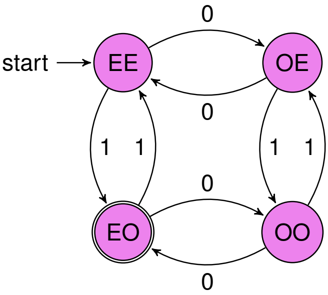
Contents

Section outline

A simple recogniser



EE 0: even, 1: even

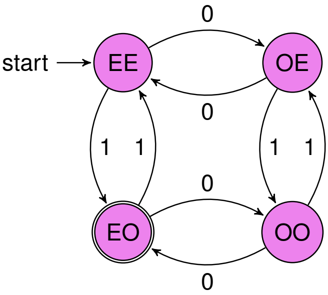
OE 0: odd, 1: even

EO 0: even, 1: odd

OO 0: odd, 1: odd

- $Q = \{\mathrm{EE}, \mathrm{EO}, \mathrm{OE}, \mathrm{OO}\}$

A simple recogniser



- $Q = \{\text{EE}, \text{EO}, \text{OE}, \text{OO}\}$
- $\Sigma = \{0, 1\}$

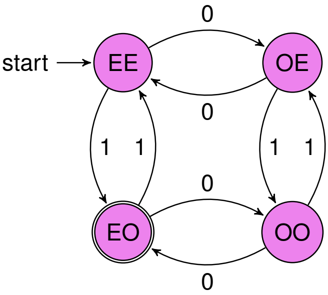
EE 0: even, 1: even

OE 0: odd, 1: even

EO 0: even, 1: odd

OO 0: odd, 1: odd

A simple recogniser



EE 0: even, 1: even

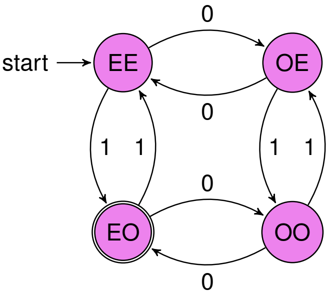
OE 0: odd, 1: even

EO 0: even, 1: odd

OO 0: odd, 1: odd

- $Q = \{\mathrm{EE}, \mathrm{EO}, \mathrm{OE}, \mathrm{OO}\}$
- $\Sigma = \{0, 1\}$
- $\delta = \left\{ \begin{array}{l} \langle \mathrm{EE}, 0\rangle \mapsto \mathrm{OE}, \\ \langle \mathrm{EE}, 1\rangle \mapsto \mathrm{EO}, \\ \langle \mathrm{OE}, 0\rangle \mapsto \mathrm{EE}, \\ \langle \mathrm{OE}, 1\rangle \mapsto \mathrm{OO}, \\ \langle \mathrm{EO}, 0\rangle \mapsto \mathrm{OO}, \\ \langle \mathrm{EO}, 1\rangle \mapsto \mathrm{EE}, \\ \langle \mathrm{OO}, 0\rangle \mapsto \mathrm{EO}, \\ \langle \mathrm{OO}, 1\rangle \mapsto \mathrm{OE} \end{array} \right\}$

A simple recogniser



EE 0: even, 1: even

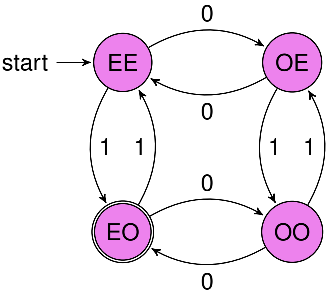
OE 0: odd, 1: even

EO 0: even, 1: odd

OO 0: odd, 1: odd

- $Q = \{\mathrm{EE}, \mathrm{EO}, \mathrm{OE}, \mathrm{OO}\}$
- $\Sigma = \{0, 1\}$
- $\delta = \left\{ \begin{array}{l} \langle \mathrm{EE}, 0 \rangle \mapsto \mathrm{OE}, \\ \langle \mathrm{EE}, 1 \rangle \mapsto \mathrm{EO}, \\ \langle \mathrm{OE}, 0 \rangle \mapsto \mathrm{EE}, \\ \langle \mathrm{OE}, 1 \rangle \mapsto \mathrm{OO}, \\ \langle \mathrm{EO}, 0 \rangle \mapsto \mathrm{OO}, \\ \langle \mathrm{EO}, 1 \rangle \mapsto \mathrm{EE}, \\ \langle \mathrm{OO}, 0 \rangle \mapsto \mathrm{EO}, \\ \langle \mathrm{OO}, 1 \rangle \mapsto \mathrm{OE} \end{array} \right\}$
- $q_I = \mathrm{EE}$

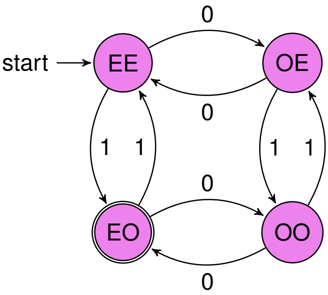
A simple recogniser



EE	0: even, 1: even
OE	0: odd, 1: even
EO	0: even, 1: odd
OO	0: odd, 1: odd

- $Q = \{\text{EE}, \text{EO}, \text{OE}, \text{OO}\}$
- $\Sigma = \{0, 1\}$
- $\delta = \left\{ \begin{array}{l} \langle \text{EE}, 0\rangle \mapsto \text{OE}, \\ \langle \text{EE}, 1\rangle \mapsto \text{EO}, \\ \langle \text{OE}, 0\rangle \mapsto \text{EE}, \\ \langle \text{OE}, 1\rangle \mapsto \text{OO}, \\ \langle \text{EO}, 0\rangle \mapsto \text{OO}, \\ \langle \text{EO}, 1\rangle \mapsto \text{EE}, \\ \langle \text{OO}, 0\rangle \mapsto \text{EO}, \\ \langle \text{OO}, 1\rangle \mapsto \text{OE} \end{array} \right\}$
- $q_I = \text{EE}$
- $F = \{\text{EO}\}$

Deterministic FA



EE 0: even, 1: even

OE 0: odd, 1: even

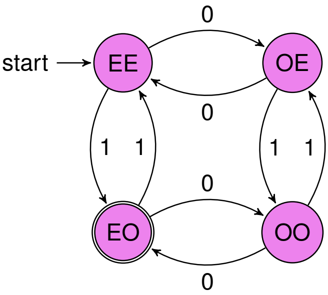
EO 0: even, 1: odd

OO 0: odd, 1: odd

DFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, where

- Q Finite set of states
- Σ Alphabet, finite set of symbols
- δ Transition function, $\delta : Q \times \Sigma \to Q$
- q_I Starting or initial state
- F Set of accepting or final states, $F \subseteq Q$

Deterministic FA



EE 0: even, 1: even

OE 0: odd, 1: even

EO 0: even, 1: odd

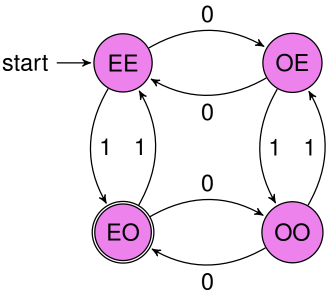
OO 0: odd, 1: odd

DFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, where

- Q Finite set of states
- Σ Alphabet, finite set of symbols
- δ Transition function, $\delta : Q \times \Sigma \to Q$
- q_I Starting or initial state
- F Set of accepting or final states, $F \subseteq Q$

- No transitions after input finishes

Deterministic FA



EE 0: even, 1: even

OE 0: odd, 1: even

EO 0: even, 1: odd

OO 0: odd, 1: odd

DFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, where

- Q Finite set of states
- Σ Alphabet, finite set of symbols
- δ Transition function, $\delta : Q \times \Sigma \to Q$
- q_I Starting or initial state
- F Set of accepting or final states, $F \subseteq Q$

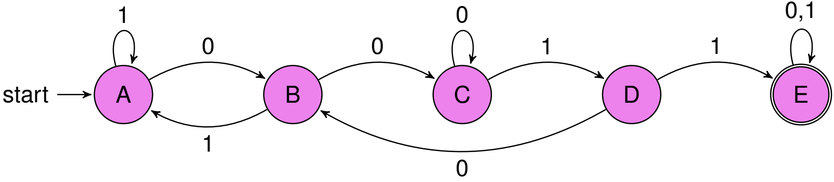
- No transitions after input finishes
- Srings taking M to an accepted state is the language L_M accepted by M

Example of designing a FA

- $\Sigma = \{0, 1\}$, want to recognize any string that does *contain* 0011 in it

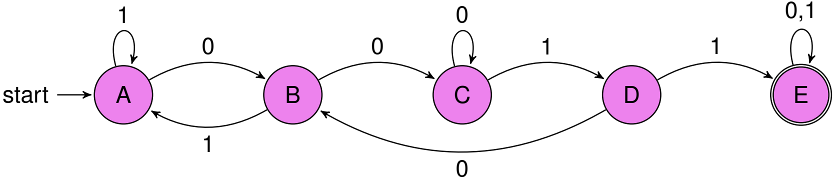
Example of designing a FA

- $\Sigma = \{0, 1\}$, want to recognize any string that does *contain* 0011 in it
- Consider m/c M_1 that accepts any string containing 0011 (L_{M_1})

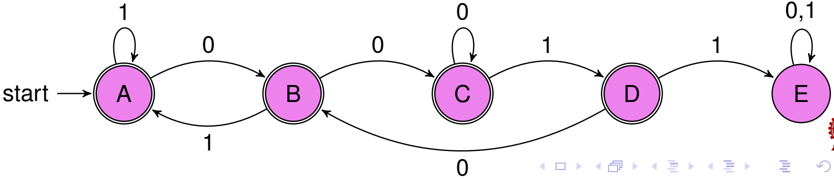


Example of designing a FA

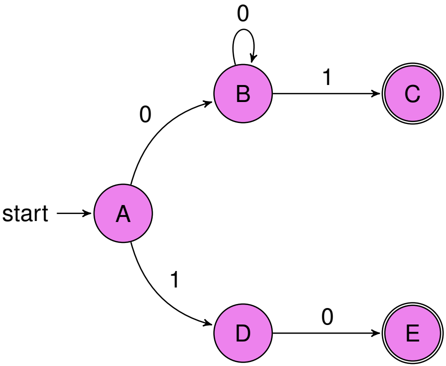
- $\Sigma = \{0, 1\}$, want to recognize any string that does *contain* 0011 in it
- Consider m/c M_1 that accepts any string containing 0011 (L_{M_1})



- Next consider M_2 obtained by interchanging the accepting and non-accepting states of M_1; $L_{M_2} = \overline{L_{M_1}}$?

