Course Overview

Goal I  Develop abstract, mathematical models of computation, such as:
- Finite State Machines
- Push Down Machines
- Turing Machines

Note physically realizable computers correspond to the simplest such model — F.S.M.

However, rest are useful abstractions of machines we could build in principle, given unlimited resources.

Goal II  Understand properties & power of these models, especially in terms of their ability to solve computational problems.

Q5. A  What problems can they solve at all?
Q5. B  What problems can they solve efficiently?

History

1930s  Alan Turing defined machines more powerful than any in existence, or even any that we could imagine — goal was to establish the boundary between what was and was not computable.

1940s/50s  In an attempt to model “brain function,” researchers defined finite state machines.

Late 1950s  Linguist Noam Chomsky began the study of formal grammars.
To see a convergence of all this into a formal theory of computer science with very deep philosophical implications as well as practical applications (compilers, web searching, hardware, AI, algorithm design, software engineering...)

**Culmination** In 1970s, Steve Cook extended all this to the "theory of NP-completeness" which separated out a class (in fact, most interesting problems in practice) which:

- could be solved, in principle
- but cannot be efficiently solved, even given Moore's law for hardware.

**Hidden Agenda** To teach you how to think precisely and develop powers of reasoning in a precise/formal/abstract fashion.

**Key** This is what separates mere programmers from computer scientists

**Precision is key** Computers/programs are very unforgiving of "fuzzy" thinking.

At the same time, everything you learn will be eminently practical and useful in real life.

Except you may not realize this till a year or two from now.

**Very Important** Review various types of proofs in chapter 1—this is critical to understanding the material presented in class.
SOFTWARE FOR DESIGNING/VERIFYING DIGITAL CIRCUIT.
LEXICAL ANALYZERS OF COMPILERS
SCANNING/SEARCHING LARGE BODIES OF TEXT (WEB SEARCH ENGINES, GREP, NAPSTER, ...)
DESIGN/VERIFICATION/IMPLEMENTATION OF SOFTWARE SYSTEMS INVOLVING INTERACTION (E.G. NETWORK PROTOCOLS, ELECTRONIC COMMERCE, ...)

PROBLEMS?

THIS COURSE WE TAKE THE FOLLOWING VIEW OF A PROBLEM.

LANGUAGE $L$ - SET OF STRINGS
INPUT STRING $x$
PROBLEM DECIDE WHETHER $x \in L$ OR NOT.

MACHINE SOLVES PROBLEM $L$ BY ACCEPTING/REJECTING $x$.

REMARK WHILE REAL-LIFE PROBLEMS ARE OFTEN NOT A SIMPLE LANGUAGE RECOGNITION PROBLEM, WE CAN CAPTURE THEIR ESSENTIAL STRUCTURE IN SOME $L$.

EXAMPLE $L =$ VALID C PROGRAMS

IN COMPILER COURSE YOU WILL SEE THAT A MACHINE WHICH CAN CHECK "LEGALITY" OR "VALIDITY" OF A PROGRAM, CAN BE ADAPTED TO GENERATE "OBJECT CODE" IN THE PROCESS OF DOING SO.
**Review of Basic Definitions**

**Alphabet**  
Any finite set of symbols — \( \Sigma \)

- \( \Sigma = \{0, 1\} \)
- \( \Sigma = \{a, b, c, \ldots, z\} \)
- \( \Sigma = \text{ASCII Characters} \)

**String**  
Finite sequence of symbols from \( \Sigma \)

\( w = w_1 \ldots w_n \) where \( \forall i, w_i \in \Sigma \)

**Examples**

- 1001
- string
- \$1,000,000

**Special Strings**
- \( \emptyset \) Empty string
- \( \# \) Blank or space

**Length**

\( |w| = |w_1 \ldots w_n| = n \)

- \( |\emptyset| = 0 \)
- \( |\#| = 1 \)
- \( |\text{string}| = 6 \)

