represent:

    each "subset of NFA_states" ➔ a single "DFA_state"

Subset construction

NFA to DFA by subset construction

- Let $N = \{Q_N, \Sigma, \delta_N, q_0, F_N\}$
- <u>Goal:</u> Build $D=\{Q_D, \Sigma, \delta_D, \{q_0\}, F_D\}$ s.t. $L(D)=L(N)$
- <u>Construction:</u>
  1. Q_D= all subsets of Q_N (i.e., power set)
  2. F_D=set of subsets S of Q_N s.t. $S \cap F_N \neq \Phi$
  3. δ_D: for each subset S of Q_N and for each input symbol a in Σ:
     - $\delta_D(S,a) = \bigcup_{p \text{ in } s} \delta_N(p,a)$

Idea: To avoid enumerating all of power set, do "lazy creation of states"

NFA to DFA construction: Example

- *L = {w | w ends in 01}*

NFA:

δ_N	0	1
q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
$*q_2$	Ø	Ø

NFA → DFA

states = 3
set of inputs = 0, 1

DFA:

δ_D
~~Ø~~
$[q_0]$
~~$[q_1]$~~
~~$*[q_2]$~~
$[q_0,q_1]$
$*[q_0,q_2]$
~~$*[q_1,q_2]$~~
~~$*[q_0,q_1,q_2]$~~

δ_D	0	1
$[q_0]$	$[q_0,q_1]$	$[q_0]$
$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_2]$
$*[q_0,q_2]$	$[q_0,q_1]$	$[q_0]$

0. Enumerate all possible subsets
1. Determine transitions
2. Retain only those states reachable from $\{q_0\}$

NFA to DFA: Repeating the example using *LAZY CREATION*

- *L = {w | w ends in 01}*

NFA:

δ_N	0	1
q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$*q_2$	$\emptyset$	$\emptyset$

DFA:

δ_D	0	1
$[q_0]$	$[q_0,q_1]$	$[q_0]$

Final

Main Idea:

Introduce states as you go (on a need basis)

Question-3

Construct a deterministic automaton equivalent to

$$M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

where δ is defined by its state table (see Table 3.2).

TABLE 3.2 State Table for Example 3.6

State/Σ	0	1
→ q_0	q_0	q_1
q_1	q_1	q_0, q_1

NFA ≈ DFA

	0	1
→ q_0	q_0	q_1
q_1	q_1	q_0, q_1

	0	1
$q_0 q_1$	$(q_0 q_1)$	$(q_0 q_1)$

State/Σ	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

DFA

Question-4

Find a deterministic acceptor equivalent to

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$$

where δ is as given by Table 3.4.

	a	b
$\rightarrow q_0$	$(q_0 q_1)$	q_2
$q_0 q_1$	$(q_0 q_1)$	$(q_1 q_2)$
q_2	ϕ	$(q_0 q_1)$
$q_1 q_2$	q_0	$(q_0 q_1)$

TABLE 3.4 State Table for Example 3.7

State/Σ	a	b
$\rightarrow q_0$	q_0, q_1	q_2
q_1	q_0	q_1
q_2	ϕ	q_0, q_1

NFA

↳ DFA

State/Σ	a	b
$[q_0]$	$[q_0, q_1]$	$[q_2]$
$[q_2]$	$\emptyset$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_0]$	$[q_0, q_1]$

DFA

Applications

- Text indexing
  - inverted indexing
  - For each unique word in the database, store all locations that contain it using an NFA or a DFA
- Find pattern P in text T
  - Example: Google querying
- Extensions of this idea:
  - PATRICIA tree, suffix tree

A few subtle properties of DFAs and NFAs

- The machine never really terminates.
  - It is always waiting for the next input symbol or making transitions.
- The machine decides when to consume the next symbol from the input and when to ignore it.
  - (but the machine can never skip a symbol)
- => A transition can happen even *without* really consuming an input symbol (think of consuming ε as a free token) – if this happens, then it becomes an ε-NFA (see next few slides).
- A single transition *cannot* consume more than one (non-ε) symbol.