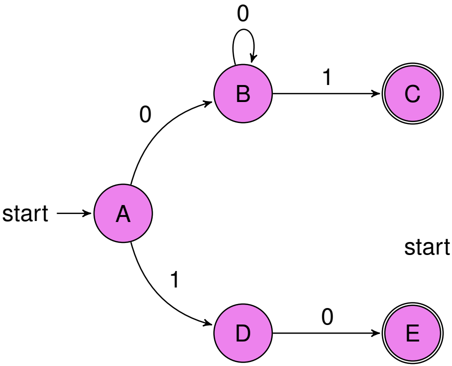


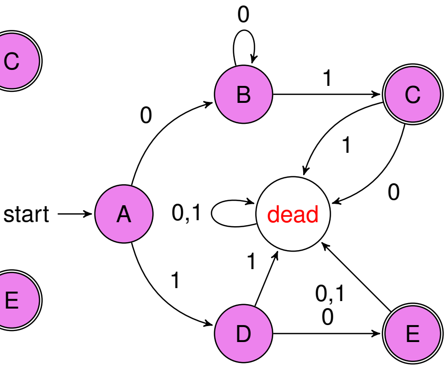
Dead states



- Recognizes $10, 01, 001, \ldots, 0^{+}1$
- What happens on 111 or 1010?

Dead states





- Recognizes $10, 01, 001, \ldots, 0^{+}1$
- What happens on 111 or 1010?

- Introduce a dead state
- Let any missing transition lead to the dead state
- Often not shown, left implicit

Regular languages

- Let $M = \langle Q, \Sigma, \delta, q_I, F \rangle$

Regular languages

- Let $M = \langle Q, \Sigma, \delta, q_I, F \rangle$
- Let $w = w_1 w_2 \ldots w_n$ be a string, $w_i \in \Sigma$

Regular languages

- Let $M = \langle Q, \Sigma, \delta, q_I, F\rangle$
- Let $w = w_1 w_2 \dots w_n$ be a string, $w_i \in \Sigma$
- M accepts w if there is a sequence of states $s_0, s_1, \dots, s_n$, $s_i \in Q$ such that $s_0 = q_I$, $\delta(s_i, w_{i+1}) = s_{i+1}$, for $1 \leq i < n$ and $s_n \in F$

Regular languages

- Let $M = \langle Q, \Sigma, \delta, q_I, F \rangle$
- Let $w = w_1 w_2 \ldots w_n$ be a string, $w_i \in \Sigma$
- M accepts w if there is a sequence of states $s_0, s_1, \ldots, s_n$, $s_i \in Q$ such that $s_0 = q_I$, $\delta(s_i, w_{i+1}) = s_{i+1}$, for $1 \leq i < n$ and $s_n \in F$
- In other words w takes M starting from the intitial state to and accepting state

Regular languages

- Let $M = \langle Q, \Sigma, \delta, q_I, F\rangle$
- Let $w = w_1 w_2 \ldots w_n$ be a string, $w_i \in \Sigma$
- M accepts w if there is a sequence of states $s_0, s_1, \ldots, s_n$, $s_i \in Q$ such that $s_0 = q_I$, $\delta(s_i, w_{i+1}) = s_{i+1}$, for $1 \leq i < n$ and $s_n \in F$
- In other words w takes M starting from the intitial state to and accepting state
- M *recognises* language A if $A = \{w | M \text{ accepts } w\}$

Regular languages

- Let $M = \langle Q, \Sigma, \delta, q_I, F \rangle$
- Let $w = w_1 w_2 \ldots w_n$ be a string, $w_i \in \Sigma$
- M accepts w if there is a sequence of states $s_0, s_1, \ldots, s_n$, $s_i \in Q$ such that $s_0 = q_I$, $\delta(s_i, w_{i+1}) = s_{i+1}$, for $1 \leq i < n$ and $s_n \in F$
- In other words w takes M starting from the intitial state to and accepting state
- M *recognises* language A if $A = \{w | M \text{ accepts } w\}$
- A language is a *regular language* if and only if it there is a finite automaton that recognises it

Regular languages

- Let $M = \langle Q, \Sigma, \delta, q_I, F \rangle$
- Let $w = w_1 w_2 \ldots w_n$ be a string, $w_i \in \Sigma$
- M accepts w if there is a sequence of states $s_0, s_1, \ldots, s_n$, $s_i \in Q$ such that $s_0 = q_I$, $\delta(s_i, w_{i+1}) = s_{i+1}$, for $1 \leq i < n$ and $s_n \in F$
- In other words w takes M starting from the intitial state to and accepting state
- M *recognises* language A if $A = \{w | M \text{ accepts } w\}$
- A language is a *regular language* if and only if it there is a finite automaton that recognises it
- Are there languages that are not regular?

Regular languages

- Let $M = \langle Q, \Sigma, \delta, q_I, F\rangle$
- Let $w = w_1 w_2 \ldots w_n$ be a string, $w_i \in \Sigma$
- *M* accepts *w* if there is a sequence of states $s_0, s_1, \ldots, s_n$, $s_i \in Q$ such that $s_0 = q_I$, $\delta(s_i, w_{i+1}) = s_{i+1}$, for $1 \leq i < n$ and $s_n \in F$
- In other words *w* takes *M* starting from the intitial state to and accepting state
- *M recognises* language *A* if $A = \{w|M \text{ accepts } w\}$
- A language is a *regular language* if and only if it there is a finite automaton that recognises it
- Are there languages that are not regular?
- Intuitively, if the language requires us to keep track of an arbitrary amount of the input to determine acceptance, a FA is likely to fail because it has only a finite number of states at its disposal

FA design practice

- FA for binary numbers that divisible by 3

Practice example of DFA

1. Draw DFA which accepts binary numbers divisible by 3.
2. Draw DFA for the given language. In all parts alphabet is $\{a,b\}$
   1. $\{w|w$ has even number of a's and each a is followed by at least one b$\}$
   2. $\{w|w$ is a string that does not contain exactly two a's $\}$
   3. $\{w|n_a \mod (3) > n_b \mod (3)$ where n_a, n_b are the numbers of a's and b's in string w respectively$\}$
   4. $\{w|w$ is a string that contains at least two a's and at most one b$\}$
   5. The empty set.
   6. All strings except empty string.

Union of Regular Languages

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$

Union of Regular Languages

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$
- Can we build a FA to recognise L?

Union of Regular Languages

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$
- Can we build a FA to recognise L?
- Running M_1 and, on failure, then M_2, is not an option because the input cannot be rewinded!

Union of Regular Languages

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$
- Can we build a FA to recognise L?
- Running M_1 and, on failure, then M_2, is not an option because the input cannot be rewinded!
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem

Union of Regular Languages

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$
- Can we build a FA to recognise L?
- Running M_1 and, on failure, then M_2, is not an option because the input cannot be rewinded!
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *any* m/c accepts it

Union of Regular Languages

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$
- Can we build a FA to recognise L?
- Running M_1 and, on failure, then M_2, is not an option because the input cannot be rewinded!
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *any* m/c accepts it
- Is this arrangement a FA, if so what is its formal representation?

Union of Regular Languages

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$
- Can we build a FA to recognise L?
- Running M_1 and, on failure, then M_2, is not an option because the input cannot be rewinded!
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *any* m/c accepts it
- Is this arrangement a FA, if so what is its formal representation?
- $Q = Q_1 \times Q_2$, $q_I = \langle q_{I_1}, q_{I_2} \rangle$

Union of Regular Languages

- Consider L_1 and L_2 over a common Σ, $L = L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$
- Can we build a FA to recognise L?
- Running M_1 and, on failure, then M_2, is not an option because the input cannot be rewinded!
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *any* m/c accepts it
- Is this arrangement a FA, if so what is its formal representation?
- $Q = Q_1 \times Q_2$, $q_I = \langle q_{I_1}, q_{I_2} \rangle$
- $F = \{\langle q_1, \_\rangle | q_1 \in F_1\} \cup \{\langle \_, q_2\rangle | q_2 \in F_2\}$

Union of Regular Languages

- Consider L_1 and L_2 over a common Σ, $L = L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$
- Can we build a FA to recognise L?
- Running M_1 and, on failure, then M_2, is not an option because the input cannot be rewinded!
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *any* m/c accepts it
- Is this arrangement a FA, if so what is its formal representation?
- $Q = Q_1 \times Q_2$, $q_I = \langle q_{I_1}, q_{I_2} \rangle$
- $F = \{\langle q_1, \_\rangle | q_1 \in F_1\} \cup \{\langle \_, q_2\rangle | q_2 \in F_2\}$
- $\delta = \{\langle\langle q_{1_a}, q_{2_c}\rangle, x\rangle \mapsto \langle q_{1_b}, q_{2_d}\rangle\} | \langle q_{1_a}, x\rangle \mapsto q_{1_b} \in \delta_1 \wedge \langle q_{1_c}, x\rangle \mapsto q_{1_d} \in \delta_2$

Intersection of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$

Intersection of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$
- Can we build a FA to recognise L?

Intersection of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cap L_2 = \{x | x \in L_1 \land x \in L_2\}$
- Can we build a FA to recognise L?
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem

Intersection of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$
- Can we build a FA to recognise L?
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *both* m/cs accepts it

Intersection of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$
- Can we build a FA to recognise L?
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *both* m/cs accepts it
- Is this arrangement a FA, if so what is its formal representation?

