Regular Languages and Finite Automata

Sets

- A set is a group of (possibly infinite) objects; its objects are called elements or members.
- The set without any element is called the empty set (written $\emptyset$).
- Let A, B be sets.
  - $A \cup B$ denotes the union of A and B.
  - $A \cap B$ denotes the intersection of A and B.
  - $\overline{A}$ denotes the complement of A (with respect to some universe U), i.e., $\overline{A} = \{x : x \in U, x \notin A\}$.A $\subseteq B$ denotes that A is a subset of B.
  - $A \subsetneq B$ denotes that A is a proper subset of B.
- The power set of a set A (written 2^A) is the set consisting of all subsets of A. E.g. $2^{\{0,1\}} = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

Sequences and Tuples

- A <u>sequence</u> is a (possibly infinite) list of ordered objects (e.g., 010101). In this course, we only deal with strings of finite length, unless stated otherwise.
- A finite sequence of k elements is also called <u>k-tuple</u> (e.g. (a, b, c, d) is a 4-tuple); a 2-tuple is also called a <u>pair</u>.
- The <u>Cartesian product</u> of sets A and B (written $A \times B$) is defined by

$$A \times B = \{(a, b) : a \in A \text{ and } b \in B\}.$$

  E.g. $\{0, 1\} \times \{a, b\} = \{(0, a), (0, b), (1, a), (1, b)\}$
- We can take Cartesian products of k sets $A_1, A_2, \ldots, A_k$

$$A_1 \times A_2 \times \cdots \times A_k = \{(a_1, a_2, \ldots, a_k) : a_i \in A_i \text{ for every } 1 \leq i \leq k\}.$$

- Define

$$A^k = \overbrace{A \times A \times \cdots \times A}^{k}.$$

Functions and Relations

- A function $f : D \to R$ maps an element in the domain D to an element in the range R. Write $f(a) = b$ if f maps $a \in D$ to $b \in R$.
- When $f : A_1 \times A_2 \times \cdots \times A_k \to B$, we say f is a k-ary function and k is the arity of f ($k = 1$: unary function; $k = 2$, binary function).
- A predicate or property is a function whose range is $\{0, 1\}$. E.g. in C language, "$x == y$" is a predicate, which returns 1 if x and y are equal; 0 otherwise.
- A property with domain $\overbrace{A \times A \times \cdots \times A}^{k}$ is a k-ary relation on A.
  - When $k = 2$, it is a binary relation.
- A binary relation R is an equivalence relation if
  - R is reflexive (for every x, xRx);
  - R is symmetric (for every x and y, xRy implies yRx; and
  - R is transitive (for every x, y, and z, xRy and yRz implies xRz.
- R is antisymmetric if $\forall$ x and y, xRy and yRx imply $x = y$. (Question: "Antisymmetric" = "not symmetric"?)
- Do you recall what a partial order relation is?

More about Sets

A set A is countably infinite if there is a bijection $f : \mathbb{N} \to A$.

Theorem 1

Let $\mathbb{B}$ *be* $\{0, 1\}$. *Then* $A = \mathbb{B} \times \mathbb{B} \times \cdots \times \mathbb{B} \times \cdots$ *is uncountable.*

Proof.

$$
\begin{array}{ll}
s_1 & = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \ldots \\
s_2 & = 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \ldots \\
s_3 & = 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \ldots \\
s_4 & = 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \ldots \\
s_5 & = 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 \ldots \\
s_6 & = 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \ldots \\
s_7 & = 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0 \ldots \\
s_8 & = 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1 \ldots \\
s_9 & = 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \ldots \\
s_{10} & = 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1 \ldots \\
s_{11} & = 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0 \ldots \\
\vdots & \quad \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots
\end{array}
$$

$$
s = 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \ldots
$$

Induction Proof

- **Induction Principle**:
  $P(0) \wedge (\forall k, P(k) \Rightarrow P(k+1)) \Rightarrow (\forall n \in \mathbb{N}, P(n))$.
  - Why do we call it a "principle"? Why not call it a "Theorem"? (Check out the *axiom of induction* in Peano Arithmetic.)
- **Well-founded Relation**:
  A binary R is called well-founded on a class X if every **non-empty subset** $S \subseteq X$ has a **minimal element** with respect to R. (E.g., $\mathbb{N}$ is well-founded; $\mathbb{Z}$ is not well-founded.)

Induction Principle $\Leftrightarrow$ $(\mathbb{N}, <)$ **is well-founded.**

To prove property $P(n)$ holds for all $n \in \mathbb{N}$,

- **(Induction Basis)**: Prove $P(0)$;
- **(Induction Step)**: Prove that if $P(k)$ holds, then $P(k+1)$ also holds.

Strings and Languages

- An <u>alphabet</u> is a nonempty finite set. E.g. $\Sigma = \{0, 1\}$.
- Members of an alphabet are called <u>symbols</u>. E.g. 0, 1 in Σ.
- A <u>string</u> over an alphabet is a finite sequence of symbols from the alphabet. E.g. 000111.
- If w is a string over an alphabet Σ, the <u>length</u> of w (written as $|w|$) is the number of symbols in w. E.g. $|000111| = 6$.
- The string of length zero is the <u>empty string</u> (written as ϵ).
- Let $x = x_1x_2 \cdots x_n$ and $y = y_1y_2 \cdots y_m$ be strings of length n and m respectively. The <u>concatenation</u> of x and y (written as $x \cdot y$ or xy) is the string $x_1x_2 \cdots x_ny_1y_2 \cdots y_m$ of length $n + m$.
- For any string x, $x^k = \overbrace{xx \cdots x}^{k}$.
- A <u>language</u> is a set of strings. E.g. $\{01, 0011, 000111, ...\}$.

Schematic of Finite Automata

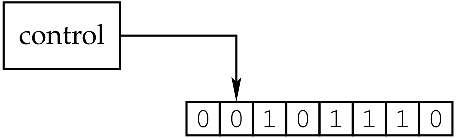


Figure: Schematic of Finite Automata

- A finite automaton has a finite set of control states.
- A finite automaton reads input symbols from left to right.
- A finite automaton accepts or rejects an input after reading the input.

Finite Automaton M_1

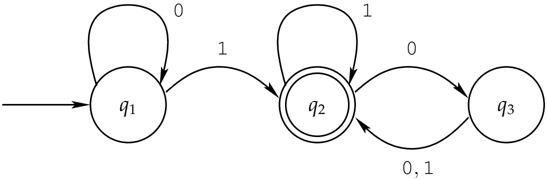
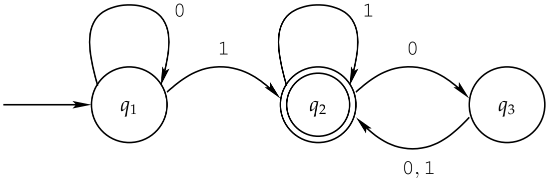


Figure: A Finite Automaton M_1

The above figure shows the state diagram of a finite automaton M_1. M_1 has

- 3 states: q_1, q_2, q_3;
- a start state: q_1;
- a accept state: q_2;
- 6 transitions: $q_1 \xrightarrow{0} q_1$, $q_1 \xrightarrow{1} q_2$, $q_2 \xrightarrow{1} q_2$, $q_2 \xrightarrow{0} q_3$, $q_3 \xrightarrow{0} q_2$, and $q_3 \xrightarrow{1} q_2$.

Accepted and Rejected String



- Consider an input string `1100`.
- M_1 processes the string from the start state q_1.
- It takes the transition labeled by the current symbol and moves to the next state.
- At the end of the string, there are two cases:
  - If M_1 is at an accept state, M_1 outputs accept;
  - Otherwise, M_1 outputs reject.
- Strings accepted by M_1: `1`, `01`, `11`, `1100`, `1101`, ...
- Strings rejected by M_1: `0`, `00`, `10`, `010`, `1010`, ...

Finite Automaton – Formal Definition

- A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a finite set of states;
  - Σ is a finite set called alphabet;
  - $\delta : Q \times \Sigma \to Q$ is the transition function;
  - $q_0 \in Q$ is the start state; and
  - $F \subseteq Q$ is the set of accept states.
- Accept states are also called final states.
- The set of all strings that M accepts is called the language of machine M (written $L(M)$).
  - Recall a language is a set of strings.
- We also say M recognizes (or accepts) $L(M)$.

For convenience, we also define the *extended transition function* $\delta^* : Q \times \Sigma^* \to Q$ as follows:

- $\delta^*(p, \epsilon) = p$,
- $\delta^*(p, ua) = \delta(\delta^*(p, u), a)$, where $a \in \Sigma, u \in \Sigma^*$

Intuitively, $\delta^*(p, w)$ is the state reached from state p following the path from p reading w.

M_1 – Formal Definition

- A finite automaton $M_1 = (Q, \Sigma, \delta, q_1, F)$ consists of
  - $Q = \{q_1, q_2, q_3\}$;
  - $\Sigma = \{\texttt{0}, \texttt{1}\}$;
  - $\delta : Q \times \Sigma \to Q$ is

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

  - q_1 is the start state; and
  - $F = \{q_2\}$.
- Moreover, we have

$$L(M_1) = \{w : \ w \text{ contains at least one } \texttt{1} \text{ and an even number of } \texttt{0}\text{'s follow the last } \texttt{1}\}$$

Finite Automaton M_2

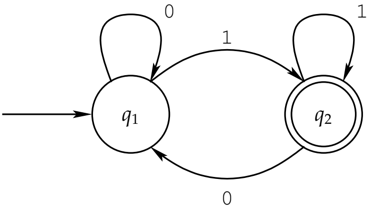


Figure: Finite Automaton M_2

- The above figure shows $M_2 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$ where δ is

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

- What is $L(M_2)$?
  - $L(M_2) = \{w : w \text{ ends in a } 1\}$.

Finite Automaton M_2

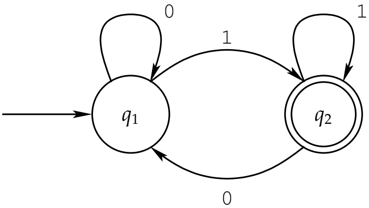


Figure: Finite Automaton M_2

- The above figure shows $M_2 = (\{q_1, q_2\}, \{\mathtt{0}, \mathtt{1}\}, \delta, q_1, \{q_2\})$ where δ is

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

- What is $L(M_2)$?
  - $L(M_2) = \{w : w \text{ ends in a } \mathtt{1}\}$.