FA with ε-Transitions

- We can allow <u>explicit</u> ε-transitions in finite automata
  - i.e., a transition from one state to another state without consuming any additional input symbol
  - Explicit ε-transitions between different states introduce non-determinism.
  - Makes it easier sometimes to construct NFAs

<u>Definition:</u> ε-NFAs are those NFAs with at least one explicit ε-transition defined.

	0	1	ε

- ε -NFAs have one more column in their transition table

Example of an ε-NFA

L = {w | w is empty, or if non-empty will end in 01}

δ_E	0	1	ε
*q'_0	Ø	Ø	$\{q'_0, q_0\}$
q_0	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
*q_2	Ø	Ø	$\{q_2\}$

ECLOSE(q'_0)

ECLOSE(q_0)

ECLOSE(q_1)

ECLOSE(q_2)

- ε-closure of a state q, ***ECLOSE(q)***, is the set of all states (including itself) that can be *reached* from q by repeatedly making an arbitrary number of ε-transitions.

$\epsilon\text{Closure}(q'_0) = \{q'_0, q_0\}$

To simulate any transition:

Step 1) Go to all immediate destination states.

Step 2) From there go to all their ε-closure states as well.

Example of an ε-NFA

L = {w | w is empty, or if non-empty will end in 01}

δ_E	0	1	ε
→ $*q'_0$	Ø	Ø	$\{q'_0, q_0\}$
q_0	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
$*q_2$	Ø	Ø	$\{q_2\}$

ECLOSE(q'_0)

ECLOSE(q_0)

Simulate for w=101:

To simulate any transition:

Step 1) Go to all immediate destination states.

Step 2) From there go to all their ε-closure states as well.

Example of another ε-NFA

δ_E	0	1	ε
→ $*q'_0$	Ø	Ø	$\{q'_0,q_0,q_3\}$
q_0	$\{q_0,q_1\}$	$\{q_0\}$	$\{q_0,q_3\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
$*q_2$	Ø	Ø	$\{q_2\}$
q_3	Ø	$\{q_2\}$	$\{q_3\}$

Simulate for w=101:

?

$q'_0 \xrightarrow{\epsilon} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2$

$w = 1$

$w = 11$

q'_0

$w = 1$

$q'_0 \rightarrow q_0 \rightarrow q_3 \rightarrow q_2$

lazy creation
subset const.

Equivalency of DFA, NFA, ε-NFA

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

- Implication:
  - DFA ≡ NFA ≡ ε-NFA
  - (all accept Regular Languages)

DFA
NFA
ε NFA
Regular language.

Eliminating ε-transitions

Let $E = \{Q_E, \Sigma, \delta_E, q_0, F_E\}$ be an ε-NFA

<u>Goal:</u> To build DFA $D=\{Q_D, \Sigma, \delta_D, \{q_D\}, F_D\}$ s.t. $L(D)=L(E)$

<u>Construction:</u>

1. Q_D = all reachable subsets of Q_E factoring in ε-closures
2. q_D = ECLOSE(q_0)
3. F_D=subsets S in Q_D s.t. $S \cap F_E \neq \Phi$
4. δ_D: for each subset S of Q_E and for each input symbol $a \in \Sigma$:
   - Let $R = \bigcup_{p \text{ in } s} \delta_E(p,a)$ // go to destination states
   - $\delta_D(S,a) = \bigcup_{r \text{ in } R} \text{ECLOSE}(r)$ // from there, take a union of all their ε-closures

Example: ε-NFA ➔ DFA

L = {w | w is empty, or if non-empty will end in 01}

ε NFA

$\Sigma = 0, 1, \epsilon$

ε closure

$\epsilon cl(q'_0) = (q'_0, q_0)$

δ_E	0	1	ε
→ *q'_0	Ø	Ø	$\{q'_0, q_0\}$
q_0	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
*q_2	Ø	Ø	$\{q_2\}$