**Cartesian Product**

\( \Sigma^r = \Sigma \times \Sigma \times \ldots \times \Sigma \)  
\( \text{r times} \)

Means all strings of length \( r \) from \( \Sigma \)

**Defn — Closure**

\( \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \)

Means all finite-length strings from \( \Sigma \)  
But \( \Sigma^* \) itself is infinite.
\[ Z = \varepsilon^3 \]
\[ \Sigma^1 = \Sigma \]

**Example**

\[ \Sigma = \{0, 1\} \]
\[ \Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\} \]

**Concatenation**

\[ x = x_1 \ldots x_n \in \Sigma^* \]
\[ y = y_1 \ldots y_m \in \Sigma^* \]

\[ \Rightarrow x \cdot y = x_1 \ldots x_n y_1 \ldots y_m \]

**Note**

\[ \varepsilon \cdot x = x \varepsilon = x \varepsilon \]

**Note**

\[ |x \cdot y| = |x| + |y| \]

**Language**

\( L \) \text{ any collection of strings} \ L \subseteq \Sigma^*

**Subtle Point**

\( L, \Sigma^* \) \text{ are potentially infinite in size, but contain only finite-length strings}

**Examples**

\[ \left\{ \begin{array}{l}
\Sigma = \{a, b, c, \ldots, z\} \\
L = \text{all English words}
\end{array} \right. \]

\[ \left\{ \begin{array}{l}
\Sigma = \{0, 1\} \\
L = \{\varepsilon, 01, 0011, 000111, \ldots\} \text{ all strings with equal \# of 0's & 1's, but with 0's preceding the 1's.}
\end{array} \right. \]

\[ \left\{ \begin{array}{l}
\Sigma = \text{ASCII} \\
L = \text{compilable C programs}
\end{array} \right. \]
Finite State Machines

Finite Automata

- Simplest model of computation
- Describes class of languages called "regular"
- Operation
  - Always in one of finitely-many states
  - Changes state in response to input
  - Accepts input by ending up in one of so-called final or accepting states

Example
F.A. below scans HTML documents, looking for a list of what could be title-author pairs, perhaps in reading list for some course.

Note
- Accepts when it finds end of a list
- Observe strings that matched the title (before \( \text{by} \) \( \text{by} \)) and author (after \( \text{by} \) \( \text{by} \)) would be stored in a table of such pairs being accumulated.

In HTML

```html
<ol> — Numbered/ordered list
<ol> — Unnumbered/unordered list.
```

Example

```html
<ol>
  <li> Othello \( \text{by} \) \( \text{by} \) Shakespeare <li>
  <li> Foundation \( \text{by} \) \( \text{by} \) Asimov <li>
</ol>
```
NOTATION

STATE TRANSITION DIAGRAM

• STATE 6
• START STATE → 1 (ALSO CALLED INITIAL STATE)
• FINAL STATE → 9 (ALSO CALLED ACCEPTING)
• TRANSITION 4 → B 5

MEANS IN STATE 4, IF THE F.A. SEES "B" IN THE INPUT, IT MOVES TO STATE 5.
**Example**

Consider the problem of checking whether a binary string $w$ contains the pattern 01.

Here

$$
\Sigma = \{0, 1\}
$$

$$
L = \{ \omega \in \Sigma^+ \mid \omega \text{ has substring } 01 \}
$$

$$
= \{ x01y \mid x, y \in \Sigma^+ \}
$$

**Note**

$$
\begin{align*}
11010 & \in L \\
000111 & \in L \\
111000 & \notin L
\end{align*}
$$

**F.A.**

![Finite Automaton Diagram]

**Meaning**

$q_0$: Waiting for first 0

$q_1$: Seen 0, waiting for 1

$q_2$: Seen 01, waiting for end of input.

**Observe**

F.A. scans input $w$ in L-to-R order (cannot back up!), symbol-by-symbol, making state transitions.

**Accepts**

If it is in accepting/final state when it reaches the end of the input.