Finite Automaton M_3

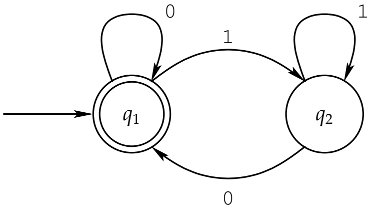


Figure: Finite Automaton M_3

- The above figure shows $M_3 = (\{q_1, q_2\}, \{\texttt{0}, \texttt{1}\}, \delta, q_1, \{q_1\})$ where δ is

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

- What is $L(M_3)$?
  - $L(M_3) = \{w : w$ is the empty string ϵ or ends in a $\texttt{0}\}$.

Finite Automaton M_3

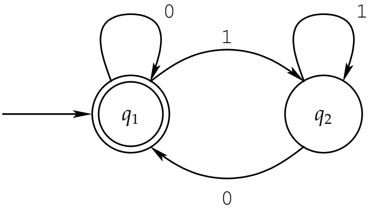


Figure: Finite Automaton M_3

- The above figure shows $M_3 = (\{q_1, q_2\}, \{\mathtt{0}, \mathtt{1}\}, \delta, q_1, \{q_1\})$ where δ is

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

- What is $L(M_3)$?
  - $L(M_3) = \{w : w$ is the empty string ϵ or ends in a $\mathtt{0}\}$.

Computation – Formal Definition

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1 w_2 \cdots w_n$ a string where $w_i \in \Sigma$ for every $i = 1, \ldots, n$.
- We say M accepts w if there is a sequence of states $r_0, r_1, \ldots, r_n$ such that

$$r_0 \stackrel{w_1}{\to} r_1 \stackrel{w_2}{\to} r_2 \cdots r_{n-1} \stackrel{w_n}{\to} r_n,$$

  - $r_0 = q_0$;
  - $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$; and
  - $r_n \in F$,
- In the above, $\delta^*(r_0, w_1 \cdots w_n) = \delta(\delta^*(r_0, w_1 \cdots w_{n-1}), w_n) = \delta(r_{n-1}, w_n) = r_n$.
- M recognizes language A if $A = \{w : M \text{ accepts } w\}$, or equivalently, $A = \{w : \delta^*(q_0, w) \in F\}$.

Definition 2

A language is called a regular language if some finite automaton recognizes it.

Regular Operations

Definition 3

Let A and B be languages. We define the following operations:

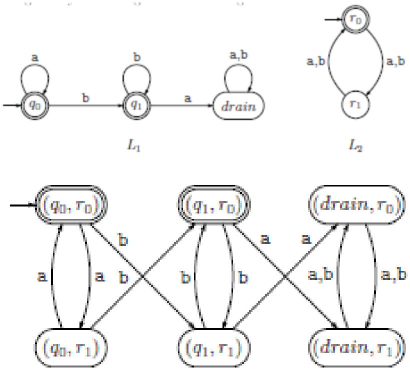
- Union: $A \cup B = \{x : x \in A \text{ or } x \in B\}$.
- Concatenation: $A \cdot B = \{xy : x \in A \text{ and } y \in B\}$.
- Star: $A^* = \{x_1 x_2 \cdots x_k : k \geq 0 \text{ and every } x_i \in A\}$.

- Note that $\epsilon \in A^*$ for every language A. ($\epsilon \in \emptyset^*$.)
- Another way of defining A^*:
  - $A^0 = \{\epsilon\}$;
  - $A^{k+1} = A \cdot A^k, k \geq 0$.
  - $A^* = \bigcup_{k \geq 0} A^k$
- What is $\emptyset^*$?

Given two automata $A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, the product A of A_1 and A_2 (written as $A_1 \times A_2$), is $(Q, \Sigma, \delta, q, F)$, where

- $Q = Q_1 \times Q_2$; $q = (q_1, q_2)$,
- $\delta((p_1, p_2), a) = (p'_1, p'_2)$ if $\delta_1(p_1, a) = p'_1$ and $\delta_2(p_2, a) = p'_2$
- F is defined depending on the goal of the construction.

Intuitively, A can be thought of as running A_1 and A_2 in parallel.



Closure Property – Union

Theorem 4

The class of regular languages is closed under the union operation. That is, $A_1 \cup A_2$ is regular if A_1 and A_2 are.

Proof.

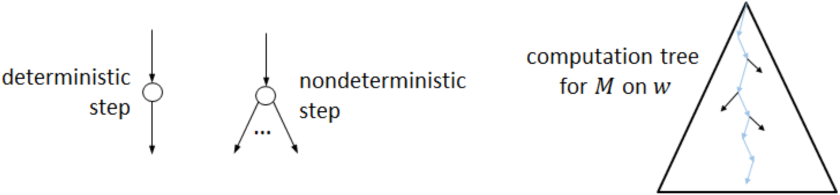
Let $M_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ recognize A_i for $i = 1, 2$. Construct $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = Q_1 \times Q_2 = \{(r_1, r_2) : r_1 \in Q_1, r_2 \in Q_2\}$;
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$;
- $q_0 = (q_1, q_2)$;
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2) = \{(r_1, r_2) : r_1 \in F_1 \text{ or } r_2 \in F_2\}$. $\square$

- Why is $L(M) = A_1 \cup A_2$?
- Can you use product construction to show that regular languages are closed under intersection?

Nondeterminism

- When a machine is at a given state and reads an input symbol, there is precisely one choice of its next state.
- This is call deterministic computation.
- In nondeterministic machines, multiple choices may exist for the next state.
- A deterministic finite automaton is abbreviated as DFA; a nondeterministic finite automaton is abbreviated as NFA.
- A DFA is also an NFA.
- Since NFA allow more general computation, they can be much smaller than DFA.



NFA N_4

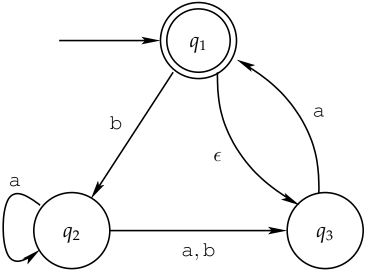


Figure: NFA N_4

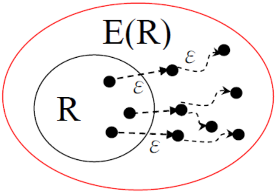
- On input string baa, N_4 has several possible computations:
  - $q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_2 \xrightarrow{a} q_2$;
  - $q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_2 \xrightarrow{a} q_3$; or
  - $q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \xrightarrow{a} q_1$.

Nondeterministic Finite Automaton – Formal Definition

- $\mathcal{P}(Q)$ (also written as 2^Q) $= \{R : R \subseteq Q\}$ denotes the power set of Q.
- For any alphabet Σ, define Σ_ϵ to be $\Sigma \cup \{\epsilon\}$.
- A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a finite set of states;
  - Σ is a finite alphabet;
  - $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is the transition function;
  - $q_0 \in Q$ is the start state; and
  - $F \subseteq Q$ is the accept states.
- In some textbooks, δ is defined as a relation $\delta \subseteq Q \times \Sigma \times Q$. E.g., $\delta(q, a) = \{q_1, q_2\}$ can the thought of as $(q, a, q_1), (q, a, q_2) \in \delta$.

ϵ-closure of NFA

Given a set R, we define the *ϵ-closure*(R) ($E(R)$) as follows:

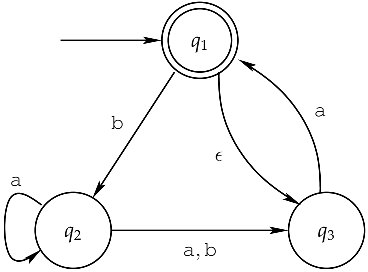


- Intuitively, ϵ-closure(p) is the set of states reachable from p by an ϵ-path.
- How to compute ϵ-closure(p)?

The *extended transition function* $\delta^* : Q \times \Sigma^* \to 2^Q$ is as follows:

- $\delta^*(p, \epsilon) = \epsilon\text{-closure}(\{p\})$
- $\delta^*(p, ua) = \epsilon\text{-closure}(\bigcup_{s \in \delta^*(p,u)} \delta(s, a))$

NFA N_4 – Formal Definition



- $N_4 = (Q, \Sigma, \delta, q_1, \{q_1\})$ is a nondeterministic finite automaton where
  - $Q = \{q_1, q_2, q_3\}$;
  - Its transition function δ is

	ϵ	a	b
q_1	$\{q_3\}$	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\{q_2, q_3\}$	$\{q_3\}$
q_3	$\emptyset$	$\{q_1\}$	$\emptyset$

Nondeterministic Computation – Formal Definition

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over Σ. We say N accepts w if w can be rewritten as $w = y_1 y_2 \cdots y_m$ with $y_i \in \Sigma_\epsilon$ and there is a sequence of states $r_0, r_1, \ldots, r_m$ such that

$$r_0 \xrightarrow{y_1} r_1 \xrightarrow{y_2} r_2 \cdots r_{m-1} \xrightarrow{y_m} r_m,$$

  - $r_0 = q_0$;
  - $r_{i+1} \in \delta(r_i, y_{i+1})$ for $i = 0, \ldots, m-1$; and
  - $r_m \in F$.
- Note that finitely many empty strings can be inserted in w.
- Also note that one sequence satisfying the conditions suffices to show the acceptance of an input string.
- M recognizes language A if $A = \{w : M \text{ accepts } w\}$, or equivalently, $A = \{w : \delta^*(q_0, w) \cap F \neq \emptyset\}$.

Equivalence of NFA's and DFA's via Subset Construction

Theorem 5

Every nondeterministic finite automaton has an equivalent deterministic finite automaton. That is, for every NFA N, there is a DFA M such that $L(M) = L(N)$*.*

Proof.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA. For $R \subseteq Q$, define $E(R) = \{q : q$ can be reached from R along 0 or more ϵ transitions $\}$. Construct a DFA $M = (Q', \Sigma, \delta', q'_0, F')$ where

- $Q' = \mathcal{P}(Q)$;
- $\delta'(R, a) = \{q \in Q : q \in E(\delta(r, a))$ for some $r \in R\}$;
- $q'_0 = E(\{q_0\})$;
- $F' = \{R \in Q' : R \cap F \neq \emptyset\}$. $\square$

Correctness Proof of Subset Construction

In NFA N, we write $r_0 \overset{w}{\Longrightarrow} r_m$ if $r_0 \overset{y_1}{\to} r_1 \overset{y_2}{\to} r_2 \cdots r_{m-1} \overset{y_m}{\to} r_m$, and $y_1 y_2 \cdots y_m = w$, where $y_i \in \Sigma_\epsilon$. I.e., there is a path from r_0 to r_m reading w. In what follows, we prove the following lemma by induction on the length of w that

Lemma 6

$$\overbrace{(\delta')^*(E(\{q_0\}), w)}^{\textit{DFA } M} = \overbrace{\{r \in Q \mid q_0 \overset{w}{\Longrightarrow} r\}}^{\textit{NFA } N}.$$

- **Induction Basis**: Consider the case $|w| = 0$, i.e., $w = \epsilon$. Clearly,
  $$(\delta')^*(E(\{q_0\}), \epsilon) = R \quad \Leftrightarrow \quad R = E(\{q_0\}) = \{r \in Q \mid q_0 \overset{\epsilon}{\Longrightarrow} r\}$$
  [Def. of $\Longrightarrow$ and $E(\{q_0\})$]

- **Induction Hypothesis**: Assume that the assertion holds for $0 \leq |w| \leq k$.