δ_D	0	1
→ *$\{q'_0, q_0\}$	$\{q_0 q_1\}$	q_0
✓ $\{q_0 q_1\}$	$\{q_0 q_1\}$	$\{q_0, q_2\}$
✓ $\{q_0\}$	$\{q_0 q_1\}$	$\{q_0\}$
* $\{q_0 q_2\}$	$q_0 q_1$	q_0

Example: ε-NFA ➔ DFA

L = {w | w is empty, or if non-empty will end in 01}

$\varepsilon u(q'_0) = q'_0, q_0$

ECLOSE

union

δ_E	0	1	ε
→ *q'_0	Ø	Ø	$\{q'_0,q_0\}$
q_0	$\{q_0,q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
*q_2	Ø	Ø	$\{q_2\}$

δ_D	0	1
→ *$\{q'_0,q_0\}$	$\{q_0,q_1\}$	$\{q_0\}$
$\{q_0,q_1\}$	$\{q_0,q_1\}$	$\{q_0,q_2\}$
$\{q_0\}$	$\{q_0,q_1\}$	$\{q_0\}$
*$\{q_0,q_2\}$	$\{q_0,q_1\}$	$\{q_0\}$

Summary

- DFA
  - Definition
  - Transition diagrams & tables
- Regular language
- NFA
  - Definition
  - Transition diagrams & tables
- DFA vs. NFA
- NFA to DFA conversion using subset construction
- Equivalency of DFA & NFA
- Removal of redundant states and including dead states
- ε-transitions in NFA
- Text searching applications

Internet sources

1.https://www.eecs.wsu.edu/~ananth/CptS317/Lectures/index.htm

2. https://nptel.ac.in/courses/106/104/106104028/

3. https://nptel.ac.in/courses/106/104/106104148/

abba
aabba
aaabba
babba
ababba
bbabba
bbbabba

baabba

ababa

ababba

- Draw a DFA for the language accepting strings ending with ‘abba’ over input alphabets ∑ = {a, b}

DFA

aba
abb
abbb
abab
abaaa

- Draw a DFA for the language accepting strings starting with ‘ab’ over input alphabets ∑ = {a, b}

DFA

NFA / NDFA

- Draw a non-deterministic finite automate which accepts 00 or 11 at the end of a string containing 0, 1 in it, e.g., 01010100 but not 000111010.
- 0101011 or
- 011 or
- 100
- 0011

NFA

0101011 or
011 or
100
0011

- Draw a non-deterministic finite automata which accept a string containing “ing” at the end of a string in a string of {a-z}, e.g., “anything” but not “anywhere”.

Something

NFA

- Convert the following Non-Deterministic Finite Automata (NFA) to Deterministic Finite Automata (DFA)-

	a	b
q_0	q_0	q_0 q_1
q_1	ϕ	b
q_2	ϕ	ϕ

Deterministic Finite Automata (DFA)

- Convert the following Non-Deterministic Finite Automata (NFA) to Deterministic Finite Automata (DFA)-

	0	1
→ q_0	q_0	$q_1 q_2$
q_1	$q_1 q_2$	q_2
q_2	$q_0 q_1$	q_1

	0	1
→ q_0	q_0	$q_1 q_2$
$q_1 q_2$	$q_0 q_1 q_2$	$q_1 q_2$
$q_0 q_1 q_2$	$q_0 q_1 q_2$	$q_1 q_2$

← DFA

Deterministic Finite Automata (DFA)

Find the string recognized by this FA

a bbb a

× ① strings ending ba

× ② strings ending aa ×

✓ ③ a followed any no. of b's then a ✓ ending with

✓ ④ string starting with an 'a' & ending with an 'a' having zero/more bs in between a's