Intersection of RLs

- Consider L_1 and L_2 over a common Σ, $L = L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$
- Can we build a FA to recognise L?
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *both* m/cs accepts it
- Is this arrangement a FA, if so what is its formal representation?
- $Q = Q_1 \times Q_2$, $q_I = \langle q_{I_1}, q_{I_2} \rangle$

Intersection of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$
- Can we build a FA to recognise L?
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *both* m/cs accepts it
- Is this arrangement a FA, if so what is its formal representation?
- $Q = Q_1 \times Q_2$, $q_I = \langle q_{I_1}, q_{I_2} \rangle$
- $F = \{\langle q_1, q_2 \rangle | q_1 \in F_1, q_2 \in F_2\}$

Intersection of RLs

- Consider L_1 and L_2 over a common Σ, $L = L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$
- Can we build a FA to recognise L?
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *both* m/cs accepts it
- Is this arrangement a FA, if so what is its formal representation?
- $Q = Q_1 \times Q_2$, $q_I = \langle q_{I_1}, q_{I_2} \rangle$
- $F = \{\langle q_1, q_2 \rangle | q_1 \in F_1, q_2 \in F_2\}$
- $\delta = \{\langle\langle q_{1_a}, q_{2_c} \rangle, x\rangle \mapsto \langle q_{1_b}, q_{2_d} \rangle\} | \langle q_{1_a}, x\rangle \mapsto q_{1_b} \in \delta_1 \wedge \langle q_{1_c}, x\rangle \mapsto q_{1_d} \in \delta_2$

Intersection of RLs

- Consider L_1 and L_2 over a common Σ, $L = L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$
- Can we build a FA to recognise L?
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *both* m/cs accepts it
- Is this arrangement a FA, if so what is its formal representation?
- $Q = Q_1 \times Q_2$, $q_I = \langle q_{I_1}, q_{I_2} \rangle$
- $F = \{\langle q_1, q_2 \rangle | q_1 \in F_1, q_2 \in F_2\}$
- $\delta = \{\langle\langle q_{1_a}, q_{2_c}\rangle, x\rangle \mapsto \langle q_{1_b}, q_{2_d}\rangle\} | \langle q_{1_a}, x\rangle \mapsto q_{1_b} \in \delta_1 \wedge \langle q_{1_c}, x\rangle \mapsto q_{1_d} \in \delta_2$
- Also, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

Intersection of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$
- Can we build a FA to recognise L?
- Clone the input sequence to apply to the two m/cs M_1 and M_2 and run them in tandem
- String is accepted if *both* m/cs accepts it
- Is this arrangement a FA, if so what is its formal representation?
- $Q = Q_1 \times Q_2$, $q_I = \langle q_{I_1}, q_{I_2} \rangle$
- $F = \{\langle q_1, q_2 \rangle | q_1 \in F_1, q_2 \in F_2\}$
- $\delta = \{\langle\langle q_{1_a}, q_{2_c}\rangle, x\rangle \mapsto \langle q_{1_b}, q_{2_d}\rangle\} | \langle q_{1_a}, x\rangle \mapsto q_{1_b} \in \delta_1 \wedge \langle q_{1_c}, x\rangle \mapsto q_{1_d} \in \delta_2$
- Also, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$
- How to constuct m/c for $\overline{\overline{L_1} \cup \overline{L_2}}$, to be seen later

Concatenation of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$

Concatenation of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
- Can we build a FA to recognise L?

Concatenation of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
- Can we build a FA to recognise L?
- A possibility is to demultiplex the input, feeding it first to M_1, and on acceptance by M_1, to M_2

Concatenation of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
- Can we build a FA to recognise L?
- A possibility is to demultiplex the input, feeding it first to M_1, and on acceptance by M_1, to M_2
- String is accepted if finally accepted by M_2

Concatenation of RLs

- Consider L_1 and L_2 over a common Σ, $L = L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
- Can we build a FA to recognise L?
- A possibility is to demultiplex the input, feeding it first to M_1, and on acceptance by M_1, to M_2
- String is accepted if finally accepted by M_2
- This scheme has a problem though, the input may be splittable into $w_1 w_2$ and also $w_1' w_2'$

Concatenation of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
- Can we build a FA to recognise L?
- A possibility is to demultiplex the input, feeding it first to M_1, and on acceptance by M_1, to M_2
- String is accepted if finally accepted by M_2
- This scheme has a problem though, the input may be splittable into $w_1 w_2$ and also $w_1' w_2'$
- While both w_1 and w_1' may be accepted by M_1, one of w_2 and w_2' may not be accepted by M_2; may other possibilities also

Concatenation of RLs

- Consider L_1 and L_2 over a common Σ, $L = L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
- Can we build a FA to recognise L?
- A possibility is to demultiplex the input, feeding it first to M_1, and on acceptance by M_1, to M_2
- String is accepted if finally accepted by M_2
- This scheme has a problem though, the input may be splittable into $w_1 w_2$ and also $w_1' w_2'$
- While both w_1 and w_1' may be accepted by M_1, one of w_2 and w_2' may not be accepted by M_2; may other possibilities also
- Thus, various splitting options may have to be considered, through *non-determinism*

Concatenation of RLs

- Consider L_1 and L_2 over a common Σ, $L = L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
- Can we build a FA to recognise L?
- A possibility is to demultiplex the input, feeding it first to M_1, and on acceptance by M_1, to M_2
- String is accepted if finally accepted by M_2
- This scheme has a problem though, the input may be splittable into $w_1 w_2$ and also $w_1' w_2'$
- While both w_1 and w_1' may be accepted by M_1, one of w_2 and w_2' may not be accepted by M_2; may other possibilities also
- Thus, various splitting options may have to be considered, through *non-determinism*
- A string w will be said to be $L_1 \circ L_2$ if there exists a way to split w as $w_1 w_2$ such that $w_1 \in L_1$ and $w_2 \in L_2$

Concatenation of RLs

- Consider L_1 and L_2 over a common Σ,
  $L = L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
- Can we build a FA to recognise L?
- A possibility is to demultiplex the input, feeding it first to M_1, and on acceptance by M_1, to M_2
- String is accepted if finally accepted by M_2
- This scheme has a problem though, the input may be splittable into $w_1 w_2$ and also $w_1' w_2'$
- While both w_1 and w_1' may be accepted by M_1, one of w_2 and w_2' may not be accepted by M_2; may other possibilities also
- Thus, various splitting options may have to be considered, through *non-determinism*
- A string w will be said to be $L_1 \circ L_2$ if there exists a way to split w as $w_1 w_2$ such that $w_1 \in L_1$ and $w_2 \in L_2$
- For regular languages L_1 and L_2, we shall seek to show that $L_1 \circ L_2$ is also regular

Non-deterministic FA

- Unlike a DFA, there may be multiple next states on the same inputs

Non-deterministic FA

- Unlike a DFA, there may be multiple next states on the same inputs
- It is permissible to take any of the outgoing transitions

Non-deterministic FA

- Unlike a DFA, there may be multiple next states on the same inputs
- It is permissible to take any of the outgoing transitions
- ϵ-transition from one state to another state without any input may also be permitted

Non-deterministic FA

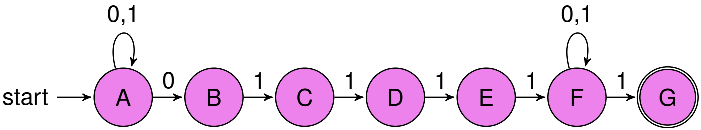
- Unlike a DFA, there may be multiple next states on the same inputs
- It is permissible to take any of the outgoing transitions
- ϵ-transition from one state to another state without any input may also be permitted
- Multiple ways to proceed on a given input

Non-deterministic FA

- Unlike a DFA, there may be multiple next states on the same inputs
- It is permissible to take any of the outgoing transitions
- ϵ-transition from one state to another state without any input may also be permitted
- Multiple ways to proceed on a given input
- String is accepted if there is at least one way to proceed to an accepting step

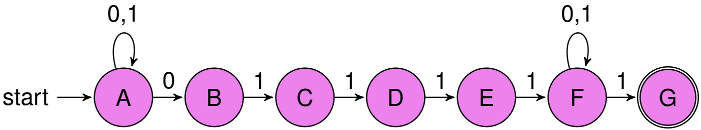
Non-deterministic FA

- Unlike a DFA, there may be multiple next states on the same inputs
- It is permissible to take any of the outgoing transitions
- ϵ-transition from one state to another state without any input may also be permitted
- Multiple ways to proceed on a given input
- String is accepted if there is at least one way to proceed to an accepting step
- Consider the behaviour on 0100011110101 of the following m/c which accepts strings containing 011111



Non-deterministic FA

- Unlike a DFA, there may be multiple next states on the same inputs
- It is permissible to take any of the outgoing transitions
- ϵ-transition from one state to another state without any input may also be permitted
- Multiple ways to proceed on a given input
- String is accepted if there is at least one way to proceed to an accepting step
- Consider the behaviour on 0100011110101 of the following m/c which accepts strings containing 011111



- Many "faulty" choices exist, but accepting state can be reached

Example NFAs

- Any binary string ending with 101

Example NFAs

- Any binary string ending with 101
- Any binary string containing 00 or 11 as a substring

Example NFAs

- Any binary string ending with 101
- Any binary string containing 00 or 11 as a substring
- NFA accepting zero or more 1's or zero or more 10 pairs

Example NFAs

- Any binary string ending with 101
- Any binary string containing 00 or 11 as a substring
- NFA accepting zero or more 1's or zero or more 10 pairs
- NFA for the set of all binary strings that have either the number of 0's odd, or the number of 1's not a multiple of 3, or both

Example NFAs

- Any binary string ending with 101
- Any binary string containing 00 or 11 as a substring
- NFA accepting zero or more 1's or zero or more 10 pairs
- NFA for the set of all binary strings that have either the number of 0's odd, or the number of 1's not a multiple of 3, or both
- NFAs make it easier to conceptually design machines, but how to realise those?

Example NFAs

- Any binary string ending with 101
- Any binary string containing 00 or 11 as a substring
- NFA accepting zero or more 1's or zero or more 10 pairs
- NFA for the set of all binary strings that have either the number of 0's odd, or the number of 1's not a multiple of 3, or both
- NFAs make it easier to conceptually design machines, but how to realise those?
- One can construct a DFA from a given NFA!

Example NFAs

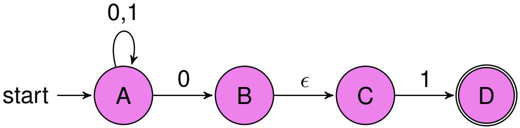
- Any binary string ending with 101
- Any binary string containing 00 or 11 as a substring
- NFA accepting zero or more 1's or zero or more 10 pairs
- NFA for the set of all binary strings that have either the number of 0's odd, or the number of 1's not a multiple of 3, or both
- NFAs make it easier to conceptually design machines, but how to realise those?
- One can construct a DFA from a given NFA!
- So, NFAs are not more powerful recognisers than DFAs

Example NFAs

- Any binary string ending with 101
- Any binary string containing 00 or 11 as a substring
- NFA accepting zero or more 1's or zero or more 10 pairs
- NFA for the set of all binary strings that have either the number of 0's odd, or the number of 1's not a multiple of 3, or both
- NFAs make it easier to conceptually design machines, but how to realise those?
- One can construct a DFA from a given NFA!
- So, NFAs are not more powerful recognisers than DFAs
- Key observation: could be at multiple NFA states at once

Example NFAs

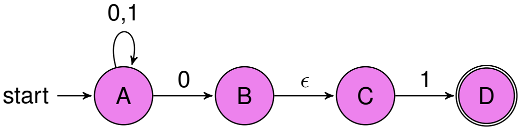
- Any binary string ending with 101
- Any binary string containing 00 or 11 as a substring
- NFA accepting zero or more 1's or zero or more 10 pairs
- NFA for the set of all binary strings that have either the number of 0's odd, or the number of 1's not a multiple of 3, or both
- NFAs make it easier to conceptually design machines, but how to realise those?
- One can construct a DFA from a given NFA!
- So, NFAs are not more powerful recognisers than DFAs
- Key observation: could be at multiple NFA states at once