Correctness Proof of Subset Construction (Cont'd)

- **Induction Step**: Consider the case when $|w| = k+1$, i.e., $w = xa$, where $x \in \Sigma^*$, $|x| = k$, and $a \in \Sigma$.

$$
\begin{array}{lll}
q_0 \stackrel{xa}{\Longrightarrow} r & & \\
\Leftrightarrow & (\exists s,t \in Q) q_0 \stackrel{x}{\Longrightarrow} s \ \wedge \ s \stackrel{a}{\rightarrow} t \ \wedge \ t \stackrel{\epsilon}{\Longrightarrow} r & [\text{Def. of } \Longrightarrow] \\
\Leftrightarrow & (\exists s,t \in Q) q_0 \stackrel{x}{\Longrightarrow} s \ \wedge \ s \stackrel{a}{\rightarrow} t \ \wedge \ r \in E(\{t\}) & [\text{Def. of } E] \\
\Leftrightarrow & (\exists s,t \in Q) s \in (\delta')^*(E(\{q_0\}), x) \ \wedge \ t \in \delta(s,a) \ \wedge \ r \in E(\{t\}) & \\
& \qquad [\text{Ind. Hyp.; Defs. of } \delta \text{ and } \Longrightarrow] & \\
\Leftrightarrow & (\exists s \in Q) s \in (\delta')^*(E(\{q_0\}), x) \ \wedge \ r \in E(\delta(s,a)) & [\text{Def. of } E] \\
\Leftrightarrow & r \in \bigcup_{s \in S} E(\delta(s,a)) \text{ for } S = (\delta')^*(E(\{q_0\}), x) & [\text{Def. of } \bigcup] \\
\Leftrightarrow & r \in \delta'(S,a) \text{ for } S = (\delta')^*(E(\{q_0\}), x) & [\text{Def. of } \delta'] \\
\Leftrightarrow & r \in \delta'((\delta')^*(E(\{q_0\}), x), a)) & \\
\Leftrightarrow & r \in (\delta')^*(E(\{q_0\}), xa) & [\text{Def. of } (\delta')^*]
\end{array}
$$

Hence, $(\delta')^*(E(q_0), w) \ = \ \{r \in Q \mid q_0 \stackrel{w}{\Longrightarrow} r\}$.

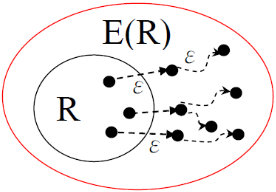
Correctness Proof of Subset Construction (Cont'd)

Now we are ready to show $L(M) = L(N)$.

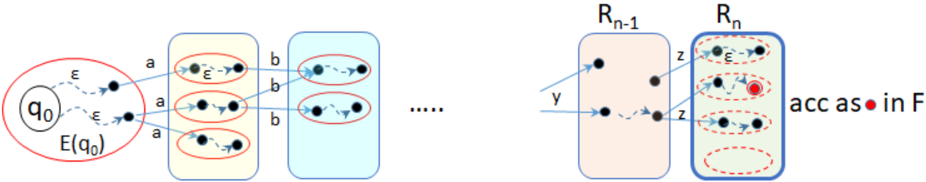
$$
\begin{array}{rcll}
w \in L(M) & \Leftrightarrow & (\delta')^*(E(\{q_0\}), w) \in F' & \\
& \Leftrightarrow & (\exists r \in F)\ \ r \in (\delta')^*(E(\{q_0\}), w) & \text{[Def. of } F'\text{]} \\
& \Leftrightarrow & (\exists r \in F)\ \ q_0 \stackrel{w}{\Longrightarrow} r & \text{[Lemma 6]} \\
& \Leftrightarrow & w \in L(N) & \text{[Def. of } L(N)\text{]}
\end{array}
$$

Equivalence of NFA's and DFA's

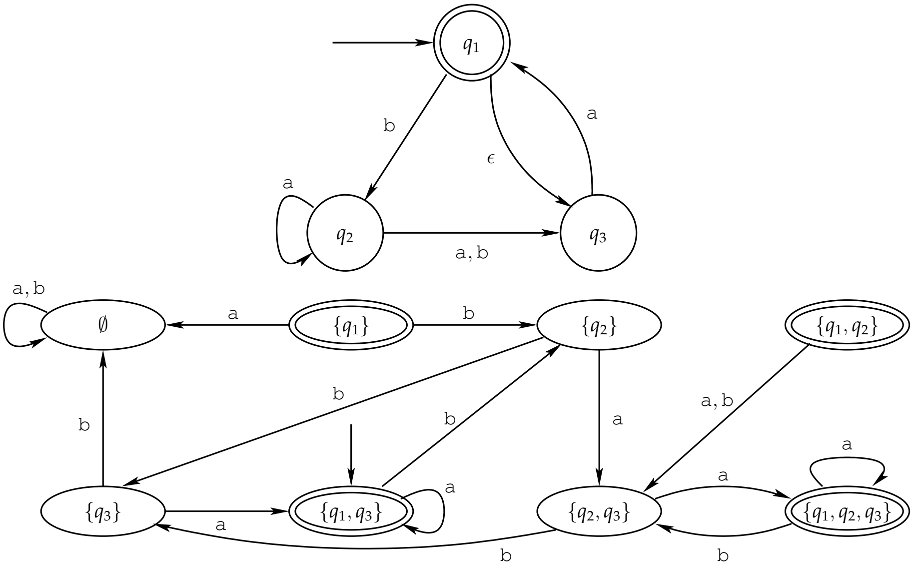
- ϵ-closure $E(R)$:



- Transition $\delta'(R, a) = \{q \mid q \in E(\delta(r, a)),\ \text{for some } r \in R\}$



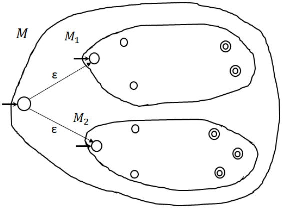
A DFA Equivalent to N_4



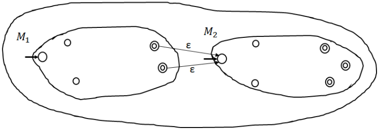
Closure Properties – Revisited

Closed under *Union*, *Concatenation* and *Star*. (Proof Idea):

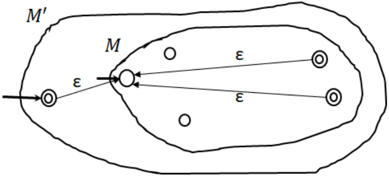
Union



Concatenation



Star



(Figures from M. Sipser's lecture notes)

Closure Properties – Revisited

Theorem 7

The class of regular languages is closed under the union operation.

Proof.

Let $N_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ recognize A_i for $i = 1, 2$. Construct $N = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{q_0\} \cup Q_1 \cup Q_2$;
- $F = F_1 \cup F_2$; and
- $\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon \end{cases}$ $\square$

- Why is $L(N) = L(N_1) \cup L(N_2)$?

Closure Properties – Revisited

Theorem 8

The class of regular languages is closed under the concatenation operation.

Proof.

Let $N_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ recognize A_i for $i = 1, 2$. Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ where

- $Q = Q_1 \cup Q_2$; and
- $$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \epsilon \\ \delta_2(q, a) & q \in Q_2 \end{cases}$$

$\square$

- Why is $L(N) = L(N_1) \cdot L(N_2)$?

Closure Properties – Revisited

Theorem 9

The class of regular languages is closed under the star operation.

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{q_0\} \cup Q_1$;
- $F = \{q_0\} \cup F_1$; and
- $\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a = \epsilon \\ \{q_1\} & q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon \end{cases}$

$\square$

- Why is $L(N) = [L(N_1)]^*$?

Closure Properties – Revisited

Theorem 10

The class of regular languages is closed under complementation.

Proof.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing A. Consider $\overline{M} = (Q, \Sigma, \delta, q_0, Q \setminus F)$. We have $w \in L(M)$ if and only if $w \notin L(\overline{M})$. That is, $L(\overline{M}) = \overline{A}$ as required. □

Theorem 11

The class of regular languages is closed under intersection.

Proof.

Recall that $R \cap S = \overline{\overline{R} \cup \overline{S}}$. □

Regular Expressions (Syntax)

Definition 12

R is a regular expression if R is

- a for some $a \in \Sigma$;
- ϵ;
- $\emptyset$;
- $(R_1 + R_2)$ where R_1 and R_2 are regular expressions;
- $(R_1 \cdot R_2)$ where R_1 and R_2 are regular expressions; or
- (R_1^*) where R_1 is a regular expression.

- We write R^+ for $R \cdot R^*$. Hence $R^* = R^+ + \epsilon$.
- Moreover, write R^k for $\overbrace{R \cdot R \cdot \cdots \cdot R}^{k}$.
  - Define $R^0 = \epsilon$. We have $R^* = R^0 + R^1 + \cdots + R^n + \cdots$.
- $L(R)$ denotes the language described by the regular expression R.
- Note that $\emptyset \neq \{\epsilon\}$. $+$ is also written as "$\cup$" is many textbooks

Regular Expressions (Semantics)

Definition 13

The language associated with a regular expression R, written as $L(R)$, is defined recursively as

- $L(a) = \{a\}, a \in \Sigma$;
- $L(\epsilon) = \{\epsilon\}$;
- $L(\emptyset) = \emptyset$;
- $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
- $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2)$
- $L(R_1^*) = (L(R_1))^*$

Examples of Regular Expressions

- For convenience, we write RS for $R \cdot S$.
- We may also write the regular expression R to denote its language $L(R)$.
- $L(\mathtt{0}^*\mathtt{1}\mathtt{0}^*) = \{w : w \text{ contains a single } \mathtt{1}\}$.
- $L(\Sigma^*\mathtt{1}\Sigma^*) = \{w : w \text{ has at least one } \mathtt{1}\}$.
- $L((\Sigma\Sigma)^*) = \{w : w \text{ is a string of even length }\}$.
- $(\mathtt{0} + \epsilon)(1 + \epsilon) = \{\epsilon, \mathtt{0}, \mathtt{1}, \mathtt{01}\}$.
- $\mathtt{1}^*\emptyset = \emptyset$.
- $\emptyset^* = \{\epsilon\}$.
- For any regular expression R, we have $R + \emptyset = R$ and $R \cdot \epsilon = R$.