---

Note

$$
L = \{ \omega \in \Sigma^+ \mid \text{F.A. accepts } \omega \}
$$
To formally define a Finite Automaton (FA), we have:

1) \( \mathcal{Q} \) \text{ Finite Set of States}  
   \[ \text{E.g., } \mathcal{Q} = \{ q_0, q_1, q_2 \} \]

2) \( \Sigma \) \text{ Input Alphabet}  
   \[ \text{E.g., } \Sigma = \{ 0, 1 \} \]

3) \( q_0 \) \text{ Initial or Start State, } q_0 \in \mathcal{Q} 

4) \( F \) \text{ Set of Final or Accepting States, } F \subseteq \mathcal{Q}  
   \[ \text{E.g., } F = \{ q_2 \} \]

5) \( \delta \) \text{ State Transition Function.} 

**Transition Function**  \( \delta : \mathcal{Q} \times \Sigma \rightarrow \mathcal{Q} \)

Here, \( \delta(q_1, 0) = q_2 \) is the same as \( \begin{array}{c}
\begin{array}{c}
\text{Diagram}
\end{array}
\end{array} \)

\( \begin{array}{c}
\begin{array}{c}
\text{Transition Table}
\end{array}
\end{array} \)

\[ \begin{array}{c|cc}
\delta & 0 & 1 \\
\hline
q_0 & q_1 & q_0 \\
q_1 & q_1 & q_2 \\
q_2 & q_2 & q_2 \\
q_0 & q_1 & q_0 \\
\end{array} \]
**Extended Transition Function**

**Goal** Extend \( S \) to multiple transitions

**Idea**
- \( S \): Single transition on input symbol \( a \in \Sigma \)
- \( \hat{S} \): Sequence of transitions on substring \( x \in \Sigma^* \)

**Formally** \( \hat{S} : \mathbb{Q} \times \Sigma^* \rightarrow \mathbb{Q} \)

**Meaning** \( \hat{S}(q, x) = p \) denotes that starting at state \( q \), portion \( x \) of input string will take F.A. to state \( p \).

**Example**

\[
\begin{align*}
\hat{S}(q_0, \text{11}) &= q_0 \\
\hat{S}(q_0, \text{110}) &= q_1 \\
\hat{S}(q_0, \text{1100}) &= q_2 \\
\hat{S}(q_0, \text{1110111}) &= q_2
\end{align*}
\]

\( \hat{S}(p_0, a_1a_2...a_n) = p_n \)

**Question** How do we get \( \hat{S} \) from \( S \)?

**Inductive Defn** \( \forall q \in \mathbb{Q}, \forall a \in \Sigma, \forall x \in \Sigma^* \),

**Basis** \( \hat{S}(q, \epsilon) = q \)

**Induction** \( \hat{S}(q, xa) = \hat{S}(\hat{S}(q, x), a) \)

**Application**

\[
\begin{align*}
\hat{S}(q_0, \epsilon) &= q_0 \\
\hat{S}(q_0, 1) &= S(\hat{S}(q_0, \epsilon), 1) = S(q_0, 1) = q_0 \\
\hat{S}(q_0, 10) &= S(\hat{S}(q_0, 1), 0) = S(q_0, 0) = q_1 \\
\hat{S}(q_0, 101) &= S(\hat{S}(q_0, 10), 1) = S(q_1, 1) = q_2
\end{align*}
\]
**FACT** 
\[ \forall a \in \Sigma, \forall q \in Q, \]
\[ \hat{s}(q, a) = s(q, a) \]

**Proof** 
\[ \hat{s}(q, a) = s(\hat{s}(q, \epsilon), a) \]
\[ = s(q, a). \]

**Thus** \( \hat{s} \) and \( s \) agree on strings of length 1, and \( s \) is only defined for such strings. — **Convention** can call \( \hat{s} \) as \( \hat{s} \) without any confusion.