NFA formalised

NFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, where

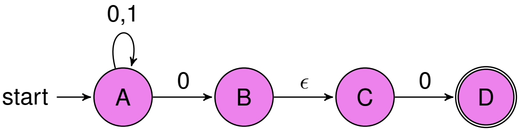
- Q Finite set of states
- Σ Alphabet, finite set of symbols, $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$
- δ Transition function, $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$
  from a given state, on a given symbol or ϵ, transition is to a set of states
- q_I Starting or initial state
- F Set of accepting or final states, $F \subseteq Q$



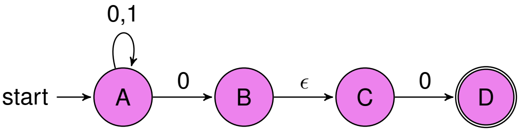
Theorem

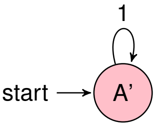
Every NFA M has an equivalent DFA M′

NFA to DFA

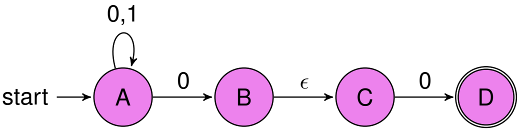


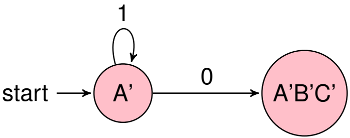
NFA to DFA



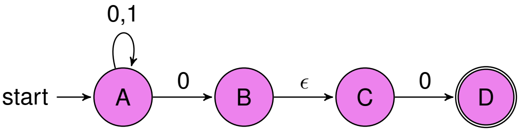


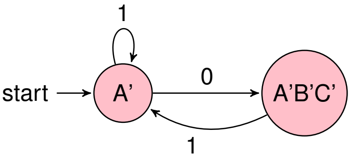
NFA to DFA



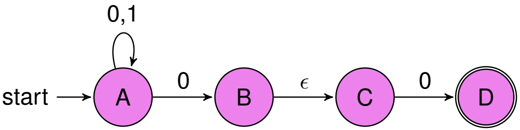


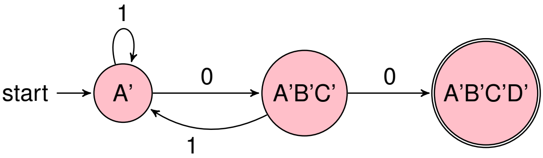
NFA to DFA



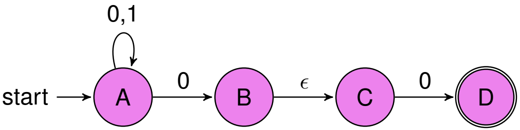


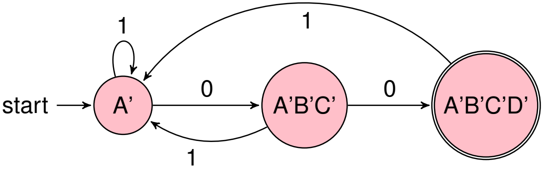
NFA to DFA



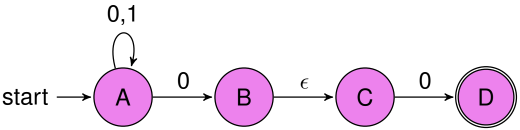


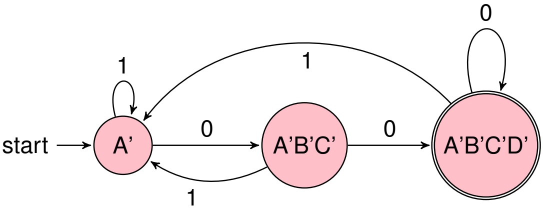
NFA to DFA



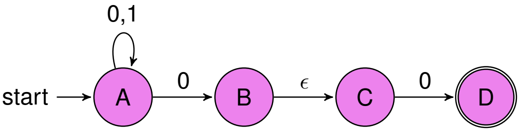


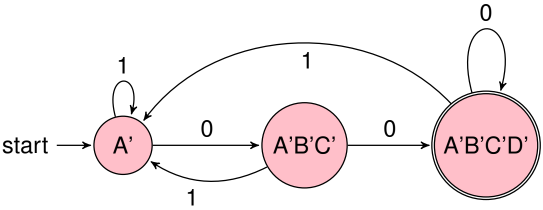
NFA to DFA

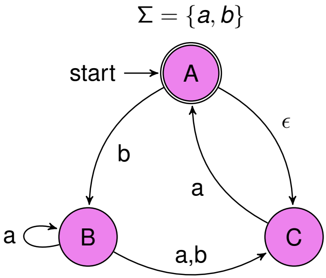




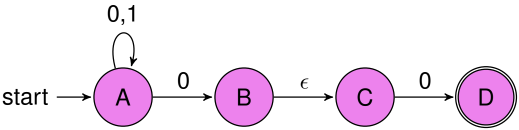
NFA to DFA

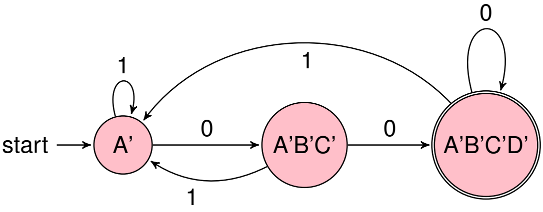

0,1
start
A
0
B
ϵ
C
0
D


1
1
0
start
A'
0
A'B'C'
0
A'B'C'D'
1

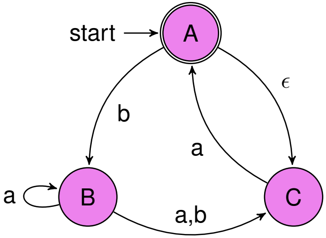

$\Sigma = \{a, b\}$
start
A
ϵ
b
a
a
B
a,b
C

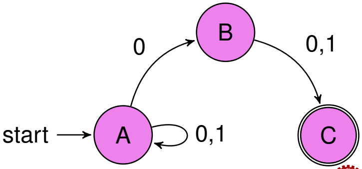
NFA to DFA





$\Sigma = \{a, b\}$



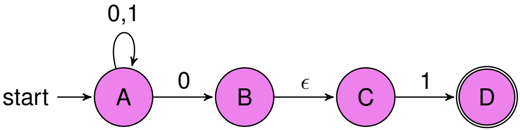


ϵ-closure and subset construction

- NFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, construct DFA $M' = \langle Q', \Sigma, \delta', q'_I, F' \rangle$

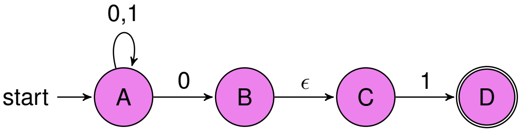
ϵ-closure and subset construction

- NFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, construct DFA $M' = \langle Q', \Sigma, \delta', q'_I, F' \rangle$



ϵ-closure and subset construction

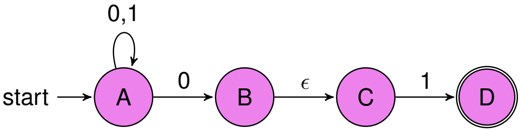
- NFA $M = \langle Q, \Sigma, \delta, q_I, F\rangle$, construct DFA $M' = \langle Q', \Sigma, \delta', q_I', F'\rangle$



- The ϵ-closure of a given set of NFA states S is the set of states reachable by a sequence of zero or more ϵ-transitions from the states in S, denote as $E(S)$

ϵ-closure and subset construction

- NFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, construct DFA $M' = \langle Q', \Sigma, \delta', q_I', F' \rangle$

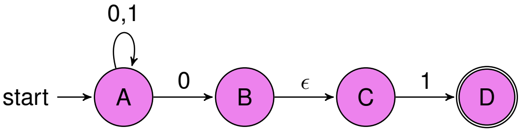


- The ϵ-closure of a given set of NFA states S is the set of states reachable by a sequence of zero or more ϵ-transitions from the states in S, denote as $E(S)$

  $\mathbf{E}(\{B\}) = \{B, C\}$, $\mathbf{E}(\{C\}) = \{C\}$

ϵ-closure and subset construction

- NFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, construct DFA $M' = \langle Q', \Sigma, \delta', q'_I, F' \rangle$



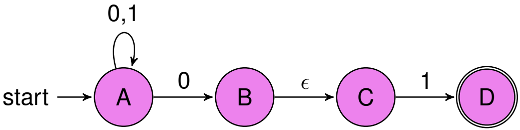
- The ϵ-closure of a given set of NFA states S is the set of states reachable by a sequence of zero or more ϵ-transitions from the states in S, denote as $E(S)$

  $\mathbf{E}(\{B\}) = \{B, C\}$, $\mathbf{E}(\{C\}) = \{C\}$

- The first subset introduced in Q' is $q'_I = \mathbf{E}(\{q_I\})$, $q'_I \in Q'$

ϵ-closure and subset construction

- NFA $M = \langle Q, \Sigma, \delta, q_I, F\rangle$, construct DFA $M' = \langle Q', \Sigma, \delta', q_I', F'\rangle$

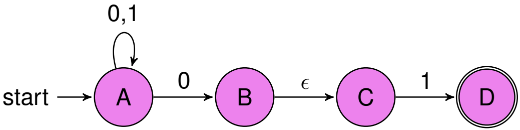


- The ϵ-closure of a given set of NFA states S is the set of states reachable by a sequence of zero or more ϵ-transitions from the states in S, denote as $E(S)$

  $\mathbf{E}(\{B\}) = \{B, C\}$, $\mathbf{E}(\{C\}) = \{C\}$
- The first subset introduced in Q' is $q_I' = \mathbf{E}(\{q_I\})$, $q_I' \in Q'$
- For $q' \in Q'$, $\delta'(q', a) = \bigcup_{q \in q'} \mathbf{E}(\delta(q, a))$, $\delta'(q', a) \in Q'$

ϵ-closure and subset construction

- NFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, construct DFA $M' = \langle Q', \Sigma, \delta', q'_I, F' \rangle$

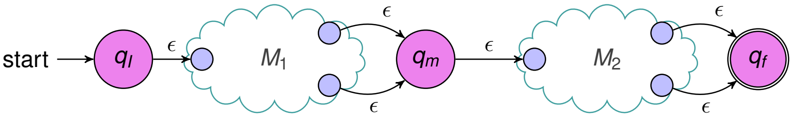


- The ϵ-closure of a given set of NFA states S is the set of states reachable by a sequence of zero or more ϵ-transitions from the states in S, denote as $E(S)$

  $\mathbf{E}(\{B\}) = \{B, C\}$, $\mathbf{E}(\{C\}) = \{C\}$
- The first subset introduced in Q' is $q'_I = \mathbf{E}(\{q_I\})$, $q'_I \in Q'$
- For $q' \in Q'$, $\delta'(q', a) = \bigcup_{q \in q'} \mathbf{E}(\delta(q, a))$, $\delta'(q', a) \in Q'$
- Any state of Q' containing a final state of M is a final state of M': $\forall q' \in Q'. [\exists f \in F. q' \cap f \neq \phi \Rightarrow q' \in F']$