Regular Expressions and Finite Automata

Lemma 14

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

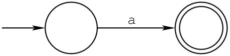
Proof.

We prove by induction on the regular expression R.

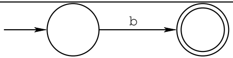
- $R = a$ for some $a \in \Sigma$. Consider the NFA $N_a = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$ where $\delta(r, y) = \begin{cases} \{q_2\} & r = q_1 \text{ and } y = a \\ \emptyset & \text{otherwise} \end{cases}$
- $R = \epsilon$. Consider the NFA $N_\epsilon = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$ where $\delta(r, y) = \emptyset$ for any r and y.
- $R = \emptyset$. Consider the NFA $N_\emptyset = (\{q_1\}, \Sigma, \delta, q_1, \emptyset)$ where $\delta(r, y) = \emptyset$ for any r and y.
- $R = R_1 + R_2$, $R = R_1 \cdot R_2$, or $R = R_1^*$. By inductive hypothesis and the closure properties of finite automata. □

Regular Expressions and Finite Automata

$\mathtt{a}$

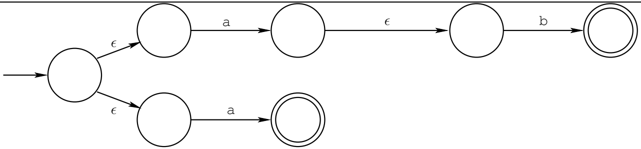


$\mathtt{b}$

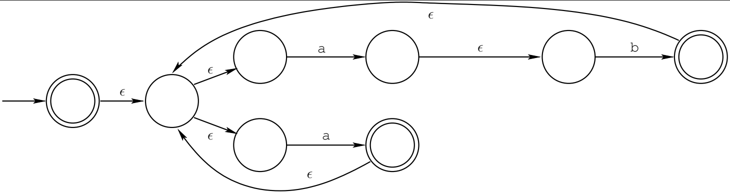


$\mathtt{ab}$

$\mathtt{ab} + \mathtt{a}$



$(\mathtt{ab} + \mathtt{a})^*$



Lemma 15

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

For the proof, we introduce a generalization of finite automata.

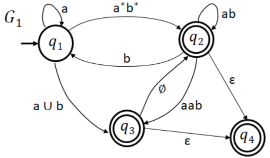
Generalized Nondeterministic Finite Automata

Definition 16

A generalized nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, q_{\text{start}}, q_{\text{accept}})$ where

- Q is the finite set of states; Σ is the input alphabet;
- $\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \to \mathcal{R}$ is the transition function, where $\mathcal{R}$ denotes the set of regular expressions;
- q_{start} is the start state; and q_{accept} is the accept state.

A GNFA accepts a string $w \in \Sigma^*$ if $w = w_1 w_2 \cdots w_k$ where $w_i \in \Sigma^*$ and there is a sequence of states $r_0, r_1, \ldots, r_k$ such that $r_0 = q_{\text{start}}$; $r_k = q_{\text{accept}}$; and for every



i, $w_i \in L(R_i)$ where $R_i = \delta(q_{i-1}, q_i)$.

(Fig. from M. Sipser)

Finite Automata to Regular Expressions - State Elimination

Proof of Lemma.

Let M be the DFA for the regular language. Construct an equivalent GNFA G by adding $q_{\text{start}}, q_{\text{accept}}$ and necessary ϵ-transitions.
CONVERT (G):

1. Let k be the number of states of G.
2. If $k = 2$, then return the regular expression R labeling the transition from q_{start} to q_{accept}.
3. If $k > 2$, select $q_{\text{rip}} \in Q \setminus \{q_{\text{start}}, q_{\text{accept}}\}$. Construct $G' = (Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$ where
   - $Q' = Q \setminus \{q_{\text{rip}}\}$;
   - for any $q_i \in Q' \setminus \{q_{\text{accept}}\}$ and $q_j \in Q' \setminus \{q_{\text{start}}\}$, define $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup R_4$ where $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.
4. return CONVERT (G').

$\square$

Finite Automata to Regular Expressions - State Elimination

Lemma 17

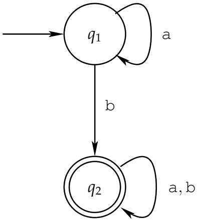
For any GNFA G, CONVERT (G) is equivalent to G.

Proof.

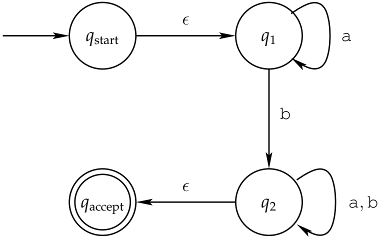
We prove by induction on the number k of states of G.

- $k = 2$. Trivial.
- Assume the lemma holds for $k-1$ states. We first show G' is equivalent to G. Suppose G accepts an input w. Let $q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}$ be an accepting computation of G. We have $q_{\text{start}} \xrightarrow{w_1} q_1 \cdots q_{i-1} \xrightarrow{w_i} q_i \xrightarrow{w_{i+1}} q_{\text{rip}} \cdots q_{\text{rip}} \xrightarrow{w_{j-1}} q_{\text{rip}} \xrightarrow{w_j} q_j \cdots q_{\text{accept}}$. Hence $q_{\text{start}} \xrightarrow{w_1} q_1 \cdots q_{i-1} \xrightarrow{w_i} q_i \xrightarrow{w_{i+1} \cdots w_j} q_j \cdots q_{\text{accept}}$ is a computation of G'. Conversely, any string accepted by G' is also accepted by G since the transition between q_i and q_j in G' describes the strings taking q_i to q_j in G. Hence G' is equivalent to G. By

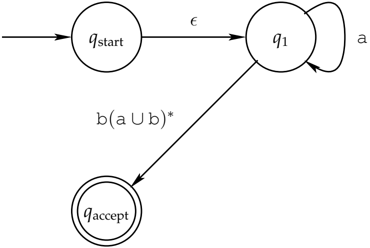
Finite Automata to Regular Expressions - State Elimination

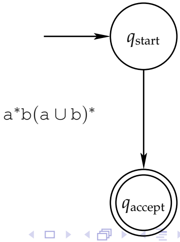


(a) DFA M

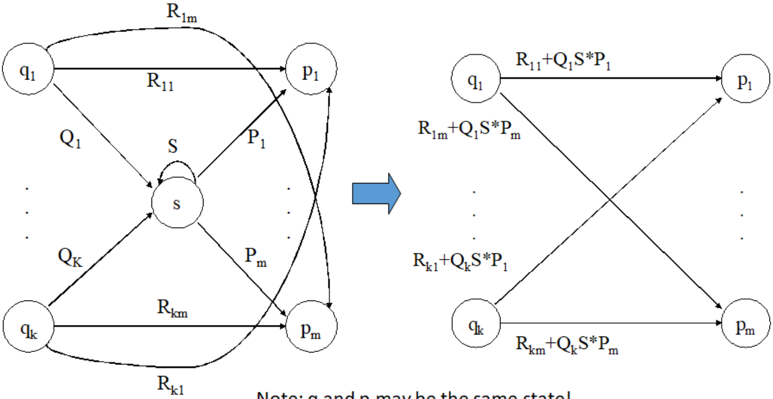


(b) GNFA G





In general ...



Note: q and p may be the same state!

Theorem 18

A language is regular if and only if some regular expression describes it.

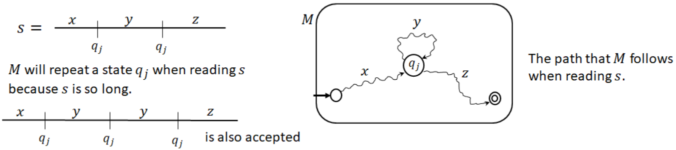
A tool for proving non-regularity.

Lemma 19

If A is a regular language, then there is a number p such that for any $s \in A$ of length at least p, there is a partition $s = xyz$ with

1. *for each $i \geq 0$, $xy^iz \in A$;*
2. $|y| > 0$; *and*
3. $|xy| \leq p$.

Proof Idea:



(Fig. from M. Sipser)

Pumping Lemma (Proof)

Proof.

Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A and $p = |Q|$.
Consider any string $s = \sigma_1\sigma_2 \cdots \sigma_{m-1}$ of length $m - 1 \geq p$. Let $q_1, \ldots, q_m$ be the sequence of states such that $q_{i+1} = \delta(q_i, \sigma_i)$ for $1 \leq i \leq m - 1$.
Since $m \geq p + 1 = |Q| + 1$, there are $1 \leq s < t \leq p + 1$ such that $q_s = q_t$ (why?). Let $x = \sigma_1 \cdots \sigma_{s-1}$, $y = \sigma_s \cdots \sigma_{t-1}$, and $z = \sigma_t \cdots \sigma_{m-1}$.
Note that $q_1 \xrightarrow{x} q_s$, $q_s \xrightarrow{y} q_t$, and $q_t \xrightarrow{z} q_m \in F$. Thus M accepts xy^iz for $i \geq 0$. Since $t \neq s$, $|y| > 0$. Finally, $|xy| \leq p$ for $t \leq p + 1$. $\square$

How to Use Pumping Lemma?

Recall that Pumping Lemma can be expressed as the following logical formula:

$$A \text{ is regular } \Rightarrow \exists p \in \mathbb{N}, \forall_{s,(s\in A)\wedge(|s|\geq p)} \exists_{x,y,z,\ s=xyz}((1) \wedge (2) \wedge (3))$$

which is equivalent to

$$\neg(\exists p \in \mathbb{N}, \forall_{s,(s\in L)\wedge(|s|\geq p)} \exists_{x,y,z,\ s=xyz}((1) \wedge (2) \wedge (3))) \Rightarrow A \text{ is NOT regular}$$

Note that the left-hand side is

$$\forall p \in \mathbb{N}, \exists_{s,(s\in A)\wedge(|s|\geq p)}, \forall_{x,y,z,\ s=xyz}(\neg(1) \vee \neg(2) \vee \neg(3))$$

How to Use Pumping Lemma?

