Let $\Sigma$ be a finite alphabet. Define
$$\text{PAL} = \{ w \in \Sigma^* \mid w^R = w \}.$$

Define $L$ recursively by
a. basis clause: \( \lambda \in L \) and for each \( a \in \Sigma \), \( a \in L \).
b. induction clause: If \( a \in \Sigma \) and \( w \in L \), then \( awa \in L \).
c. extremal clause.

**Theorem.** \( L = \text{PAL} \).

**Lemma.** \( L \subseteq \text{PAL} \).

**Proof.** The proof is by induction corresponding to the recursive definition of $L$.

First some notation, define $P(w)$ on $w^R = w$.

We need to prove: \( \forall w \in L, P(w) \).

**Basis step.** Trivially, $P(\lambda)$ and $P(a)$, for each \( a \in \Sigma \).

**Induction step.** Assume as induction hypothesis that $P(w)$, and prove that $P(awa)$.

Namely,
\[(awa)^R = a^R w^Ra^R = awa.\]
Lemma 2. \( \text{PAL} \subseteq L \).

Proof. The proof is by complete induction on the length of strings in \( L \).

**Basis Step.** Let \( n = 0 \). \( \lambda \) is the only string of length 0. By the basis clause of the definition of \( L \), \( \lambda \in L \).

Let \( n = 1 \). The only words of length 1 are letters \( a \in \Sigma \); by definition, for all \( a \in \Sigma \), \( a \in L \).

So, for all words \( z \) of length 0 or 1,
\[ z \in \text{PAL} \Rightarrow z \in L. \]

**Induction Step.** Let \( n \geq 2 \). Assume as induction hypothesis
\[ \forall w \ ( |w| < n \text{ and } w \in \text{PAL} \Rightarrow w \in L). \]

Let \( |z| = n \) and \( z \in \text{PAL} \). We will show that \( z \in L \). Since \( z \in \text{PAL} \), the first and last letters of \( z \) are identical. So for some letter \( a \in \Sigma \) and some word \( w \),
\[ z = awa. \]

Since \( z^R = z \),
\[ a = (awa)^R = a^R w^R a^R = awa. \]

From \( awa = awa^R \), we conclude that \( w = w^R \).

Thus \( w \in \text{PAL} \). Since \( |w| < n \), the induction hypothesis yields \( w \in L \). Thus, \( z = awa \), where \( w \in L \). So by the induction clause of the definition of \( L \), \( z \in L \).

Thus, \( \text{PAL} \subseteq L \).
Day 4, Fall 2008
Chapter 1.1 Finite Automata

extremely memory limited machine - cannot store its own history.

machine is made from a finite set of objects

time is a discrete number of steps, at each step machine receives an input signal (stimuli). What machine does at time \( t \) (response) depends on its input and current state.

example parity machine

Any number of even ones will cause no net change of state.
This machine has a feeble memory, in that it can distinguish two classes of history - even no. of 1's vs. odd no. of 1's.
Let's add one more feature to the parity machine:

\[ \text{State diagram.} \]

Each node is a state.

The start state (labeled \( q_0 \)) has an arrow pointing to it. There must be a unique start state.

Final states are drawn with a double circle.

Number of edges from each node = number of symbols in \( \Sigma \).

The machine either accepts or rejects. It accepts a word if beginning in the start state, the machine is in a final state when it reaches the end of the word. (Greater formality to follow.)
The language recognized by $M$ is

$$L(M) = \{ w \mid M \text{ accepts } w \}.$$

The parity machine recognizes

$$\{ w \in \{0,1\}^* \mid w \text{ has an odd number of ones} \}.$$ Note: $\lambda \notin L(\text{parity machine})$

Note: Change parity machine as follows:

```
\begin{tikzpicture}
    \node[state] (q0) at (0,0) {$q_0$};
    \node[state] (q1) at (1,0) {$q_1$};
    \node[state] (q2) at (2,0) {$q_2$};
    \node[state] (q3) at (3,0) {$q_3$};
    \node[state] (q4) at (4,0) {$q_4$};
    \draw[->] (q0) edge (q1);
    \draw[->] (q1) edge (q2);
    \draw[->] (q2) edge (q3);
    \draw[->] (q3) edge (q4);
    \draw[->] (q4) edge (q0);
\end{tikzpicture}
```

This recognizes

$$\{ w \in \{0,1\}^* \mid w \text{ has an even number of ones} \}.$$ This machine accepts $\lambda$. 
A finite automaton is a 5-tuple \( M = (Q, \Sigma, \delta, q_0, F) \), where

- \( Q \) is a finite set of states,
- \( \Sigma \) is a finite alphabet
- \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function,
- \( q_0 \in Q \) is the start state
- \( F \subseteq Q \) is the set of accept or final states.