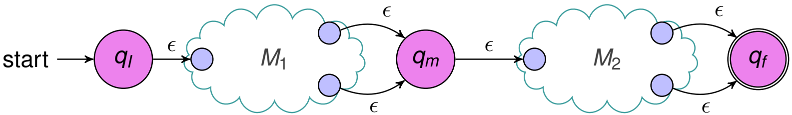
NFA recognisers

- $L_1 \circ L_2$:

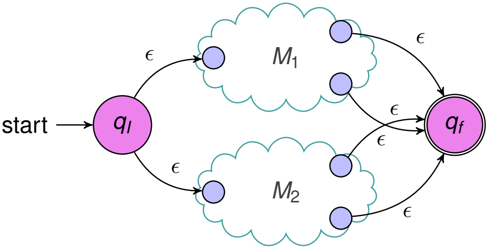


NFA recognisers

- $L_1 \circ L_2$:

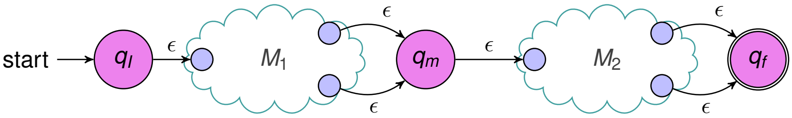


- $L_1 \cup L_2$:

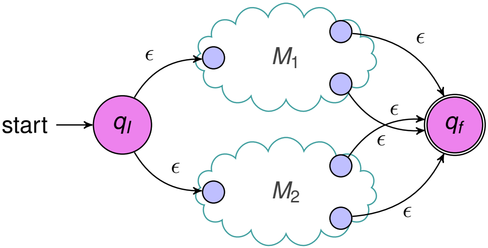


NFA recognisers

- $L_1 \circ L_2$:



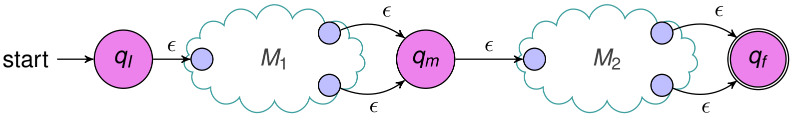
- $L_1 \cup L_2$:



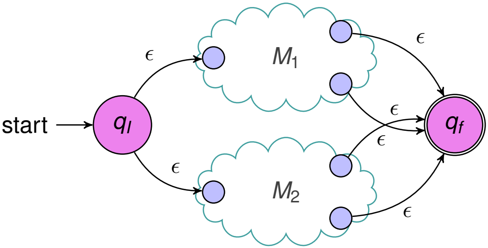
- For $\overline{L}$, convert the NFA to a DFA and then interchange accepting and non-accepting states

NFA recognisers

- $L_1 \circ L_2$:



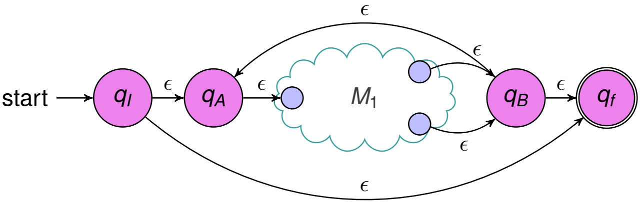
- $L_1 \cup L_2$:



- For $\overline{L}$, convert the NFA to a DFA and then interchange accepting and non-accepting states
- For $L_1 \cap L_2$, use recogniser for $\overline{\overline{L_1} \cup \overline{L_2}}$

NFA recogniser for Kleene star

$L^* = \{x_1 x_2 \dots x_k | k \geq 0 \wedge x_i \in L\}$
$L : M = \langle Q, \Sigma, \delta, q_I, F\rangle$ for L^* from $L_1 : M_1 = \langle Q_1, \Sigma, \delta_1, q_{I_1}, F_1\rangle$



- Initial state of M is q_I, $F = \{q_f\}$
- $Q = \{q_I, q_A, q_B, q_f\} \cup Q_1$
- For $q \in Q, a \in \Sigma_\epsilon$, $\delta(q, a) =$

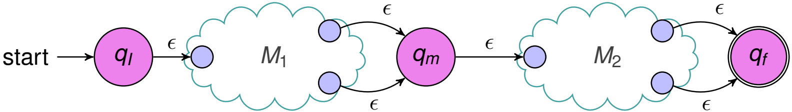
$$\begin{cases} \delta_1(q,a) & q \in \left(Q_1 \cap \overline{F_1}\right) \\ \delta_1(q,a) \cup \{q_B\} & q \in F_1 \end{cases}$$

for $q \in Q, a \in \Sigma_\epsilon$, $\delta(q, a) =$

$$\begin{cases} \{q_A, q_f\} & q = q_I, a = \epsilon \\ \{q_{I_1}\} & q = q_A, a = \epsilon \\ \{q_A, q_f\} & q = q_B, a = \epsilon \\ \{\} & q = q_I, a \neq \epsilon \\ \{\} & q = q_A, a \neq \epsilon \\ \{\} & q = q_B, a \neq \epsilon \end{cases}$$

Formalisation of $L_1 \circ L_2$

- From $L_1 : M_1 = \langle Q_1, \Sigma, \delta_1, q_{I_1}, F_1 \rangle$, $L_2 : M_2 = \langle Q_2, \Sigma, \delta_2, q_{I_2}, F_2 \rangle$ construct $L = L_1 \circ L_2 : M = \langle Q, \Sigma, \delta, q_I, F \rangle$



- $Q = \{q_I, q_m, q_f\} \cup Q_1 \cup Q_2$
- Initial state of M is q_I, $F = \{q_f\}$
- For $q \in Q, a \in \Sigma_\epsilon$, $\delta(q,a) =$

$$\begin{cases} \{q_{I_1}\} & q = q_I, a = \epsilon \\ \{\} & q = q_I, a \neq \epsilon \\ \delta_1(q,a) & q \in \left(Q_1 \cap \overline{F_1}\right) \\ \delta_1(q,a) \cup \{q_m\} & q \in F_1 \end{cases}$$

for $q \in Q, a \in \Sigma_\epsilon$, $\delta(q,a) =$

$$\begin{cases} \{q_{I_2}\} & q = q_m, a = \epsilon \\ \{\} & q = q_m, a \neq \epsilon \\ \delta_2(q,a) & q \in \left(Q_2 \cap \overline{F_2}\right) \\ \delta_2(q,a) \cup \{q_f\} & q \in F_2 \end{cases}$$

- Formalise the other diagramatic constructions ...

Closure properties of RLs

- Closed under complementation
  if L_1 is a regular language, so is $\overline{L_1}$
- Closed under union
  if L_1 and L_2 are regular languages, so is $L_1 \cup L_2$
- Closed under concatenation
  if L_1 and L_2 are regular languages, so is $L_1 \circ L_2$
- Closed under Kleene star
  if L_1 is a regular language, so is L_1^*
- Closed under intersection
  if L_1 and L_2 are regular languages, so is $L_1 \cap L_2$

Consider $\Sigma = \{a, b, c, \ldots, z\}$, $L_1 = \{aa, b\}$, $L_2 = \{x, yy\}$ to work out examples of above

Practice example of NFA

1. Give an NFA recognizing the language $(01 \cup 001 \cup 010)^\star$
   i. Show that the class of regular languages are closed under complementation.
   ii. Show that the class of regular languages are closed under string reversal.
2. Let $A/B = \{w | wx \in A \text{ for some } x \in B\}$. Show that if A is regular and B is any language then A/B is regular.
3. For languages A and B, let the **shuffle** of A and B be the language $\{w | w = a_1 b_1 \ldots a_k b_k\}$, where $a_1 \ldots a_k \in A$ and $b_1 \ldots b_k \in B$, each $a_i, b_i \in \Sigma^\star$. Show that the class of regular languages are closed under shuffle.

Regular expressions

REs are recursively described over a given alphabet Σ as follows:

Base clauses $a \in \Sigma$, ϵ is the empty string

- a is a RE, denoting $\{a\}$
- ϵ is a RE, denoting $\{\epsilon\}$
- $\varnothing$ is a RE

REs to simplify $a(b^*)$, $(ab)|c$, $a(b|c)$, aa^*, aa^+, $a\epsilon$, $a|\epsilon$, $a\varnothing$, $a|\varnothing$

Regular expressions

REs are recursively described over a given alphabet Σ as follows:

Base clauses $a \in \Sigma$, ϵ is the empty string

- a is a RE, denoting $\{a\}$
- ϵ is a RE, denoting $\{\epsilon\}$
- $\varnothing$ is a RE

Inductive clauses given REs R_1 and R_2 or R,

- $R_1 \circ R_2$ is a RE, representing concatenation
- $R_1 | R_2$ is a RE, representing alternation (set union)
- R^* is a RE, representing the Kleene star of R
- (R) is a RE, for grouping, sometimes optional

REs to simplify $a(b^*)$, $(ab)|c$, $a(b|c)$, aa^*, aa^+, $a\epsilon$, $a|\epsilon$, $a\varnothing$, $a|\varnothing$

Regular expressions

REs are recursively described over a given alphabet Σ as follows:

Base clauses $a \in \Sigma$, ϵ is the empty string

- a is a RE, denoting $\{a\}$
- ϵ is a RE, denoting $\{\epsilon\}$
- $\varnothing$ is a RE

Inductive clauses given REs R_1 and R_2 or R,

- $R_1 \circ R_2$ is a RE, representing concatenation
- $R_1 | R_2$ is a RE, representing alternation (set union)
- R^* is a RE, representing the Kleene star of R
- (R) is a RE, for grouping, sometimes optional

Binding priorities and associativity are as described below

- Kleene star has highest binding priority
- concatenation has intermediate binding priority
- alternation has lowest binding priority
- left associativity is used

REs to simplify $a(b^*)$, $(ab)|c$, $a(b|c)$, aa^*, aa^+, $a\epsilon$, $a|\epsilon$, $a\varnothing$, $a|\varnothing$

Regular expressions

REs are recursively described over a given alphabet Σ as follows:

Base clauses $a \in \Sigma$, ϵ is the empty string

- a is a RE, denoting $\{a\}$
- ϵ is a RE, denoting $\{\epsilon\}$
- $\varnothing$ is a RE

Inductive clauses given REs R_1 and R_2 or R,

- $R_1 \circ R_2$ is a RE, representing concatenation
- $R_1|R_2$ is a RE, representing alternation (set union)
- R^* is a RE, representing the Kleene star of R
- (R) is a RE, for grouping, sometimes optional

Binding priorities and associativity are as described below

- Kleene star has highest binding priority
- concatenation has intermediate binding priority
- alternation has lowest binding priority
- left associativity is used