In view of "

$$\forall p \in \mathbb{N}, \exists_{s,(s\in A)\wedge(|s|\geq p)}, \forall_{x,y,z,\ s=xyz}(\neg(1) \vee \neg(2) \vee \neg(3)), \text{"} \Rightarrow \text{NOT regular}$$

proving A is not regular resembles a *two-player game* between **YOU** and your **adversary** (ADV), such that your goal is to prove non-regularity, while ADV wants to spoil it.

1. ADV picks an arbitrary $p \in \mathbb{N}$
2. YOU pick an s, $s \in A$, $|s| \geq p$
3. ADV picks arbitrary x, y, z with $s = xyz$
4. YOU show $\neg(1) \vee \neg(2) \vee \neg(3)$
   - $\neg(2)$ and $\neg(3)$ are trivial to check
   - YOU establish $(2) \wedge (3) \Rightarrow \neg(1)$, i.e., $(|y| > 0) \wedge (|xy| \leq p) \ \Rightarrow\ \exists i \geq 0, xy^iz \notin A$.

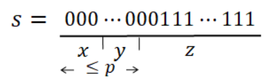
Applications of Pumping Lemma

Example 20

$B = \{\texttt{0}^n\texttt{1}^n : n \geq 0\}$ is not a regular language.

Proof.

Suppose B is regular. Let p be the pumping length given by the pumping lemma. Choose $s = \texttt{0}^p\texttt{1}^p$. Then $s \in B$ and $|s| \geq p$, there is a partition $s = xyz$ such that $xy^iz \in B$ for $i \geq 0$.



- $y \in \texttt{0}^+$ or $y \in \texttt{1}^+$. $xz \notin B$. A contradiction.
- $y \in \texttt{0}^+\texttt{1}^+$. $xyyz \notin B$. A contradiction. □

Corollary 21

$C = \{w : w$ *has an equal number of* $\texttt{0}$*'s and* $\texttt{1}$*'s*$\}$ *is not a regular language.*

Applications of Pumping Lemma

Example 22

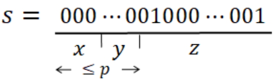
Is $B' = \{\texttt{0}^n\texttt{1}^n : 0 \leq n \leq 10^{100}\}$ regular?
(What if ADV picks $p = 2 \times 10^{101}$?)

Example 23

$F = \{ww : w \in \{\texttt{0},\texttt{1}\}^*\}$ is not a regular language.

Proof.

Suppose F is a regular language and p the pumping length. Choose $s = \texttt{0}^p\texttt{1}\texttt{0}^p\texttt{1}$. By the pumping lemma, there is a partition $s = xyz$ such that $|xy| \leq p$ and $xy^iz \in F$ for $i \geq 0$. Since $|xy| \leq p$, $y \in \texttt{0}^+$. But then $xz \notin F$. A contradiction. $\square$



Applications of Pumping Lemma

Example 24

$D = \{1^{n^2} : n \geq 0\}$ is not a regular language.

Proof.

Suppose D is a regular language and p the pumping length. Choose $s = 1^{p^2}$. By the pumping lemma, there is a partition $s = xyz$ such that $|y| > 0$, $|xy| \leq p$, and $xy^iz \in D$ for $i \geq 0$.
Consider the strings xyz and xy^2z. We have $|xyz| = p^2$ and $|xy^2z| = p^2 + |y| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$. Since $|y| > 0$, we have $p^2 = |xyz| < |xy^2z| < (p+1)^2$. Thus $xy^2z \notin D$. A contradiction. $\square$

Theorem 25

For $\{1^{f(n)} : n \geq 0\}$ to be regular, $f(n)$ must be a linear function of the form $f(n) = an + b$.

Applications of Pumping Lemma

Example 26

$E = \{0^i 1^j : i > j\}$ is not a regular language.

Proof.

Suppose E is a regular language and p the pumping length. Choose $s = 0^{p+1}1^p$. By the pumping lemma, there is a partition $s = xyz$ such that $|y| > 0$, $|xy| \leq p$, and $xy^iz \in E$ for $i \geq 0$. Since $|xy| \leq p$, $y \in 0^+$. But then $xz \notin E$ for $|y| > 0$. A contradiction. $\square$

Pumping Lemma is not a Sufficient Condition

Example 27

We know $L = \{b^m c^m | m > 0\}$ is not regular. Let us consider $L' = a^+ L \cup (b+c)^*$. L' is not regular. If L' would be regular, then we can prove that L is regular (using the closure properties we will see next). However, the Pumping lemma does apply for L' with $n = 1$.

Consider string $ab^n c^n$ and partition $\overbrace{\epsilon}^{u}\overbrace{a}^{v}\overbrace{b^n c^n}^{w}$. Then $uv^i w, \forall i \geq 0$ remains in L'.

This shows the Pumping lemma is not a sufficient condition for a language to be regular. That is, satisfying PL does not always yield a regular language.

Be cautious that you CANNOT use partition $\overbrace{a}^{u}\overbrace{b}^{v}\overbrace{b^{n-1} c^n}^{w}$ to establish a contradiction, because it is the role of ADV (not YOU) to pick a partition

Use of closure properties to show non-regularity

- We can easily prove $L_1 = \{0^n 1^n | n > 0\}$ is not a regular language.
- L_2 = the set of strings with an equal number of 0's and 1's isn't either, but that fact is trickier to prove.
- Regular languages are closed under $\cap$.
- If L_2 were regular, then $L_2 \cap L(0^* 1^*) = L_1$ would be, but it isn't.

Closure properties

Let L and M be regular. Then $L = L(R) = L(D)$ and $M = L(S) = L(F)$ for regular expressions R and S, and DFA D and F.
We have seen that RL are closed under the following operations:

- Union : $L \cup M = L(R + S)$
- Complement : $\bar{L} = L(\bar{D})$
- Intersection : $L \cap M = \overline{\bar{L} \cup \bar{M}}$
- Difference : $L - M = L \cap \bar{M}$
- Concatenation : $LM = L(RS)$
- Closure : $L^* = L(R^*)$
- Prefix : $\mathit{Prefix}(L) = \{x \mid \exists y \in \Sigma^*, xy \in L\}$ (Hint: in D, make final all states in a path from the start state to final state)
- quotient, morphism, inverse morphism, substitution, ...

Quotient

Definition 28

$L_1, L_2 \subseteq \Sigma^*$, $L_1/L_2 = \{x \in \Sigma^* \mid \exists y \in L_2, xy \in L_1\}$.

$\overbrace{q_0 \xrightarrow{x} q}^{x \in L_1/L_2} \overbrace{\xrightarrow{y} \circledcirc}^{y \in L_2}$, where $xy \in L_1$. E.g. $\{00, 111\}/\{\epsilon, 1\} = \{00, 111, 11\}$

Note: $\mathit{Pref}(L) = L/\Sigma^*$.

Theorem 29

$L, R \subseteq \Sigma^*$. *If R is regular, then R/L is also regular.*

Proof Idea: Given an FA, change F to $F' = \{q \in Q \mid \exists y \in L, \delta^*(q, y) \in F\}$,

i.e., mark q as "Accept" if $\overbrace{q_0 \xrightarrow{x} q}^{x \in R/L} \overbrace{\xrightarrow{y} \circledcirc}^{y \in L}$. Note that L can be an arbitrary language.

Example 30

$L = \{a^{n^2} \mid n \geq 0\}$. $L/L = \{a^{n^2 - m^2} \mid m, n \geq 0\} = a(aa)^* + (a^4)^*$. Notice that L is not regular, but L/L is regular.

Morphisms (also called Homomorphisms)

- A morphism h is a mapping: $h : \Sigma \to \Delta^*$
- h can be extended to $h : \Sigma^* \to \Delta^*$ with $h(xy) = h(x)h(y), h(\epsilon) = \epsilon$
- Given a language $L \subseteq \Sigma^*$, $h(L) = \bigcup_{x \in L} \{h(x)\} \subseteq \Delta^*$

Example 31

$h(0) = ab, h(1) = ba, h(2) = \epsilon.$
$h(00212) = ababba;\quad h(0022212222) = ababba;$ (h is many-to-one)
$h(\{0^n 2 1^n | n \geq 0\}) = \{(ab)^n (ba)^n | n \geq 0\}$

Theorem 32

Regular Languages are closed under morphism.

Note that $h(K \cup L) = h(K) \cup (L); h(K \cdot L) = h(K) \cdot h(L); h(K^*) = h(K)^*$.

Inverse Morphisms

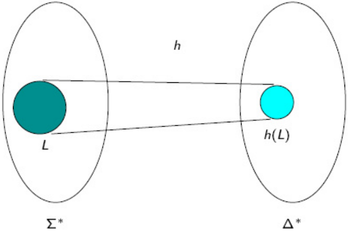
Given $h : \Sigma^* \to \Delta^*$, and $K \subseteq \Delta^*$, the inverse morphism $h^{-1}(K) = \{x \in \Sigma^* \mid h(x) \in K\}$.

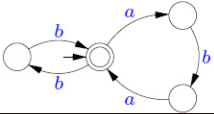
- It is easy to see $L \subseteq h^{-1}(h(L))$. How about R vs. $h(h^{-1}(R))$?
- Note that in Example 31, $\{0^n 2 1^n | n \geq 0\} \subsetneq h^{-1}(\{(ab)^n (ba)^n | n \geq 0\})$

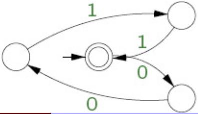
Theorem 33

Regular languages are closed under inverse morphism.

- Consider $h(0) = ab; h(1) = ba$, and FA M accepting $R \subseteq \{a, b\}^*$. Find FA M' accepting $h^{-1}(R) \subseteq \{0, 1\}^*$.
- M and M' have identical states
- $p \overset{ab}{\to} q$ in M iff $p \overset{0}{\to} q$ in M'; $\quad p \overset{ba}{\to} q$ in M iff $p \overset{1}{\to} q$ in M'







Shuffle

Definition 34

$x \| \epsilon = \epsilon \| x = \{x\}$
$ax \| by = a(x \| by) \cup b(ax \| y)$
$K \| L = \bigcup_{x \in K, y \in L} x \| y$

$abb \| aca = \{a\underline{a}bb\underline{ca}, \underline{a}ab\underline{c}b\underline{a}, aabcab, aacabb, aacbab,$
$aacbba, abbaca, ababca, abacba, abacab, acabba, acabab, acaabb\}.$

Theorem 35

If K, L are regular, so is $K \| L$.

- Given $L(M_1) = K, L(M_2) = L$, can you construct an FA M s.t. $L(M) = K \| L$?
- The next page contains an alternative proof using closure properties of regular languages.