**Parity machine:**

- \( Q = \{ q_0, q_1 \} \)
- \( \Sigma = \{ 0, 1 \} \)

**\( \delta \) given by transition table:**

\[
\begin{array}{c|cc}
\delta & 0 & 1 \\
\hline
q_0 & q_0 & q_1 \\
q_1 & q_1 & q_0 \\
\end{array}
\]

- \( F = \{ q_1 \} \)
Define \( M \) accepts \( w = a_1 \ldots a_n \) if there is a sequence of states \( r_0, \ldots, r_n \) in \( Q \) such that

1. \( r_0 = q_0 \)
2. \( \delta(r_i, a_{i+1}) = r_{i+1} \) for \( i = 0, \ldots, n-1 \), and
3. \( r_n \in F \).

Example:
\[
\delta(q_0, 1) = q_1
\]
\[
\delta(q_1, 0) = q_1
\]
\[
\delta(q_1, 1) = q_0
\]
\[
\delta(q_0, 1) = q_1 \in F
\]

So, parity machine accepts 1011. The sequence of states is \( q_0, q_1, q_1, q_0, q_1 \).
Some operations on languages:

union \( L_1 \cup L_2 \)

intersection \( L_1 \cap L_2 \)

complement \( \Sigma = \Sigma^* - L \)

concatenation
\[
L_1 L_2 = \{ xy \mid x \in L_1 \text{ and } y \in L_2 \}
\]

powers:
\[
L^0 = \{ \lambda \}
L^1 = L
L^{n+1} = L^n L, \quad n \geq 1.
\]

(give example with \( \lambda \) in \( L \))

(For Kleene closure)
\[
L^* = \bigcup_{i=0}^{\infty} L^i
= \{ w_1 \ldots w_k \mid k \geq 0, \ w_i \in L, \text{each } i = 1, \ldots, k \}.
\]

Note: \( \lambda \in L^* \), for all \( L \).
\[
L^+ = \bigcup_{i=1}^{\infty} L^i.
\]
\[ \Sigma^* = \text{set of all words} \]

\[ \phi^* = \{ \lambda \} \]

**Theorem.** \[ S^{**} = S^* \]

**Proof.** \[ S^* \subseteq S^{**}, \text{ because for any language } L, L \subseteq L^* \]

Now show \[ S^{**} \subseteq S^* \]. Let \( w \in S^{**} \).

\( w \) is a concatenation of words in \( S^* \);

\( w = w_1 \cdots w_n \), each \( w_i \in S^* \). Each \( w_i \) is a concatenation of words in \( S \). So,

\( w \) is a concatenation of a concatenation of words in \( S \). So, \( w \) is a concatenation of words in \( S \). So, \( w \in L^* \).

(End Insert)
Study the examples pp 37-40 and the section "Designing Finite Automata."

Work through the examples!

Next, I want to give you a highly structured way to describe certain languages. (See §1.3.)

Let $\Sigma$ be a finite alphabet.

We define regular expressions (§1.26).

Each regular expression describes a language over $\Sigma$.

\[ \Sigma^* \subseteq \{ \lambda, \phi, (, ), \cup, \cdot, * \} \]

**Definition.**

**Basis.** $\alpha$ is a regular expression, for each $\alpha \in \Sigma$.

* is a regular expression.

\( \phi \) is a regular expression.

**Inductive clause.** If $r_1$ and $r_2$ are regular expressions, then so are

\( r_1 \cup r_2 \)

\( (r_1) \)

\( (r_1)^* \). \)
Examples: \( \Sigma = \{a, b\} \)
- \( a \cdot b \)
- \( a + b \)
- \( (a \cdot b) \cdot (a \cdot b) \)
- \( a \cdot (b \cdot a) \)
- \( (a \cdot b)^* \)
- \( a(b^*) \)

Precedence rules for removal of parenthesis increases readability:

(Bad in the order \( * , \cdot , \cup \))
- \( * \) has higher precedence than concatenation or union, and \( \cdot \) has higher precedence than union, \( (((01)^*) \cup 0) \)
becomes
(01)^* \cup 0

But \( ((01)^*) \cup 0 \)
becomes
01^* \cup 0
Each regular expression $\alpha$ over $\Sigma$ denotes a language $L(\alpha)$ as follows:

$L(\varnothing) = \varnothing$, empty language
$L(\lambda) = \{ \lambda \}$

For $a \in \Sigma$, $L(a) = \{ a \}$

If $\alpha_1$ and $\alpha_2$ are regular expressions,

$L(\alpha_1 \cup \alpha_2) = L(\alpha_1) \cup L(\alpha_2)$
$L(\alpha_1 \alpha_2) = L(\alpha_1) L(\alpha_2)$
$L(\alpha_1)^* = L(\alpha_1)^*$

Eg. Let $\alpha = ((a U b)(a U b))^*$.

$L(\alpha)$ contains: $\lambda$, $aa$, $ab$, $aaaa$, $L(\alpha)$ does not contain $abb$

$L(\alpha)$ = set of words of even length.
Now we will use regular expressions to give examples of finite automata and to describe the languages they recognize.

\[(a \cup b)(a \cup b)^*\]

= all words over \(\{a, b\}\) of length \(\geq 1\).

If the start state is also a final state, the FA accepts \(\lambda\).

\[(a \cup b)^*\]

= \(\{a, b\}^*\)