REs to simplify $a(b^*)$, $(ab)|c$, $a(b|c)$, aa^*, aa^+, $a\epsilon$, $a|\epsilon$, $a\varnothing$, $a|\varnothing$

Regular expressions

REs are recursively described over a given alphabet Σ as follows:

Base clauses $a \in \Sigma$, ϵ is the empty string

- a is a RE, denoting $\{a\}$
- ϵ is a RE, denoting $\{\epsilon\}$
- $\varnothing$ is a RE

Inductive clauses given REs R_1 and R_2 or R,

- $R_1 \circ R_2$ is a RE, representing concatenation
- $R_1|R_2$ is a RE, representing alternation (set union)
- R^* is a RE, representing the Kleene star of R
- (R) is a RE, for grouping, sometimes optional

Binding priorities and associativity are as described below

- Kleene star has highest binding priority
- concatenation has intermediate binding priority
- alternation has lowest binding priority
- left associativity is used

REs to simplify $a(b^*)$, $(ab)|c$, $a(b|c)$, aa^*, aa^+, $a\epsilon$, $a|\epsilon$, $a\varnothing$, $a|\varnothing$

Any language described by REs is regular, by earlier constructions

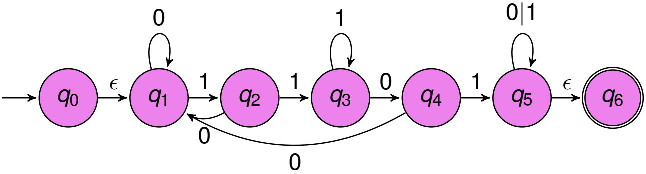
Practice problems of REs

Write regular expressions for the following languages. In all parts alphabet is $\{0, 1\}$.

1. $L = \{w|w$ does not contain the substring 110$\}$.
2. $L = \{w|w$ is any string except 11 and 111$\}$.
3. $L = \{w|w$ has odd number of 1's$\}$.
4. $L = \{w|w$ contains at least two 0's or exactly two 1's$\}$.
5. $L = \{w|$ every 1 in w is either at the end of w or is immediately followed by another 1$\}$.
6. $L = \{w|w$ has equal number of 0's and 1's, such that no prefix has two more 0's than 1's, nor two more 1's than 0's $\}$.
7. $L = \{w|$ every pair of adjacent 0's in w appear before any pair of adjacent 1's in $w\}$.
8. $L = \{w|w$ is the binary representation of numbers divisible by 5 $\}$.

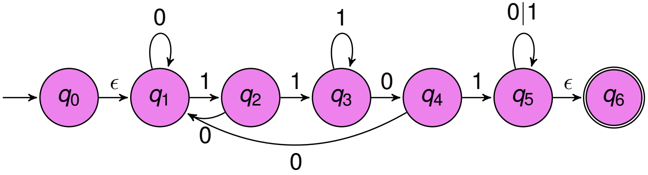
Find a regular expression that denotes all bit strings whose value, when interpreted as a binary integer, is greater than or equal to 40.

Representing FAs as RLs



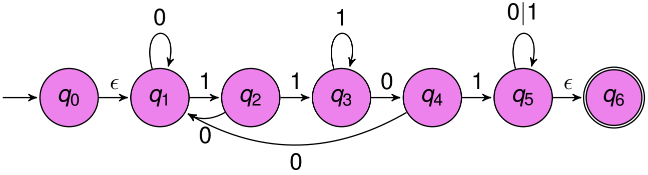
- Edges are labelled by REs, single start and end states, GNFA

Representing FAs as RLs



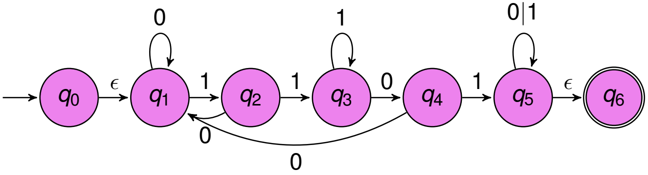
- Edges are labelled by REs, single start and end states, GNFA
- Can we remove q_1?

Representing FAs as RLs



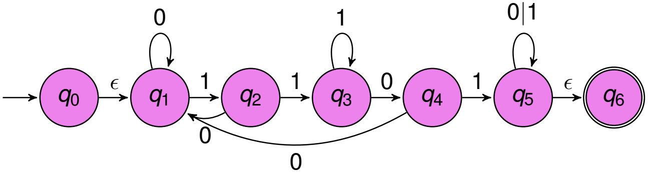
- Edges are labelled by REs, single start and end states, GNFA
- Can we remove q_1? All nodes to q_1 should go to sucessors of q_1

Representing FAs as RLs



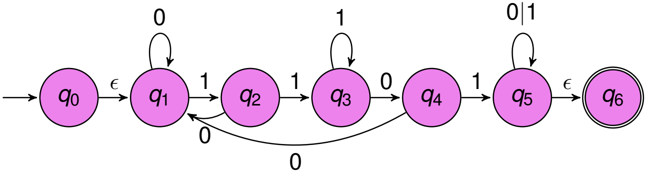
- Edges are labelled by REs, single start and end states, GNFA
- Can we remove q_1? All nodes to q_1 should go to sucessors of q_1
- q_0 to q_2 on $\epsilon \cdot 0^* \cdot 1 \equiv 0^*1$

Representing FAs as RLs



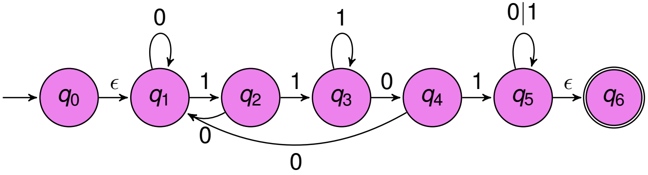
- Edges are labelled by REs, single start and end states, GNFA
- Can we remove q_1? All nodes to q_1 should go to sucessors of q_1
- q_0 to q_2 on $\epsilon \cdot 0^* \cdot 1 \equiv 0^*1$; q_2 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1)

Representing FAs as RLs



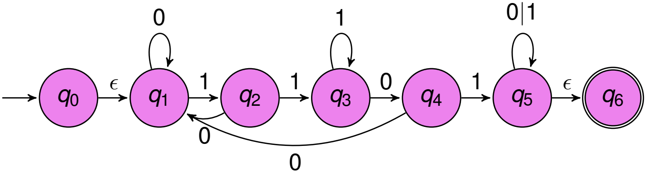
- Edges are labelled by REs, single start and end states, GNFA
- Can we remove q_1? All nodes to q_1 should go to sucessors of q_1
- q_0 to q_2 on $\epsilon \cdot 0^* \cdot 1 \equiv 0^*1$; q_2 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1); q_4 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1)

Representing FAs as RLs

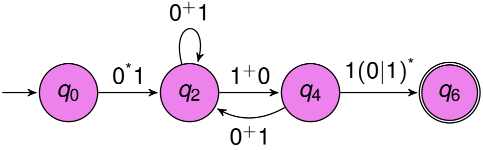


- Edges are labelled by REs, single start and end states, GNFA
- Can we remove q_1? All nodes to q_1 should go to sucessors of q_1
- q_0 to q_2 on $\epsilon \cdot 0^* \cdot 1 \equiv 0^*1$; q_2 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1); q_4 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1)
- Dropping q_3, q_2 to q_4 on 1^+0 and dropping q_5, q_4 to q_6 on $1(0|1)^*$

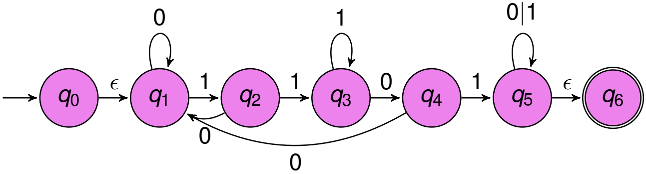
Representing FAs as RLs



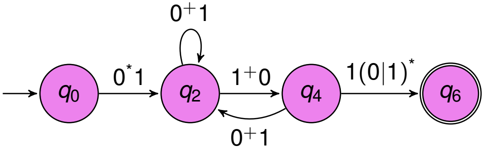
- Edges are labelled by REs, single start and end states, GNFA
- Can we remove q_1? All nodes to q_1 should go to sucessors of q_1
- q_0 to q_2 on $\epsilon \cdot 0^* \cdot 1 \equiv 0^*1$; q_2 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1); q_4 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1)
- Dropping q_3, q_2 to q_4 on 1^+0 and dropping q_5, q_4 to q_6 on $1(0|1)^*$

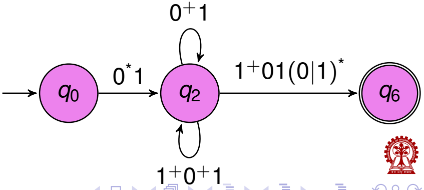


Representing FAs as RLs

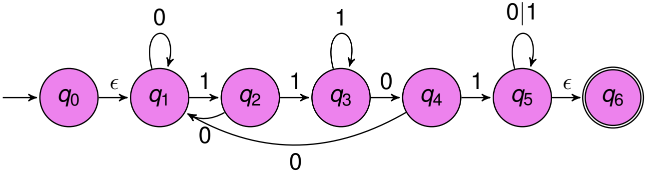


- Edges are labelled by REs, single start and end states, GNFA
- Can we remove q_1? All nodes to q_1 should go to sucessors of q_1
- q_0 to q_2 on $\epsilon \cdot 0^* \cdot 1 \equiv 0^*1$; q_2 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1); q_4 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1)
- Dropping q_3, q_2 to q_4 on 1^+0 and dropping q_5, q_4 to q_6 on $1(0|1)^*$

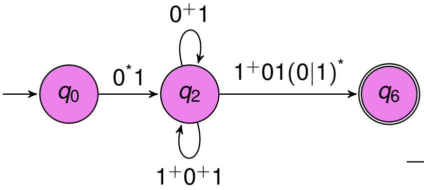


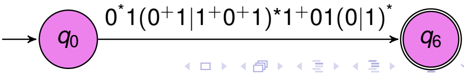


Representing FAs as RLs



- Edges are labelled by REs, single start and end states, GNFA
- Can we remove q_1? All nodes to q_1 should go to sucessors of q_1
- q_0 to q_2 on $\epsilon \cdot 0^* \cdot 1 \equiv 0^*1$; q_2 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1); q_4 to q_2 on $0 \cdot 0^* \cdot 1 \equiv 00^*1$ (also 0^+1)
- Dropping q_3, q_2 to q_4 on 1^+0 and dropping q_5, q_4 to q_6 on $1(0|1)^*$





GNFA

A generalised non-deterministic finite automaton is the quintuple $M = \langle Q, \Sigma, \delta, q_I, q_F \rangle$