Shuffle (cont'd)

Proof.

copies of alphabet

$$\Sigma,\ \Sigma_1 = \{\, a_1 \mid a \in \Sigma \,\},\ \Sigma_2 = \{\, a_2 \mid a \in \Sigma \,\}$$

$$h_1 : \Sigma_1 \cup \Sigma_2 \to \Sigma^* \qquad a_1 \mapsto a \qquad a_2 \mapsto \epsilon$$

$$h_2 : \Sigma_1 \cup \Sigma_2 \to \Sigma^* \qquad a_1 \mapsto \epsilon \qquad a_2 \mapsto a$$

$$g : \Sigma_1 \cup \Sigma_2 \to \Sigma^* \qquad a_1 \mapsto a \qquad a_2 \mapsto a$$

$$\begin{array}{ccccc} abbba & \stackrel{h_1}{\leftarrow} & a_1 b_1 a_2 c_2 b_1 a_2 c_2 b_1 a_1 & \stackrel{h_2}{\rightarrow} & acac \\ \in K & & \downarrow g & & \in L \\ & & abacbacba & & \end{array}$$

$$K \parallel L = g(\, h_1^{-1}(K) \cap h_2^{-1}(L) \,)$$

$\frac{1}{2}L$

Definition 36

$\frac{1}{2}L = \{x \in \Sigma^* | \exists y \in \Sigma^*, xy \in L; \ \ |y| = |x|\}.$

Theorem 37

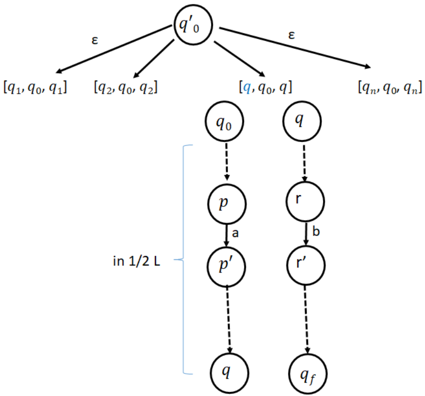
If L is regular, so is $\frac{1}{2}L$.

Proof.

guess middle state, simulate halves in parallel
$Q' = \{q'_0\} \cup Q \times Q \times Q$ (Note: middle, 1st, 2nd)
$\delta'(q'_0, \epsilon) = \{[q, q_0, q] | q \in Q\}$ – ϵ-move
$\delta'([q, p, r], a) = \{[q, \delta(p, a), \delta(r, b)]|$ some $b \in \Sigma\}$
$F' = \{[q, q, p] | q \in Q, p \in F\}$ □

Note: $x \in \frac{1}{2}L$ if $\exists q \in Q, v \in \Sigma^*$, $q_0 \overbrace{\rightarrow \dots \rightarrow}^{x} q$; $q \overbrace{\rightarrow \dots \rightarrow}^{v} p$; $|x| = |v|$ and $p \in F$.

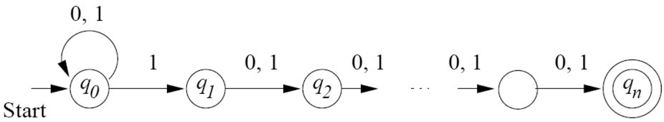
$\frac{1}{2}L$



- Can you show $\frac{1}{3}L = \{x \in \Sigma^* | \exists yz \in \Sigma^*, xyz \in L; \ \ |x| = |y| = |z|\}$ to be regular as well?
- How about $\frac{2}{3}L_{*-*} = \{xz | \exists x, y, z \in \Sigma^*, xyz \in L; |x| = |y| = |z|\}$? (Not regular)

Exponential Blow-Up in Subset Construction

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



$$L(N) = \{x\,\overbrace{1c_2c_3\cdots c_n}^{\text{length } n} : x \in \{0,1\}^*, c_i \in \{0,1\}\}.$$

- Suppose an equivalent DFA $D = (Q_D, \Sigma, \delta_D, q_0', F_D)$ with <u>fewer than</u> 2^n states exists. read.
- There are 2^n bitsequences $a_1a_2\cdots a_n \in \{0,1\}^n$.
- $\exists q \in Q_D,\ a_1a_2\cdots a_n,\ b_1b_2\cdots b_n \in \{0,1\}^n,\ a_1a_2\cdots a_n \neq b_1b_2\cdots b_n$ $\delta_D^*(q_0', a_1a_2\cdots a_n) = q = \delta_D^*(q_0', b_1b_2\cdots b_n)$

Exponential Blow-Up (Cont'd)

Let i be the first position from the right such that $a_i \neq b_i$. I.e.,
$a_1 \cdots a_{i-1} 1 a_{i+1} \cdots a_n$
$b_1 \cdots b_{i-1} 0 b_{i+1} \cdots b_n$
and $a_{i+1} \ldots a_n = b_{i+1} \ldots b_n$

Now

$$\delta_D^*(q_0', a_1 \cdots a_{i-1} 1 a_{i+1} \cdots a_n 0^{i-1}) = \delta_D^*(q_0', b_1 \cdots b_{i-1} 0 b_{i+1} \cdots b_n 0^{i-1})$$

as for some $r \in Q_D$

$$q_0' \stackrel{a_1 \cdots a_{i-1} 1 a_{i+1} \cdots a_n}{\rightarrow} q \stackrel{0^{i-1}}{\rightarrow} r$$

and

$$q_0' \stackrel{b_1 \cdots b_{i-1} 0 b_{i+1} \cdots b_n}{\rightarrow} q \stackrel{0^{i-1}}{\rightarrow} r.$$

Furthermore

$$\delta_D^*(q_0', a_1 \cdots a_{i-1} 1 a_{i+1} \cdots a_n 0^{i-1}) \in F_D$$
$$\delta_D^*(q_0', b_1 \cdots b_{i-1} 0 b_{i+1} \cdots b_n 0^{i-1}) \notin F_D$$

– A contradiction!

Decision Properties

- A decision property for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.
- Example: Is language L empty?
  - Suppose the representation is a DFA (or a RE that you will convert to a DFA).
  - Can you tell if $L(A) = \emptyset$ for DFA A?
- The complexity depends on how languages are represented. E.g., DFA vs. NFA vs. RE for regular languages.

Why Decision Properties

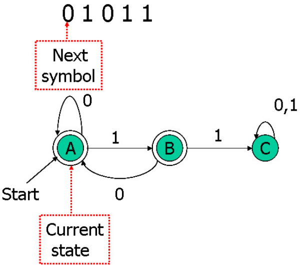
- When we talked about protocols represented as DFAs, we noted that important properties of a good protocol were related to the language of the DFA.
- Example: Does the protocol terminate? = Is the language finite?
- Example: Can the protocol fail? = Is the language nonempty?

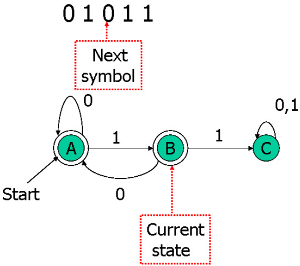
The Membership Question

Definition 38

Is string w in regular language L?

- Assume L is represented by a DFA A.
- Simulate the action of A on the sequence of input symbols forming w. (Question: What is the running time?)





The Emptiness Problem

Definition 39

Given a regular language, does the language contain any string at all.

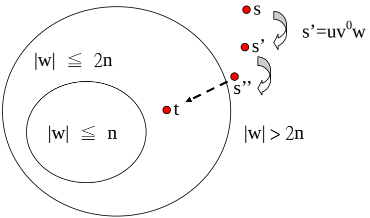
- Assume representation is DFA.
- Construct the transition graph.
- Compute the set of states reachable from the start state.
- If any final state is reachable, then yes, else no.
- Question: What is the running time?

The Infiniteness Problem

Definition 40

Is a given regular language infinite?

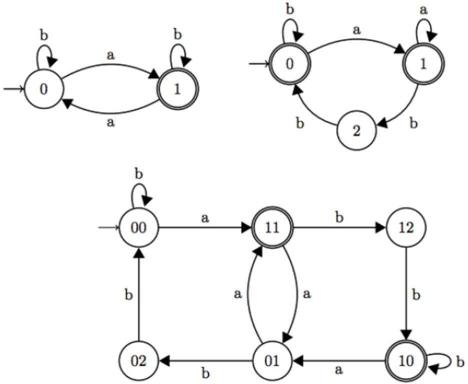
- Start with a DFA for the language.
- **Key idea**: if the DFA has n states, and the language contains any string of length n or more, then the language is infinite.
- **Second key idea**: if there is a string of length $> n$ (= number of states) in L, then there is a string of length between n and $2n - 1$.

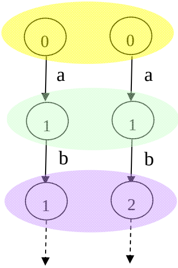


- Test for membership all strings of length between n and $2n - 1$. If any are accepted, then infinite, else finite.

The Product Automaton $M \times N$

Idea: Running two automata M and N in parallel.





Running the two FAs in parallel

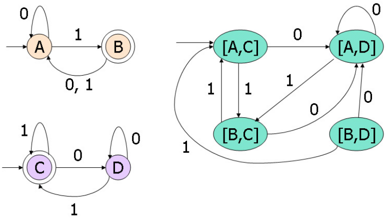
The Equivalence Problem

Definition 41

Given regular languages L and M, is $L = M$?

- Algorithm involves constructing the product DFA from DFA's for L and M.
- Let these DFA's have sets of states Q and R, respectively.
- Product DFA has set of states $Q \times R$. I.e., pairs $[q, r]$ with q in Q, r in R.

- Make the final states of the product DFA be those states $[q, r]$ such that exactly one of q and r is a final state of its own DFA. Thus, the product accepts w iff w is in exactly one of L and M.
- The product DFA's language is empty iff L = M.

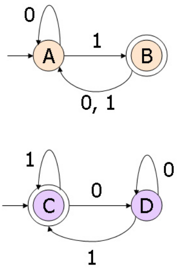


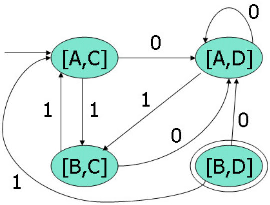
The Containment Problem

Definition 42

Given regular languages L and M, is $L \subseteq M$?

- Algorithm also uses the product automaton.
- How do you define the final states $[q, r]$ of the product so its language is empty iff $L \subseteq M$?
  - Answer: q is final; r is not.





Note: the only final state is unreachable, so containment holds.