Describe in English the set accepted by this FA

string containing 'n' number of 0's or 1's

✓ starting with 0's string not containing 10

✗ string containing continuous 1's and continuous 0's

NFA

State/Σ	0	1
→ q_0	q_0, q_3	q_0, q_4
q_3	q_f	
q_4		q_f
q_f	q_f	q_f

DFA → 5 states

DFA

Q	Q_0	Q_1
$\rightarrow [q_0]$	$[q_0, q_3]$	$[q_0, q_4]$
$[q_0, q_3]$	$[q_0, q_3, q_f]$	$[q_0, q_4]$
$[q_0, q_4]$	$[q_0, q_3]$	$[q_0, q_4, q_f]$
$[q_0, q_3, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$
$[q_0, q_4, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$

Minimization of DFA

Example 1

	0	1
→ A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

Partitioning Method for minimization of DFA

P_0 π_0 ABCDE

P_0 (ABCD) (E)

P_1 (ABC) (D) (E)

P_2 (AC) (B) (D) (E)

↓ ↓ ↓ ↓

A B D E

	0	1
A	B	A
B	B	D
D	B	E
E	B	A

minimized DFA

minimization of DFA (Example 2)

DFA

	0	1
→ A	B	C
B	D	E
C	E	D
D	F	F
E	F	F
F	F	F

→ minimized DFA

P_0 (ADE) (BCF)

P_1 (ADE) (BC) (F)

P_2 $\underbrace{(A)}_{A}$ $\underbrace{(DE)}_{B}$ $\underbrace{(BC)}_{C}$ $\underbrace{(F)}_{F}$

	0	1
→ A	B	C
B	B	B
C	B	B
F	F	F

minimized DFA

~~Exploo~~ Example 3 of minimization of DFA

	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

P_0 $(q_0\, q_1\, q_3\, q_4\, q_5\, q_6\, q_7)\ (q_2)$

P_f $(q_0 q_4)(q_1 q_7)(q_2)(q_3 q_5)(q_6)$

5 state DFA

Equivalence & Minimization of DFAs

Applications of interest

- Comparing two DFAs:
  - $L(DFA_1) == L(DFA_2)$?

- How to minimize a DFA?
  1. Remove unreachable states
  2. Identify & condense equivalent states into one

When to call two states in a DFA "equivalent"?

Two states p and q are said to be *equivalent* iff:

i) Any string w accepted by starting at p is also accepted by starting at q;

AND

i) Any string w rejected by starting at p is also rejected by starting at q.

➔ p≡q

Computing equivalent states in a DFA

Table Filling Algorithm

A	=							
B	=	=						
C	x	x	=					
D	x	x	x	=				
E	x	x	x	x	=			
F	x	x	x	x	x	=		
G	x	x	x	=	x	x	=	
H	x	x	=	x	x	x	x	=
	A	B	C	D	E	F	G	H

Pass #0

1. Mark accepting states ≠ non-accepting states

Pass #1

1. Compare every pair of states
2. Distinguish by one symbol transition
3. Mark = or ≠ or blank(tbd)

Pass #2

1. Compare every pair of states
2. Distinguish by up to two symbol transitions (until different or same or tbd)

….

(keep repeating until table complete)

Table Filling Algorithm - step by step

A	=							
B		=						
C			=					
D				=				
E					=			
F						=		
G							=	
H								=
	A	B	C	D	E	F	G	H

Table Filling Algorithm - step by step

	A	B	C	D	E	F	G	H
A	~~X~~							
B		~~X~~						
C			~~X~~					
D				~~X~~				
E	X	X	X	X	~~X~~			
F					X	~~X~~		
G					X		~~X~~	
H					X			~~X~~

1. Mark X between accepting vs. non-accepting state

Table Filling Algorithm - step by step

A	=							
B		=						
C	X		=					
D	X			=				
E	X	X	X	X	=			
F					X	=		
G	X				X		=	
H	X				X			=
	A	B	C	D	E	F	G	H

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

Table Filling Algorithm - step by step

A	=							
B		=						
C	X	X	=					
D	X	X		=				
E	X	X	X	X	=			
F					X	=		
G	X	X			X		=	
H	X	X			X			=
	A	B	C	D	E	F	G	H

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

Table Filling Algorithm - step by step

A	=							
B		=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F			X		X	=		
G	X	X	X		X		=	
H	X	X	=		X			=
	A	B	C	D	E	F	G	H

