**NON-REGULAR LANGUAGES.**

Consider:
- \( L_{eq} = \{ w \mid w \text{ has equal \# of } 0\text{'s and } 1\text{'s} \} \)
- \( L_{alt} = \{ w \mid w \text{ has alternating } 0\text{'s and } 1\text{'s} \} \)

**Claim**
- \( L_{alt} \) is regular

**Proof**
- Reg. Exp. \( R = (\varepsilon + 0)(10)^* (\varepsilon + 1) \)

**Claim**
- \( L_{eq} \) is non-regular

**Question** How do we prove this?

**Intuition**
- DFAs can count up to \( n = |Q| \) only
- \( L_{eq} \) requires unbounded counting.

**Pumping Lemma**

Let \( L \subseteq \Sigma^* \) be regular

Then \( \exists \) constant \( n \) (which depends on \( L \)) such that

For all \( w \in L \) with \( |w| \geq n \),

can decompose \( w = xyz \)

such that

a) \( |y| \geq 1 \) (\( y \neq \varepsilon \))

b) \( |xy| \leq n \)

c) \( \forall k \geq 0, \ xy^kz \in L \).

**Observe** This is a property of regular languages which we can use to show that a given language is not regular.

**Idea**
Pick \( w \in L \) such that we can easily argue that \( xy^kz \notin L \) for some choice of \( k \).

Problem must do so regardless of \( n \) and \( x, y, z \) choices.
Application: Proving $L$ is non-regular.

Idea: Play following game:

a) You assume $L$ is regular.

b) Adversary thinks of value $n > 0$ (but doesn't tell you the value of $n$).

c) You pick string $w \in L$ with $|w| \geq n$ (in terms of unknown parameter $n$).

d) Adversary thinks of decomposition $w = xyz$ with $y \neq \varepsilon$, and $|xy| \leq n$ (but doesn't tell you $x$, $y$, or $z$).

e) You find $k \geq 0$ such that $xy^kz \notin L$ (without knowing $x$, $y$, or $z$).

Note: Order of steps is very important.

Note: Lack of information about $n$, $x$, $y$, or $z$ makes this quite difficult — but, surprisingly, we can still play the game successfully.

Example: $L_{eq}$ is non-regular.

Proof: Assume $L$ is regular and apply P.L.

By P.L. there exists some $n > 0$.

We choose $w = 0^n 1^n$.

Clearly $w \in L$ and $|w| \geq n$.

Note: Clever choice of $w$.

By P.L. $w = xyz$ such that \[ \sum y \neq \varepsilon \cdot |xy| \leq n. \]
BUT SINCE $|xy| \leq n$, IT MUST BE THE CASE THAT $x, y$ HAVE ONLY 0's.

LET $|x| = a, |y| = b$, AND $|xy| = a+b \leq n$, $b > 0$.

THEN $w = \overbrace{0^a \ 0^b \ 0^{n-a-b}}^x \ y \ z$

WE CHOOSE $k = 0$

$\Rightarrow xy^k z \in L_{EQ}$

$\Rightarrow xz \in L_{EQ}$

$\Rightarrow 0^{n-b} \ 1^n \in L_{EQ}$

BUT FOR $b > 0$, THIS IS NOT TRUE.

CONTRADICTION

THUS ASSUMING $L_{EQ}$ IS REGULAR GIVES A CONTRADICTION.

$\Rightarrow L_{EQ}$ IS NON-REGULAR.

NOTE CHOOSING ANY $k \neq 1$ WILL WORK!

EXAMPLE 2 $L_P = \{w |$ LENGTH OF $w$ IS A PRIME NUMBER $\}$

CLAIM $L_P$ IS NOT REGULAR.

PROOF ASSUME $L_P$ IS INDEED REGULAR AND APPLY P.L.

BY P.L. THERE EXISTS AN $n > 0$. 
We choose prime $p \geq n+2$

(Since there an infinite number of primes, we can always claim $p \geq n+2$ exists, without knowing $n$).

Choose $w = 1^p$.

Clearly $w \in L$ and $|w| \geq n$.

By P.L. $w = xyz$ such that

\[
\{ \\
\cdot y \neq e \\
\cdot |xy| \leq n
\} \tag{4}
\]

Suppose $|x| = a$, $|y| = b$ with $b > 0$, $|a+b| \leq n$.

We choose $k = p - b$.

Get $xy^{p-b}z \in L_p$.