The Minimum-State DFA for a Regular Language

- In principle, since we can test for equivalence of DFA's we can, given a DFA A find the DFA with the fewest states accepting $L(A)$.
- Test all smaller DFA's for equivalence with A.
- But that's a terrible algorithm.

– Efficient State Minimization

- Construct a table with all pairs of states.
- If you find a string that *distinguishes* two states (takes exactly one to an accepting state), mark that pair.
- Algorithm is a recursion on the length of the shortest distinguishing string.

Equivalence Relation

Definition 43

A binary relation R on a set S is a subset of $S \times S$. An equivalence relation on a set satisfies

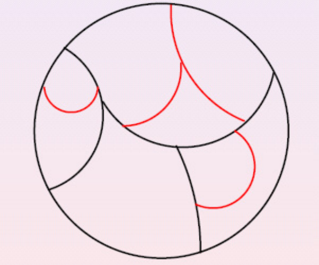
1. Reflexivity: For all x in S, xRx
2. Symmetry: For $x, y \in S$ $xRy \Leftrightarrow yRx$
3. Transitivity: For $x, y, z \in S$ $xRy \wedge yRz \Rightarrow xRz$

- Every equivalence relation on S partitions S into equivalence classes.
- The number of equivalence classes is called the index of the relation.
- An equivalence class containing x is written as $[x]$.
- E.g., *Mod* 3 is an equivalence relation which partitions $\mathbb{N}$ into equivalence classes $\{0, 3, 6, \ldots\}$, $\{1, 4, 7, \ldots\}$, and $\{2, 5, 8, \ldots\}$. The index is 3.

Refinement

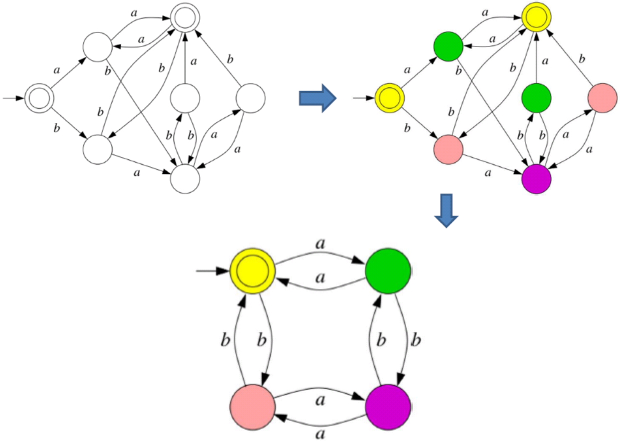
Definition 44

An equivalence relation R_1 is a refinement of R_2 if $R_1 \subseteq R_2$, i.e. $(x, y) \in R_1 \Rightarrow (x, y) \in R_2$



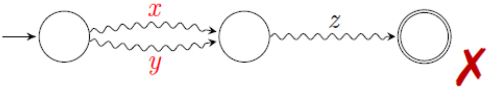
Minimizing DFAs

The Idea: Identify ”indistinguishable states”; Merge those states.



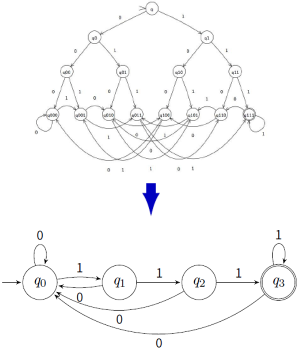
Distinguishable Strings

- Given a language L, strings x, y are distinguishable by L if there is string z such that $xz \in L$ **and** $yz \notin L$ (or the other way round).
- Equivalently, strings x, y are indistinguishable if for every string z, $xz \in L \Leftrightarrow$yz $\in L$. (Later such x, y are written as $x \equiv_L y$.)
- If x and y are distinguishable by L, any DFA accepting L must reach different states upon reading x and y.



- Given a DFA M and a state q, let $L_M(q) = \{w \mid q \xrightarrow{w} q_f, q_f \in F\}$, i.e., the set of strings leading M to acceptance from q.
  - It is possible to have $p, q \in Q$, $L_M(p) = L_M(q)$. Such states will be merged in state minimization procedure.

Minimal DFA and Distinguishability

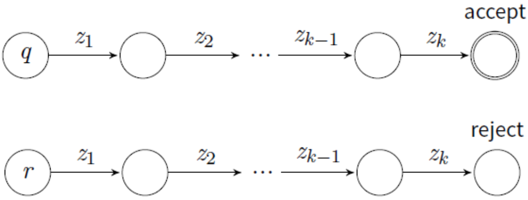


- Distinguishable strings must be associated with different states.
- Indistinguishable strings may end up in the same state.
- Indistinguishability induces an equivalence relation over Σ^*, which is of *finite index* for regular languages (**Myhill-Nerode Theorem**).

DFA minimial $\Leftrightarrow$ Every pair of states is distinguishable

Distinguishable States

Two states q and r are *distinguishable* if $\exists z_1, \ldots, z_k$



I.e., $L_M(q) \neq L_M(r)$.

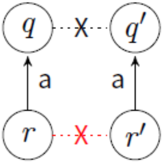
Indistinguishability (over Q) is also an equivalence relation, which partitions the set of states into equivalence classes.

Finding (In)distinguishable States

Phase 1:

If q is accepting and q' is rejecting
Mark (q, q') as distinguishable (X)

Phase 2:



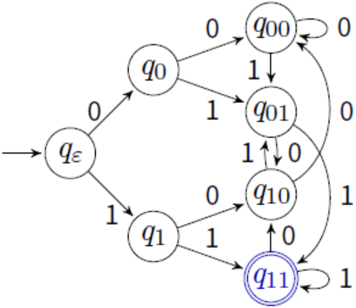
If (q, q') are marked
Mark (r, r') as distinguishable (X)

Phase 3:

Unmarked pairs are indistinguishable
Merge them into groups

An Example

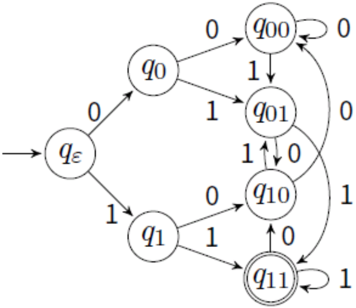
(Phase 1) q_{11} is distinguishable from all other states



	q_ε	q_0	q_1	q_{00}	q_{01}	q_{10}
q_0						
q_1						
q_{00}						
q_{01}						
q_{10}						
q_{11}	X	X	X	X	X	X

An Example (Cont'd)

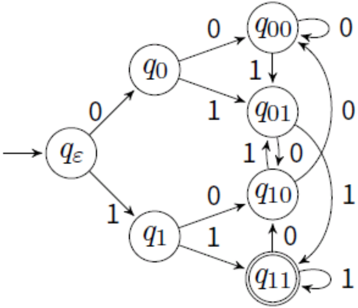
(Phase 2) Looking at $(r, r') = (q_\epsilon, q_0)$, Neither $(q_0, q_{00})_{input\ 0}$ nor $(q_1, q_{01})_{input\ 1}$ are distinguishable



q_0						
q_1						
q_{00}						
q_{01}						
q_{10}						
q_{11}	X	X	X	X	X	X
	q_ε	q_0	q_1	q_{00}	q_{01}	q_{10}

An Example (Cont'd)

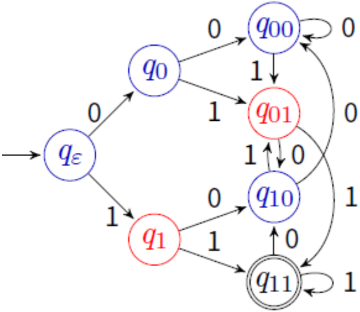
(Phase 2) Looking at $(r, r') = (q_\epsilon, q_1)$, $(q_1, q_{11})_{input\ 1}$ is distinguishable



q_0						
q_1	X					
q_{00}						
q_{01}						
q_{10}						
q_{11}	X	X	X	X	X	X
	q_ε	q_0	q_1	q_{00}	q_{01}	q_{10}

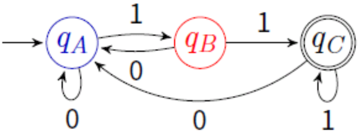
Example

(Phase 3) Merge states into groups (also called *equivalence classes*)



q_0	A					
q_1	X	X				
q_{00}	A	A	X			
q_{01}	X	X	B	X		
q_{10}	A	A	X	A	X	
q_{11}	X	X	X	X	X	X
	q_ε	q_0	q_1	q_{00}	q_{01}	q_{10}

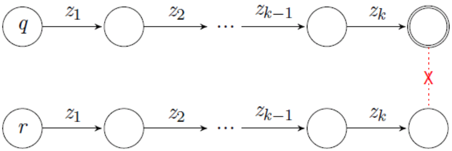
Minimized DFA:

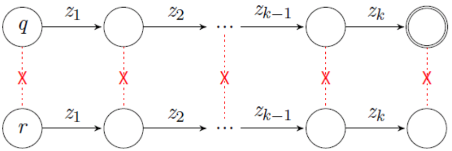


Why It Works?

Why have we found all distinguishable pairs?

- Because we work backwards!





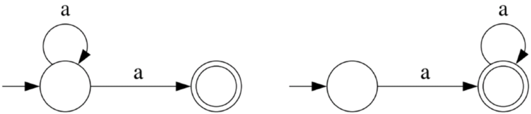
- It suffices to iterate Phase 2 at most $|Q|^2$ times. Why? What is the shortest string that distinguishes two states?

Unique Minimum DFA

Theorem 45

Every regular language has a single minimal automaton (up to isomorphism).

However, minimal NFAs are not unique as the following examples show.



Theorem 46

Given an NFA M and a number k, deciding if there is another NFA M' equivalent to M with at most k states is PSPACE-complete (polynomial-space complete).

Another way of Characterizing Regular Languages – Residuals of Languages

- The residual of a language $L \subseteq \Sigma^*$ with respect to a word w is the language
$$L^w = \{u \in \Sigma^* \mid wu \in L\}$$
- A language $L' \subseteq \Sigma^*$ is a residual of L if $L' = L^w$ for some $w \in \Sigma^*$.
- We define "indistinguishability" over strings
$$x \equiv_L y \Leftrightarrow (\forall z \in \Sigma^*, xz \in L \Leftrightarrow yz \in L).$$
$\equiv_L$ is an equivalence relation. Note that $x \equiv_L y \Leftrightarrow L^x = L^y$.
- Note that $\forall a \in \Sigma$, $(L^x = L^y) \Rightarrow (L^{xa} = L^{ya})$
  - The implication is that if we treat each residual L^w of L as a "state" and define $\delta(L^w, a) = L^{wa}$, δ is "consistent" in that $L^x = L^y$ (same state) implies $\delta(L^x, a) = L^{xa} = L^{ya} = \delta(L^y, a)$ (also same state).