**Exercise** Prove that
\[ \forall q \in Q, \forall x, y \in \Sigma^*, \]
\[ \hat{s}(q, xy) = \hat{s}(\hat{s}(q, x), y). \]

**Language of F.A. M.**
- \( M = (Q, \Sigma, s, q_0, F) \)
- \( L(M) = \{ \text{all strings accepted by } M \} \)

**Defined**
\[ L(M) = \{ w \in \Sigma^* \mid \hat{s}(q_0, w) \in F \} \]

**Exercise** Consider \( M \) below — what is \( L(M) \)?

![Diagram of a Finite Automaton](image)
**NON-DETERMINISM**

**DETERMINISTIC F.A. (DFA)**

- \[ S(q, a) \text{ is unique (each } q \text{ has exactly one arrow going out for each } a \in \Sigma \) \]
- **Means** for specific input \( w \), execution is totally predictable & repeatable.

**NON-DETERMINISTIC F.A. (NFA)**

- \[ S(q, a) \text{ is a set of states} \]
  - Empty set is possible
  - Multiple states are possible
- Thus \( q \) could have multiple (or no) arrows going out for each \( a \in \Sigma \)
- **Means** multiple choices allows NFA to "guess") the right action, instead of having it hardwired in advance.

**EXAMPLE**

\[ L_{01} = \{ w \mid w \text{ ends in } 01 \} \]

**IDEA**

NFA "guesses" the end of input and then looks for \( 01 \) — using nondeterminism.

**Diagram**

![Diagram of NFA]

**Observe**

Non-determinism implies that instead of unique execution trace (as in DFA's), we have a tree of possible executions.
**Example** for input $w = 0010$

- $q_0 \rightarrow q_1$ (Stuck)
- $q_1 \rightarrow q_0$
- $q_0 \rightarrow q_0$
- $q_0 \rightarrow q_0$
- $q_0 \rightarrow q_1$
- $q_1 \rightarrow q_1$

**But** for $w = 0001$

- $q_0 \rightarrow q_0$
- $q_0 \rightarrow q_0$
- $q_0 \rightarrow q_0$
- $q_0 \rightarrow q_0$
- $q_0 \rightarrow q_1$
- $q_1 \rightarrow \text{(Accept)}$

**Acceptance** when there exists at least one execution path ends in a final state.

**Reselection** when all possible execution paths either get "Stuck" or end in a non-final state.

**Interpretation**

**View 1** It always makes the right choices to ensure acceptance — assuming an accepting path exists.

**View 2** It spawns off multiple copies of itself to explore all possible paths.

**View 3** It explores multiple paths in parallel.
An NFA \( N \) for language \( L \) must ensure:
\[ \forall x \in L, \forall x \in L \] 
all paths are rejecting.
\[ \forall x \in L, \] 
at least one path is accepting.

**Thus** while \( N \) is free to "guess", it must verify its guesses are "correct".

**Formally** \( N = (Q, \Sigma, S, q_0, F) \)
where everything is same as in DFA except \( S \)
\[ S : Q \times \Sigma \rightarrow 2^Q \] 
\( (2^Q : \text{powerset of } Q) \)

That is \( S(q_1, a) \) is subset of \( Q \).

**Above example** \( N_{01} \) has following \( S \)-transition table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>{( q_0, q_1 )}</td>
<td>{( q_0 )}</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( \emptyset )</td>
<td>{( q_1 )}</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

**Extending \( S \) to \( \hat{S} \)**
\[ \hat{S}(q_1, w) : \{ \text{states that can be reached from } q_1 \text{ on input } w \} \]

**Examples**
\[ \hat{S}(q_0, 00) = \{ q_0, q_0, q_1 \} \]
\[ \hat{S}(q_0, 000) = \{ q_0, q_1 \} \]

**Inductive Defn**
\[ \forall q \in Q, \forall x \in \Sigma^*, \forall a \in \Sigma \]

**Basis** \( \hat{S}(q_1, \varepsilon) = \{ q_1 \} \)

**Induction**
Suppose \( \hat{S}(q_1, x) = \{ b_1, b_2, \ldots, b_k \} \)
Suppose \( S(b_i, a) = S_i \) \((\forall i = 1, \ldots, k)\)