But $|xy^{p-b}z| = a + (p-b)b + (p-a-b)$

$= (p-b)b + (p-b)$

$= (p-b)(b+1)$

Note $b+1 \geq 2$ since $b \geq 1$.

Note $p-b \geq 2$ since $\{ \\
\cdot p \geq n+2 \\
\cdot b \leq n
\}$.

Thus $|xy^{p-b}z|$ has 2 non-trivial factors and cannot be a prime.

Contradiction
Proof of Pumping Lemma.

Given regular language \( L \), let \( M \) be smallest DFA for \( L \), with \( |Q| = n \).

Consider any \( w \in L \) with \( |w| = m \geq n \).

Since \( w \in L(M) \) consider accepting path \( \hat{s}(q_0, w) \) in \( M \) for \( w = a_1 a_2 \ldots a_m \).

Setting \( p_i = \hat{s}(q_0, a_1 a_2 \ldots a_i) \) and \( p_0 = q_0 \).

Since \( m \geq n \) and each \( p_i \) \((0 \leq i \leq m)\) belongs to \( G \) which has only \( n \) states.

By pigeon-hole \( p_0, p_1, \ldots, p_m \) not all distinct.

Let \( 0 \leq i < j \leq n \) be such that \( p_i = p_j \).

Cycle

\[ x = a_1 \ldots a_i \]

\[ y = a_{i+1} \ldots a_j \]

\[ z = a_{j+1} \ldots a_m \]

Observe:

a) \( |y| = j - i \geq 1 \) (since \( i < j \))

b) \( |xy| = j \leq n \) (choose \( i < j \leq n \))

c) \( \forall k > 0, \ xy^k z \in L(M) \) (pumping cycle)

Done.
Remark. Computers have fixed/bounded memory and are like DFA's, so cannot do unbounded counting and only accept regular languages.

But since memory is really large & can add more memory on demand, we choose to view as "infinite" memory in later abstractions.

Closure Properties

Theorem. If $L_1, L_2$ are regular, then so are

- $L_1 \cup L_2$
- $L_1 \cdot L_2$
- $L_1^*$

Proof. Regular $L_1, L_2$ must have R.E. $R_1, R_2$ such that

\[
L_1 = L(R_1) \\
L_2 = L(R_2)
\]

Then

- $L_1 \cup L_2 = L(R_1) \cup L(R_1) = L(R_1 \cup R_2)$
- $L_1 \cdot L_2 = L(R_1) \cdot L(R_2) = L(R_1 \cdot R_2)$
- $L_1^* = L(R_1)^* = L(R_1^*)$

must all be regular as they are generated by $R_i$.

Thus class of regular languages is closed under operations $\cup, \cdot, ^*$ — in that, we cannot create languages outside this class by applying the operations to languages inside the class.
REMARK IN ABOVE PROOF, WE USED R.E. REPRESENTATION OF REGULAR LANGUAGES — BUT IN GENERAL WE CAN USE ANY THAT IS CONVENIENT:

DFA | NFA | ε-NFA | R.E

**Complement**

DEFN \( \overline{L} = \Sigma^* - L \)

**Theorem** REGULAR LANGUAGES ARE CLOSED UNDER COMPLEMENTATION.

**Proof** CONSIDER REGULAR \( L \)

ASSUME DFA \( M = (Q, \Sigma, \delta, q_0, F) \) ACCEPTS \( L \)

CONSTRUCT DFA \( \overline{M} = (Q, \Sigma, \delta, q_0, F) \)

CLEARLY \( L(\overline{M}) = \overline{L(M)} = \overline{L} \)

THUS \( \overline{L} \) HAS DFA AND IS REGULAR.

**Intersection**

**Theorem** \( L_1, L_2 \) REGULAR \( \Rightarrow L_1 \cap L_2 \) REGULAR

**Proof** DE MORGAN'S LAW \( L_1 \cap L_2 \) = \( \overline{L_1} \cup \overline{L_2} \)

USE CLOSURE UNDER \( \cup \) AND...