Myhill-Nerode Theorem

Theorem 47 (Myhill-Nerode Theorem)

A language is regular iff it has finitely many residuals.

Proof.

($\Rightarrow$) Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. The language recognized by A with q the initial state, denoted by $L_A(q)$, is a residual of $L(A)$. Moreover, if $\delta(q_0, x) = \delta(q_0, y) = q$, for some q, then $L^x = L^y$.

($\Leftarrow$) Let $L \subseteq \Sigma^*$ be a regular language, the canonical DFA of L $M_L = (Q_L, \Sigma, \delta_L, q_{0L}, F_L)$ is

- Q_L is the set of residuals of L, i.e., $Q_L = \{L^w \mid w \in \Sigma^*\}$
- $\delta_L(R, a) = L^{wa}$, where $R = L^w$, for some w, where $R \in Q_L$ and $a \in \Sigma$
- $q_{0L} = L^\epsilon = L$
- $F_L = \{R \in Q_L \mid \epsilon \in R\}$

It is easy to show that $L(M_L) = L$.

$\square$

An Example of M_L

$L = a^*b^* \subseteq \{a, b\}^*$

- $Q_L = \{Q_1, Q_2, Q_3\}$, where
  $Q_1 = a^*b^*(= L^\epsilon), Q_2 = b^*(= L^{ab}), Q_3 = \emptyset(= L^{aba})$
  (How about L^{aaa}, L^{aabbb}?)
- $q_{0L} = Q_1$
- $F_L = \{Q_1, Q_2\}$
- $\delta_L(Q_1, a) = Q_1$, $\delta_L(Q_1, b) = Q_2$, $\delta_L(Q_2, a) = Q_3$, $\delta_L(Q_2, b) = Q_2$,
  $\delta_L(Q_3, a \mid b) = Q_3$.
  - E.g., $\delta_L(Q_2, a) = \delta_L(L^{ab}, a) = L^{aba} = \emptyset = Q_3$

Uniqueness of the Canonical DFA

Theorem 48

If L is regular, then M_L is the unique minimal DFA up to isomorphism recognizing L.

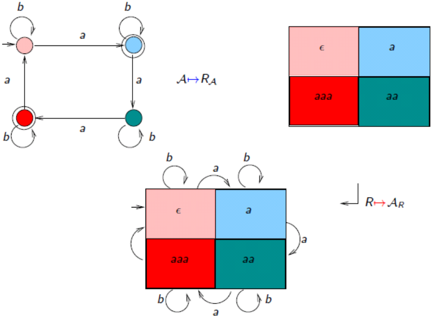
- Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting L. Define a relation $R_{\mathcal{A}}$ as follows:
  - For $x, y \in \Sigma^*, xR_{\mathcal{A}}y \Leftrightarrow \delta(q_0, x) = \delta(q_0, y)$.

FACT: $R_{\mathcal{A}}$ refines $\equiv_L$.

- Can you show $xR_{\mathcal{A}}y \implies x \equiv_L y$?
- If so, $|Q| \geq$ the index of L under $\equiv_L$. Hence, M_L is a minimal DFA.

Deterministic Finite Automata and the Induced Relations - An Example

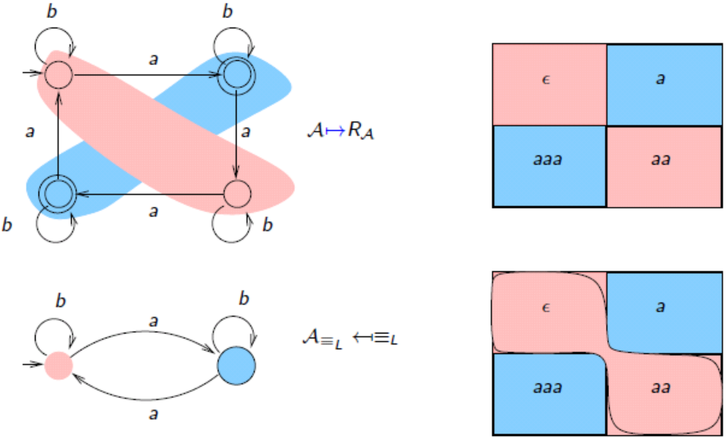
Example: $L(\mathcal{A})$ is "Odd number of a's":



Four equivalence classes under $R_{\mathcal{A}}$, $[\epsilon] = \{\epsilon, b, bb, aaaa, ...\}$;
$[a] = \{a, ab, abb, ...\}$; $[aa] = \{aa, aab, aabb....\}$; $[aaa] = \{aaa, aaab, aaabb, ...\}$.

Canonicity of M_L

Let $\mathcal{A}$ be a DFA for L with no unreachable states. Let $A_{\equiv_L}$ be M_L. In the following example, $L^a = L^{aaa}$ and $L^\epsilon = L^{aa}$, i.e., $a \equiv_L aaa$ and $\epsilon \equiv_L aa$.)



In Summary

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Consider two computations

$$q_0 \xrightarrow{x} p \xrightarrow{z} r_1 \qquad q_0 \xrightarrow{y} q \xrightarrow{z} r_2$$

- The min. proc. is to identify all *indistinguishable* pairs (p, q) such that $L_{\mathcal{A}}(p) = L_{\mathcal{A}}(q)$. That is, $\forall z$, either $r_1, r_2 \in F$ or $r_1, r_2 \notin F$.
  - Define $R_{\mathcal{A}}$ over Σ^* as $xR_{\mathcal{A}}y$ iff $\delta^*(q_0, x) = \delta^*(q_0, y)$, i.e., $p = q$ in Fig.
- Another viewpoint is to identify x, y with identical residual, i.e., $L^x = L^y$. In this case, x and y are *indistinguishable* strings.
  - The notion of residuals induces an equivalence relation $\equiv_L$ over Σ^* s.t. $x \equiv_L y$ iff $L^x = L^y$
  - **Myhill-Nerode Thm**: $\equiv_L$ is of finite index iff L is regular.
- $R_{\mathcal{A}}$ refines $\equiv_L \quad \Rightarrow \quad \equiv_L$ induces a minimal equivalent DFA.

Applications of the Myhill-Nerode Theorem

The MN theorem can be used to show that a particular language is regular without actually constructing the automaton or to show conclusively that a language is not regular.
Example. Is the following language regular

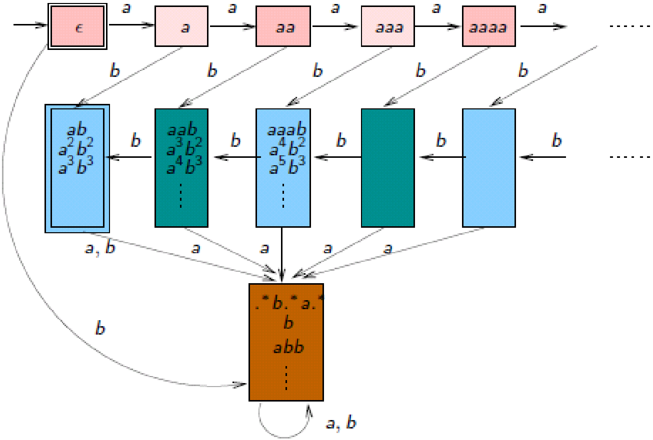
1. $L_1 = \{xy : \ |x| = |y|, x, y \in \Sigma^*\}$?
2. Example. What about the language
   $L_2 = \{xy : \ |x| = |y|, x, y \in \Sigma^* \text{ and } y \text{ ends with a } 1\}$?
3. Example. What about the language
   $L_3 = \{xy : \ |x| = |y|, x, y \in \Sigma^* \text{ and } y \text{ contains a } 1\}$?

Applications of the Myhill-Nerode Theorem (cont'd)

1. For the language L_1 there are two equivalence classes of $\equiv_{L_1}$. The first C_1 contains all strings of *even* length and the second C_2 all strings of *odd* length.
2. For L_2 we have the additional constraint that y ends with a 1. Class C_2 remains the same as that for L_1. Class C_1 is refined into classes C_1' which contains all strings of even length that end in a 1 and C_1'' which contains all strings of even length which end in a 0. Thus L_1 and L_2 are both regular.
3. For L_3 we have to distinguish for example, between the even length strings in the sequence 01, 0001, 000001,..., as 00 distinguishes the first string from all the others after it in the sequence ($0100 \notin L_3$, but $00100, 0000100 \ldots \in L_3$), 0000 distinguishes the second from all the others ...

The $\equiv_L$ for $L = \{a^n b^n \; n \geq 0\}$

Describe the equivalence classes of $\equiv_L$ for $L = \{a^n b^n \; n \geq 0\}$



The automaton is NOT of finite state.

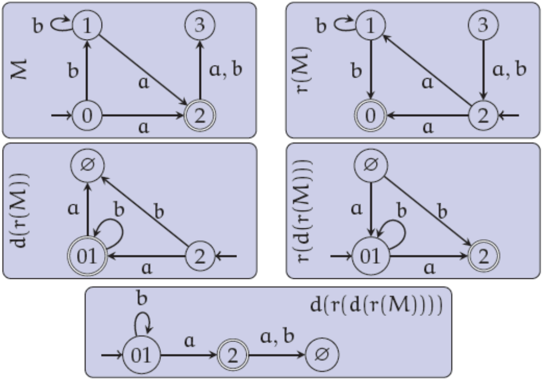
More on FA Minimization

- For any regular language, there is a *unique* minimal DFA.
- By finding the minimal DFA, we can also prove equivalence (or not) of different FA
- In general the idea behind minimization is:
  - Throw away unreachable states (easy)
  - Merge equivalent states
- There are two well-known algorithms for minimization:
  - **Hopcroft's algorithm**: find and eliminate equivalent states by partitioning the set of states – $O(n \log n)$ time
  - **Brzozowski's algorithm**: "double reversal" – exponential worst-time complexity
- There are many versions of the "partitioning" algorithm. In practice, there is no clear winner, different algorithms run faster on different inputs.
- Double reversal algorithm also works for NFAs (resulting in the minimal equivalent DFA)
- NFA minimization is intractable (i.e., hard).

Brzozowski's algorithm

Theorem 49

Applying "reachable(subset(reverse[reachable(subset(reverse(M))]))" results in the minimal DFA that implements M. (Brzozowski, 1962)



(Fig. from https://dsacl3-2019.github.io/slides/regular-fsa.pdf)