- Q The set of states
- Σ The input alphabet
- q_I The initial state
- q_F The accepting state

  Let R be the set of all regular expressions over the input alphabet Σ
- δ The transition function $\delta : (Q \setminus \{q_F\}) \times (Q \setminus \{q_I\}) \rightarrow R$

GNFA algorithm

- Start with a DFA
- Introduce a new start and a new final state
- While intermediate states remain do
  - Pick any state s_x and eliminate by constructing direct edge from predecessor state s_a to successor state s_b
  - If label on edge from s_a to s_x is L_a, s_x to s_b is L_b and s_x to s_x is L_x, add label from s_a to s_b as $L_a L_x^* L_b$ (no L_x^* if no loop on s_x)

GNFA algorithm

- Start with a DFA
- Introduce a new start and a new final state
- While intermediate states remain do
  - Pick any state s_x and eliminate by constructing direct edge from predecessor state s_a to successor state s_b
  - If label on edge from s_a to s_x is L_a, s_x to s_b is L_b and s_x to s_x is L_x, add label from s_a to s_b as $L_a L_x^* L_b$ (no L_x^* if no loop on s_x)

Lemma

If a language is described by a regular expression, then it is regular

Lemma

If a language is regular, then it can be described by a regular expression

Theorem

A language is regular iff it is described by a regular expression

Linear grammar

Definition (Linear grammar)

A linear grammar is a context-free grammar that has at most one nonterminal in the right hand side of each of its productions

Definition (Left linear grammar)

Linear grammar where all nonterminals in right hand sides are at the left ends

Definition (Right linear grammar)

Linear grammar where all nonterminals in right hand sides are at the right ends

Linear grammars from FA and vice versa

Given $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, proceed as follows to generate RL or LL grammars:

- Augment M with a new start state S [$\delta(S, \epsilon) = q_I$] and a new final state F [$\forall q_F \in F, \delta(q_F, \epsilon) = F$]
- For transition $\delta(A, \mathtt{a}) = B$

  to get RL grammar add production $A \to \mathtt{a}B$

  Future of A is $\mathtt{a}$ followed by future of B

  to get LL grammar add production $B \to A\mathtt{a}$

  Past of B is $\mathtt{a}$ preceeded by past of B
- For the RL grammar, S is the start symbol, also add $F \to \epsilon$ [the final state has no future]
- For the LL grammar, F is the start symbol, also add $S \to \epsilon$ [the initial state has no past]

Following the reverse procedure, given a RL or LL grammar, the corresponding FA can be constructed

Linear grammars from FA and vice versa (contd.)

Lemma

If a language is described by a linear grammar, then it is regular

Lemma

If a language is regular, then it can be described by a linear grammar

Theorem

A language is regular iff it is described by a linear grammar

Non-RLs

A finite automaton can only remember finitely many things using its finite set of states

What about the following languages?

Non-RLs

A finite automaton can only remember finitely many things using its finite set of states

What about the following languages?

- Strings having underlined text
  underlining is done using BS and _

Non-RLs

A finite automaton can only remember finitely many things using its finite set of states

What about the following languages?

- Strings having underlined text
  underlining is done using BS and _
- $B = \{w|\text{count("01")=count("10")}\}$
  001111001011101

Non-RLs

A finite automaton can only remember finitely many things using its finite set of states

What about the following languages?

- Strings having underlined text
  underlining is done using BS and _
- $B = \{w | \text{count("01")=count("10")}\}$
  0011110010111 01
- $C = \{w | \text{count("0")=count("1")}\}$
  001111001011101

Non-RLs

A finite automaton can only remember finitely many things using its finite set of states

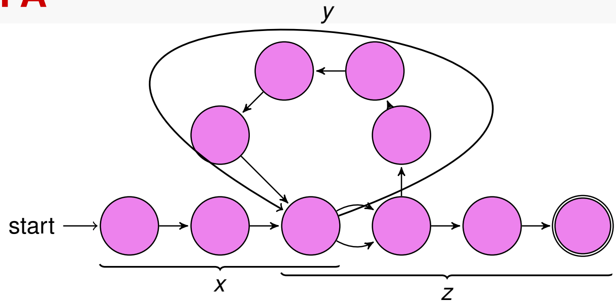
What about the following languages?

- Strings having underlined text
  underlining is done using BS and _
- $B = \{w | \text{count("01")=count("10")}\}$
  001111001011101
- $C = \{w | \text{count("0")=count("1")}\}$
  001111001011101
- $D = \{0^n 1^n | n \geq 0\}$
  0000000111111111

Existence of non-regular languages

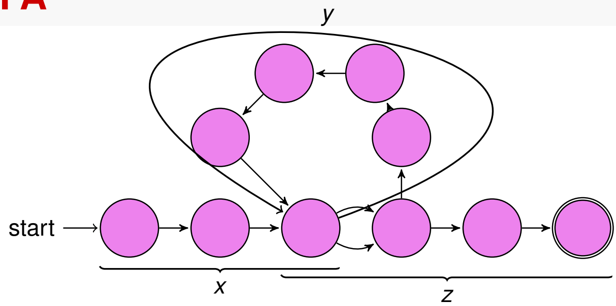
- The set of all languages over $\Sigma = \{0, 1\}$ is uncountable – powerset of the Σ^*
- The set of regular languages is countable – can be enumerated using the DFA
- Follows from above

Pumping in FA



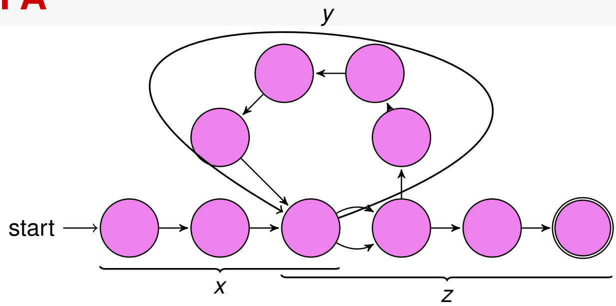
- All strings of the form xy^iz, $i \geq 0$ are accepted (are in the language) L

Pumping in FA



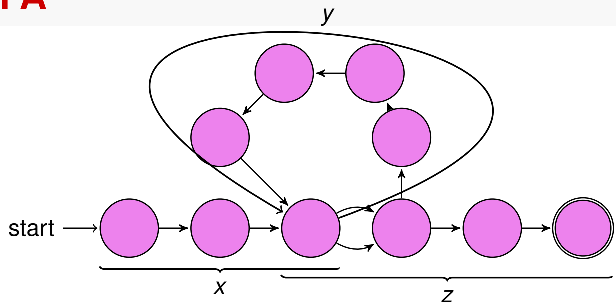
- All strings of the form xy^iz, $i \geq 0$ are accepted (are in the language) L
- For a regular language L there is a finite state recogniser say M with Q as its set of states, let $p = |Q|$

Pumping in FA



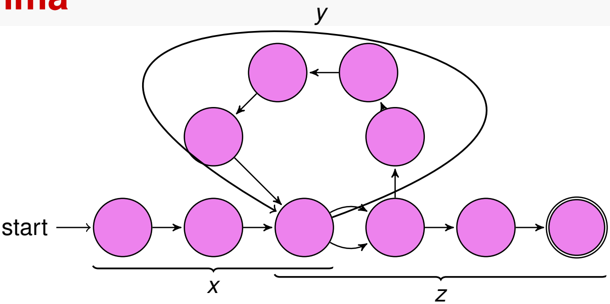
- All strings of the form xy^iz, $i \geq 0$ are accepted (are in the language) L
- For a regular language L there is a finite state recogniser say M with Q as its set of states, let $p = |Q|$
- A string $s \in L$ with $|s| \geq p$ must contain a cycle

Pumping in FA



- All strings of the form xy^iz, $i \geq 0$ are accepted (are in the language) L
- For a regular language L there is a finite state recogniser say M with Q as its set of states, let $p = |Q|$
- A string $s \in L$ with $|s| \geq p$ must contain a cycle
- A string $s \in L$ containing a cycle y can be modified to s' by *pumping* in any number of copies of y so that $s' \in L$

Pumping lemma



Lemma

Let L be a regular language, then there exists an integer $p \geq 1$ depending only on L such that every string $w \in L$, $|w| \geq p$ can be written as $w = xyz$, satisfying:

- $xy^iz \in L$ *for any* $i \geq 0$
- $|y| > 0$ – *cycle has at least one edge in it*
- $|xy| \leq p$ – *cycle is seen before the string gets longer that p*

Pumping lemma (contd.)

Proof.

- Let $p = |Q|$, where $M = \langle Q, \Sigma, \delta, q_I, F \rangle$ is a recogniser for L
- For $s \in L$, $|s| \geq p$, let q_0 be the start state and let $q_1, \ldots, q_p$ be the sequence of the next p states visited as the string is recognised/generated
- By the pigeon hole principle let q_l be a state which is revisited
- Let y be the string from the first instance of q_l to a repeated instance of q_l
- Now s may be written as xyz such that: $xy^iz \in L$ for any $i \geq 0$, $|y| > 0$ and $|xy| \leq p$

□

L not regular by PL

Proof scheme for L not regular by PL

- Assume L is regular, with pumping length p
- Find a long enough string $s \in L$, $|s| \geq p$
- Express s in the form xyz
- All strings of the form xy^iz must, therefore, be in L
- For some i show that $xy^iz \notin L$ leading to a contradiction

PL applications

$a^n b^n$ is not regular

- Consider $L = \{a^n b^n | n \geq 0\}$ over $\Sigma = \{a, b\}$
- Let $s \in L$ be $s = a^p b^p$, clearly $|s| \geq p$, so by PL, $s = xyz$ with $|xy| \leq p$ and $|y| \geq 1$, so $xy^i z \in L, \forall i \geq 0$
- Using $|xy| \leq p$, we know y only consists of instances of a
- As $|y| \geq 1$, it contains at least one a
- Now pump y as $xy^2 z$ has more of a's than b's (no b was added)
- Thus, $xy^2 z \notin L$ – a contradiction
- Thus, the assumption that L is regular must be incorrect, hence L is not regular

PL applications

$a^n b^n$ is not regular

- Consider $L = \{a^n b^n | n \geq 0\}$ over $\Sigma = \{a, b\}$
- Let $s \in L$ be $s = a^p b^p$, clearly $|s| \geq p$, so by PL, $s = xyz$ with $|xy| \leq p$ and $|y| \geq 1$, so $xy^i z \in L, \forall i \geq 0$
- Using $|xy| \leq p$, we know y only consists of instances of a
- As $|y| \geq 1$, it contains at least one a
- Now pump y as xy^2z has more of a's than b's (no b was added)
- Thus, $xy^2z \notin L$ – a contradiction
- Thus, the assumption that L is regular must be incorrect, hence L is not regular

- Similarly, language of balanced parentheses is not regular
- Similarly, language of equal number of 0'1 and 1's is not regular

Practice problems of Pumping Lemma (RLs)

Which of these languages are regular?