**Question** DO YOU SEE HOW TO MAKE THIS PROOF CONSTRUCTIVE?

**Remark** SEE BOOK FOR DIRECT CONSTRUCTION ON DFAS — INTERESTING IDEA OF A PRODUCT OF 2 DFAS.
**DEFN**
- $e^R = e$
- $w^R = (a_1a_2 \ldots a_n)^R = a_na_{n-1} \ldots a_1$
- $L^R = \{ w^R \mid w \in L \}$

**Theorem**
Regular languages are closed under reversal.

**Proof**
Extend reversal to RE languages inductively.

**Basis**
\[
\begin{align*}
\epsilon^R &= \epsilon \\
\emptyset^R &= \emptyset \\
\alpha^R &= \alpha + \alpha \in \Sigma
\end{align*}
\]

**Induction**
\[
\begin{align*}
(E_1 + E_2)^R &= E_1^R + E_2^R \\
(E_1 \cdot E_2)^R &= E_2^R E_1^R \\
(E_1^*)^R &= (E_1^*)^*
\end{align*}
\]

Verify via induction; in each step above,
\[L(E^R) = L(E)^R\]

**Example**
\[E = abc + bca^*b\]
\[\Rightarrow E^R = cba + aca^*b.\]

**Question**
Think about reversing an $\epsilon$-NFA!

**See book** for other operations:
- Homomorphisms
- Inverse homomorphisms.
DECISION PROBLEMS

Fix any property $P$ of languages

Input: regular language $L$

Output: does $L$ have property $P$? — Yes/No.

DECISION ALGORITHM

Input: any representation of $L$ (DFA/NFA/G-NFA/RE)
— can easily convert between these.

Output: Yes/No.

Must always terminate in finite time with correct answer.

EMPTINESS

Problem: decide if a regular language $L$ is empty.

Assume: we convert representation of $L$ into RE: $R$.

Algorithm: recursively computes boolean predicate

$$\text{empty}(R) <\equiv \text{true}$$

RECURSIVE ALGORITHM

Basis

\[
\begin{align*}
\text{empty}(\emptyset) & = \text{true} \\
\text{empty}(\epsilon) & = \text{false} \\
\text{empty}(a) & = \text{false}, \quad \forall a \in \Sigma
\end{align*}
\]

Induction

\[
\begin{align*}
\text{empty}(R^\star) & = \text{false} \\
\text{empty}(R + S) & = \text{empty}(R) \land \text{empty}(S) \\
\text{empty}(R \cdot S) & = \text{empty}(R) \lor \text{empty}(S)
\end{align*}
\]
**ALTERNATIVE** use DFA representation and check if any state in F is reachable from q0 (see book).

**MEMBERSHIP**

**Problem** given w ∈ Σ* and L ⊆ Σ*, where L is regular, does w belong to L?

**Algorithm** assume DFA M for L, and simulate M on w.

See book on simulating NFA/G-NFA directly on w, without first converting to DFA, for a possible increase in efficiency.

**EQUALITY**

Given regular languages L₁ and L₂ determine whether L₁ = L₂.

**Idea** consider L = (L₁ ∪ L₂) ∪ (L₁ ∩ L₂)

Clearly L is regular (via closure properties)

Also L₁ = L₂ ⇔ L = Ø

**Algorithm**

1) Compute L (using closure construction)
2) Compute R.E. for L, say R
3) Compute EMPTY(R)

See book for DFA-based algorithm.
Finiteness

Problem: Given regular language $L$, is $|L|$ finite?
Assume: DFA $M$ for $L$ has $|Q| = n$.

Theorem: $L$ is infinite if and only if there exists $w \in L$ such that $n \leq |w| < 2n$.

Algorithm: For $w \in \Sigma^*$ with $n \leq |w| < 2n$, test membership of $w$ in $L$.

Proof ($\Leftarrow$): Given $w \in L$ with $n \leq |w|$, apply P.I. $w = xyz$ with $y \neq \varepsilon$ such that $\forall k \geq 0$, $xy^kz \in L$.

Thus, $L$ is infinite.

Proof ($\Rightarrow$): Suppose $L$ is infinite.
Clearly, if $w \in L$, such that $|w| \geq n$.
Choose $\hat{w}$ to be the shortest string in $L$ of length at least $n$.

Claim: $|\hat{w}| < 2n$.

Suppose $|\hat{w}| \geq 2n$.
By P.I., $\hat{w} = xyz$ with $|xy| \leq n$, $|y| \geq 1$ and $xy^0z = xz \in L$.

But:
@ $|xz| \geq n$ since $|xz| = |\hat{w}| - |y| \geq 2n - n = n$.
@ $|xz| < |\hat{w}|$ since $|y| \geq 1$.