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

Table Filling Algorithm - step by step

A	=							
B		=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F			X	X	X	=		
G	X	X	X	=	X		=	
H	X	X	=	X	X			=
	A	B	C	D	E	F	G	H

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

Table Filling Algorithm - step by step

A	=							
B		=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F			X	X	X	=		
G	X	X	X	=	X	X	=	
H	X	X	=	X	X	X		=
	A	B	C	D	E	F	G	H

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

Table Filling Algorithm - step by step

A	=							
B		=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F			X	X	X	=		
G	X	X	X	=	X	X	=	
H	X	X	=	X	X	X	X	=
	A	B	C	D	E	F	G	H

1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

Table Filling Algorithm - step by step

A	=							
B	=	=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F	X	X	X	X	X	=		
G	X	X	X	=	X	X	=	
H	X	X	=	X	X	X	X	=
	A	B	C	D	E	F	G	H

1. Mark X between accepting vs. non-accepting state
2. Pass 1:
   Look 1- hop away for distinguishing states or strings
3. Pass 2:
   Look 1-hop away again for distinguishing states or strings

continue….

Table Filling Algorithm - step by step

	A	B	C	D	E	F	G	H
A	=							
B	=	=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F	X	X	X	X	X	=		
G	X	X	X	=	X	X	=	
H	X	X	=	X	X	X	X	=

1. Mark X between accepting vs. non-accepting state
2. Pass 1:
   Look 1- hop away for distinguishing states or strings
3. Pass 2:
   Look 1-hop away again for distinguishing states or strings

continue….

Equivalences:
- A=B
- C=H
- D=G

Table Filling Algorithm - step by step

Retrain only one copy for each equivalence set of stat

Equivalences:
- A=B
- C=H
- D=G

Table Filling Algorithm – special case – Assignment

A	=							
B		=						
C			=					
D				=				
E				?	=			
F						=		
G							=	
H								=
	A	B	C	D	E	F	G	H

Q) What happens if the input DFA has more than one final state?

Can all final states initially be treated as equivalent to one another?

How to minimize a DFA?

- <u>Goal:</u> Minimize the number of states in a DFA
- <u>Algorithm:</u>

Depth-first traversal from the start sta

1. Eliminate states unreachable from the start state

Table filling algorithm

2. Identify and remove equivalent states
3. Output the resultant DFA

Are Two DFAs Equivalent?

Is $q_0 \equiv q_0$'?
: if yes, then $DFA_1 \equiv DFA_2$
: else, not equiv.

1. Make a new dummy DFA by just putting together both DFAs
2. Run table-filling algorithm on the unified DFA
3. *IF* the start states of both DFAs are found to be equivalent,

   THEN: $DFA_1 \equiv DFA_2$

   ELSE: different

Myhill-Nerode Theorem
(table filling method)

Steps:

1) Draw a table for all pairs of states (P,Q)
2) Mark all pairs where $P \in F$ and $Q \notin F$
3) If there are any Unmarked pairs (P,Q) such that $[\delta(P,x), \delta(Q,x)]$ is marked, then mark [P,Q] where 'x' is an input symbol
   REPEAT THIS UNTIL NO MORE MARKINGS CAN BE MADE
4) Combine all the Unmarked Pairs and make them a single state in the minimized DFA

a
a
q1
q3
b
a
q0
a
a
b
b
q2
q4
b
b

a
a
b
q1
q3
b
a
q0q2
a
b
b
q4
a
a
B
b
C
a
a
b
b
A
b
D

Minimize the given DFA

NFA
↳ DFA
↳ minimized DFA

Minimal DFA