1. $L = \{0^{2n} | n \geq 1\}$.
2. $L = \{0^m 1^n 0^{n+m} | m \geq 1 \text{ and } n \geq 1\}$.
3. $L = \{x | x = x^R\}$ (reverse of x, palindrome).
4. $L = \{010010001 \ldots 0^i 1 | i \text{ is any positive integer}\}$
5. $L = \{0^n | n \text{ is a prime}\}$.
6. $L = \{w | w \text{ has equal number of 0 and 1}\}$.
7. $L = \{w | w \text{ has equal number of 01 and 10}\}$.
8. $L = \{1^k y | y \in \{0,1\}^\star$ and y contains at most k 1's, for $k \geq 1\}$.
9. $L = \{a^i b^j c^k | i, j, k \geq 0 \text{ and if } i = 1 \text{ then } j = k\}$
10. $L = \{\omega\omega^R x | \omega, \omega^R, x \in \Sigma^+\}$.
11. $L = \{a^i b^j c^k | i \neq j \text{ or } j \neq k\}$.

Myhill-Nerode theorem

- For a language L and strings x and y over Σ, a string z is a *distinguishing extension* if either $xz \in L$ or $yz \in L$

Myhill-Nerode theorem

- For a language L and strings x and y over Σ, a string z is a *distinguishing extension* if either $xz \in L$ or $yz \in L$
- For a language L over Σ, two words $x, y \in \Sigma^*$ are L-equivalent ($x \equiv_L y$) iff for all words $z \in \Sigma^*$, we have $xz \in L$ iff $yz \in L$ – no z exists to distinguish x and y

Myhill-Nerode theorem

- For a language L and strings x and y over Σ, a string z is a *distinguishing extension* if either $xz \in L$ or $yz \in L$
- For a language L over Σ, two words $x, y \in \Sigma^*$ are L-equivalent ($x \equiv_L y$) iff for all words $z \in \Sigma^*$, we have $xz \in L$ iff $yz \in L$ – no z exists to distinguish x and y
- For a $M = \langle Q, \Sigma, \delta, q_I, F\rangle$, two words $x, y \in \Sigma^*$ are M-equivalent ($x \equiv_M y$) iff $\delta^*(q_I, x) = \delta^*(q_I, x)$ – both x and y take M from q_I to the same state

Myhill-Nerode theorem

- For a language L and strings x and y over Σ, a string z is a *distinguishing extension* if either $xz \in L$ or $yz \in L$
- For a language L over Σ, two words $x, y \in \Sigma^*$ are L-equivalent ($x \equiv_L y$) iff for all words $z \in \Sigma^*$, we have $xz \in L$ iff $yz \in L$ – no z exists to distinguish x and y
- For a $M = \langle Q, \Sigma, \delta, q_I, F \rangle$, two words $x, y \in \Sigma^*$ are M-equivalent ($x \equiv_M y$) iff $\delta^*(q_I, x) = \delta^*(q_I, x)$ – both x and y take M from q_I to the same state
- Clearly, $\equiv_L$ and $\equiv_M$ are an equivalence relations

Myhill-Nerode theorem

- For a language L and strings x and y over Σ, a string z is a *distinguishing extension* if either $xz \in L$ or $yz \in L$
- For a language L over Σ, two words $x, y \in \Sigma^*$ are L-equivalent ($x \equiv_L y$) iff for all words $z \in \Sigma^*$, we have $xz \in L$ iff $yz \in L$ – no z exists to distinguish x and y
- For a $M = \langle Q, \Sigma, \delta, q_I, F\rangle$, two words $x, y \in \Sigma^*$ are M-equivalent ($x \equiv_M y$) iff $\delta^*(q_I, x) = \delta^*(q_I, x)$ – both x and y take M from q_I to the same state
- Clearly, $\equiv_L$ and $\equiv_M$ are an equivalence relations
- Further, $\equiv_M$ has only as many equivalence classes as $|Q|$

Myhill-Nerode theorem (contd.)

Lemma

If $A = \mathbf{L}(M)$ for a DFA M then for any $x, y \in \Sigma^*$ if $x \equiv_M y$ then $x \equiv_A y$

Proof.

- Suppose that $A = \mathbf{L}(M)$, then $w \in L \leftrightarrow \delta^*(q_I, w) \in F$
- Suppose also that $x \equiv_M y$, then $\delta^*(q_I, x) = \delta^*(q_I, x)$
- Let $z \in \Sigma^*$, clearly $\delta^*(q_I, xz) = \delta^*(q_I, xz)$, therefore,
  $xz \in A \Leftrightarrow \delta^*(q_I, xz) \in F \Leftrightarrow \delta^*(q_I, yz) \in F \Leftrightarrow yz \in A$
- Thus, $x \equiv_A y$

$\square$

Observation

Whenever two elements arrive at the same state of M they are in the same equivalence class of $\equiv_A$; meaning that each equivalence class of $\equiv_A$ is a union of equivalence classes of $\equiv_M$

Myhill-Nerode theorem (contd.)

Corollary

If A is regular then $\equiv_A$ has a finite number of equivalence classes

Proof.

Let M be a DFA such that $A = \mathbf{L}(M)$, the lemma shows that $\equiv_A$ has at most as many equivalence classes as $\equiv_M$, which equals the number of states of M □

Theorem (Myhill-Nerode)

L is regular if and only if $\equiv_L$ has a finite number of equivalence classes (which also corresponds to the minimum number of states for a recogniser of L)

Proof.

Will show is that if $\equiv_A$ has a finite number of equivalence classes then we can build a DFA $M = \langle Q, \Sigma, \delta, q_I, F \rangle$ accepting A where there is one state in Q for each equivalence class of $\equiv_A$

Myhill-Nerode theorem (contd.)

(contd.)

- Let $A_1, \ldots, A_r$ be the equivalence classes of $\equiv_A$

Myhill-Nerode theorem (contd.)

(contd.)

- Let $A_1, \ldots, A_r$ be the equivalence classes of $\equiv_A$
- Define $Q = \{q_1, \ldots, q_r\}$

Myhill-Nerode theorem (contd.)

(contd.)

- Let $A_1, \ldots, A_r$ be the equivalence classes of $\equiv_A$
- Define $Q = \{q_1, \ldots, q_r\}$
- Let $q_I \in Q$ be the q_i such that $\epsilon \in A_I$

Myhill-Nerode theorem (contd.)

(contd.)

- Let $A_1, \ldots, A_r$ be the equivalence classes of $\equiv_A$
- Define $Q = \{q_1, \ldots, q_r\}$
- Let $q_I \in Q$ be the q_i such that $\epsilon \in A_I$
- Note that for any A_j and any $a \in \Sigma$, for every $x, y \in A_j$, xa and ya will both be contained in the same equivalence class of $\equiv_A$

Myhill-Nerode theorem (contd.)

(contd.)

- Let $A_1, \ldots, A_r$ be the equivalence classes of $\equiv_A$
- Define $Q = \{q_1, \ldots, q_r\}$
- Let $q_I \in Q$ be the q_i such that $\epsilon \in A_I$
- Note that for any A_j and any $a \in \Sigma$, for every $x, y \in A_j$, xa and ya will both be contained in the same equivalence class of $\equiv_A$
- $\delta(q_j, a) = q_k$ such that for some $x \in A_j$, $xa \in A_k$

□

Myhill-Nerode theorem (contd.)

(contd.)

- Let $A_1, \ldots, A_r$ be the equivalence classes of $\equiv_A$
- Define $Q = \{q_1, \ldots, q_r\}$
- Let $q_I \in Q$ be the q_i such that $\epsilon \in A_I$
- Note that for any A_j and any $a \in \Sigma$, for every $x, y \in A_j$, xa and ya will both be contained in the same equivalence class of $\equiv_A$
- $\delta(q_j, a) = q_k$ such that for some $x \in A_j$, $xa \in A_k$
- Note that either $A_j \subseteq A$ or $A_j \cap A = \varnothing$, therefore, let $F = \{q_j | A_j \subseteq A\}$

$\square$

Myhill-Nerode theorem (contd.)

(contd.)

- Let $A_1, \ldots, A_r$ be the equivalence classes of $\equiv_A$
- Define $Q = \{q_1, \ldots, q_r\}$
- Let $q_I \in Q$ be the q_i such that $\epsilon \in A_I$
- Note that for any A_j and any $a \in \Sigma$, for every $x, y \in A_j$, xa and ya will both be contained in the same equivalence class of $\equiv_A$
- $\delta(q_j, a) = q_k$ such that for some $x \in A_j$, $xa \in A_k$
- Note that either $A_j \subseteq A$ or $A_j \cap A = \varnothing$, therefore, let $F = \{q_j | A_j \subseteq A\}$
- Through induction, it is evident that $\Sigma^*(q_I, x) = q_j \Leftrightarrow x \in A_j$

□

Myhill-Nerode theorem (contd.)

(contd.)

- Let $A_1, \ldots, A_r$ be the equivalence classes of $\equiv_A$
- Define $Q = \{q_1, \ldots, q_r\}$
- Let $q_I \in Q$ be the q_i such that $\epsilon \in A_I$
- Note that for any A_j and any $a \in \Sigma$, for every $x, y \in A_j$, xa and ya will both be contained in the same equivalence class of $\equiv_A$
- $\delta(q_j, a) = q_k$ such that for some $x \in A_j$, $xa \in A_k$
- Note that either $A_j \subseteq A$ or $A_j \cap A = \varnothing$, therefore, let $F = \{q_j | A_j \subseteq A\}$
- Through induction, it is evident that $\Sigma^*(q_I, x) = q_j \Leftrightarrow x \in A_j$
- This, together with the choice of F ensures that $\mathbf{L}(M) = A$ □

$\{0^n1^n|n \geq 0\}$ is not regular

$A = \{0^n1^n|n \geq 0\}$ is not regular

- Consider the sequence of strings $x_1, x_2, \ldots$ where $x_i = 0^i$ for $i \geq 1$
- We now see that no two of these are equivalent to each other with respect to $\equiv_A$
- Consider $x_i = 0^i$ and $x_j = 0^j$ for $i \neq j$
- Let $z = 1^i$ and notice that $x_i z = 0^i 1^i \in A$ but $x_j z = 0^j 1^i \notin A$
- Thus, no two of these strings are equivalent to each other and thus A cannot be regular

Summary

- The following are equally powerful as generators or recognisers
  - Deterministic finite automata
  - Non-deterministic finite automata
  - Regular expressions
  - Regular languages
  - Linear grammars
- Not all languages are regular
- Proving that a language is not regular
  - Pumping lemma
  - Myhill-Nerode theorem