This contradicts choice of $\hat{w}$ as shortest such string.

Therefore, $\exists w \in L$ with $n \leq |w| < 2n$. Done.
DFA MINIMIZATION

Given: DFA \( M = (Q, \Sigma, S, q_0, F) \)

Find: DFA \( M' \) with fewest possible number of states, such that \( L(M') = L(M) \).

IDEA: Partition \( Q \) into equivalence classes and collapse.

Equivalence: \( p \equiv q \) if for all \( w \in \Sigma^* \),
\[ S(p, w) \in F \iff S(q, w) \in F \]

Classes: Partition \( Q = C_1 \cup C_2 \cup \ldots \cup C_r \) such that
for all \( p \in C_i \), \( q \in C_j \),
\[ p \equiv q \implies i = j \]

Finding \( C_i \): Idea—identify inequivalent pairs \((p, q)\).

Observe: "(Proof)" of inequivalence for \( p, q \) is \( w \in \Sigma^* \) such that one of \( S(p, w) \) and \( S(q, w) \)
is final and the other is non-final.

Proof's length: \(|w| = m\).

\[ \begin{array}{c}
  p \xrightarrow{w_1} p_1 \xrightarrow{w_2} p_2 \ldots \xrightarrow{w_{m-1}} p_{m-1} \xrightarrow{w_m} p_m \\
  q \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \ldots \xrightarrow{w_{m-1}} q_{m-1} \xrightarrow{w_m} q_m \\
\end{array} \]

One is final but not the other.

Note: \( w_1 \ldots w_m \) is length \( n-1 \), proof of inequivalence of \( p_i, q_i \).

Define: say \( p \equiv_i q \) if for \( |w| \leq i \)
\[ S(p, w) \in F \iff S(q, w) \in F \]

Means: no inequivalence proof of length \( \leq i \).
CLEARLY $p \equiv_{i+1} q$ IF AND ONLY IF FOR SOME $a \in \Sigma$

$S(p, a) \equiv_i S(q, a)$

ALGORITHM (TO RECURSIVELY DETERMINE $\equiv_i$)

STEP 0

PARTITION $Q = C_1 \cup C_2$, WITH $C_1 = F$, $C_2 = Q - F$

(Since $p \not\equiv_0 q$ IF AND ONLY IF ONE IS FINAL BUT NOT BOTH)

STEP $i+1$

DECIDE $p \equiv_{i+1} q$ IF AND ONLY IF $\forall a \in \Sigma,$

$S(p, a) \equiv_i S(q, a)$

OBTAIN REFINED PARTITION $C_1, C_2, C_3, \ldots.$

TERMINATE WHEN STOP DISCOVERING NEW CLASSES, IN

AT MOST $|Q|$ STEPS.

EXAMPLE

![Diagram](image)

ALGORITHM

STEP 0 $C_1 = \{1, 2, 5, 6\}$ $C_2 = \{3, 4\}$

STEP 1 $C_1 = \{1, 2, 5, 6\}$ $C_2 = \{3\}$ $C_3 = \{4\}$

STEP 2 $C_1 = \{1, 5, 6\}$ $C_2 = \{2\}$ $C_3 = \{3\}$ $C_4 = \{4\}$

STEP 3 $C_1 = \{1\}$ $C_2 = \{2\}$ $C_3 = \{3\}$ $C_4 = \{4\}$

$C_5 = \{5, 6\}$

STEP 4, NO CHANGE — TERMINATE
Constructing $M^1$

1) Get partition $\mathcal{Q} = C_1 \cup C_2 \cup \ldots \cup C_k$ in $M$

2) Construct $M^1$

- **States** $\mathcal{Q}^1 = \{ C_i, C_j, \ldots, C_k \}$

- **Transitions** If $s(p, q) = q'$ in $M$
  Then add $s(C_i, \alpha) = C_j$ to $M^1$
  Where $p \in C_i$, $q' \in C_j$

- **Start State** $C_i$ containing $q_0$ from $M$

- **Final States** Any $C_j$ containing a final state from $M$.

Verify each step works given notion of equivalence.

E.g. in defining transitions, all states in $C_i$ will go to a state in $C_j$ on input $\alpha$ or if $C_j$ contains final state, then all its states are final.

**Example**

![Diagram]

Observe $C_5$ is not reachable from start state and must be removed!

Myhill-Nerode Thm: Above process gives smallest possible DFA for a language — and this is unique.