**Then** \( S(q_1, wa) = S_1 \cup S_2 \cup \ldots \cup S_k \)
\[ \hat{S}(q_1, x_1) = \bigcup_{p_i \in \hat{S}(q_1, x_1)} S(p_i, a) \]

**Shorthand**

**Example**

- \[ \hat{S}(q_0, 0) = S(q_0, 0) = \{q_0, q_1\} \]
- \[ \hat{S}(q_0, 00) = \bigcup_{p_i \in \hat{S}(q_0, 0)} S(p_i, a) = S(q_0, 0) \cup S(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\} \]
- \[ \hat{S}(q_0, 001) = \bigcup_{p_i \in \hat{S}(q_0, 00)} S(p_i, 1) = S(q_0, 1) \cup S(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\} \]

**Observe** \[ \hat{S}(q_0, 001) \] contains final state \( q_2 \)

\[ \implies 001 \text{ is accepted} \]

**Language of an NFA**

Given NFA \( N = (Q, \Sigma, \delta, q_0, F) \)

Then \[ L(N) = \{ \omega \in \Sigma^* | \hat{S}(q_0, \omega) \cap F \neq \emptyset \} \]

**Verify** this is consistent with earlier explanations.
**Example**  
$L_{123} \subseteq \{1, 2, 3\}^*$

**Language**  
All strings $w \in \{1, 2, 3\}^*$ such that the largest symbol in $w$ appears previously without any intervening larger symbol.

**E.G.**  

```
...11
...2112
...3121213
```

**$N_{123}$**

```
START 1 1
\[ \begin{array}{ccc}
1 & 2 & 1 \\
2 & 9 & 1 \\
3 & 3 & 3
\end{array} \]
```

**In $p$**

HAVEN'T GUESSED YET

**In $s$**

GUSSSED LAST SYMBOL IS A 3, AND JUST SAW THE PRECEDING 3 — WAITING TO VERIFY THAT INTERVENING SYMBOLS ARE LESS THAN 3

**Similarly $01, 97$.**

\[
\begin{array}{c|ccc}
 & 1 & 2 & 3 \\
\hline
p & \{1, 3\} & \{1, 3\} & \{1, 3\} \\
q & \{e\} & \emptyset & \emptyset \\
r & \{e\} & \{e\} & \emptyset \\
s & \{s\} & \{s, s\} & \{s, s\} \\
t & \emptyset & \emptyset & \emptyset \\
\end{array}
\]
Example \( \omega = 31213 \)

\[ \begin{array}{cccccc}
\circ & \rightarrow & \circ & \rightarrow & \circ & \rightarrow \\
\rightarrow & \circ & \rightarrow & \circ & \rightarrow & \circ \\
\rightarrow & \circ & \rightarrow & \circ & \rightarrow & \circ \\
3 & 1 & 2 & 1 & 3 & \text{STUCK}
\end{array} \]

Note \( 8121 \notin L_{123} \).

Comparing the power of DFAs & NFAs

Power? Ability to accept languages

Observe DFA is a special case of NFA with \( |S(9,4)| = 1 \)

\[ \Rightarrow \text{Power (DFA)} \leq \text{Power (NFA)} \]

Theorem For every NFA \( N \) there is a DFA \( M \) which accepts exactly the same language

Thus \( \text{Power (NFA)} = \text{Power (DFA)} \)

Question Then why bother with NFAs, which we can't really implement directly?

Proof of Theorem Converts an NFA \( N \) with \( k \) states into a DFA \( M \) with \( 2^k \) states.

--- Best possible, in that for many NFAs need such a blowup in states when getting DFA.
In general, NFAs are easier to construct, specify, or comprehend due to succinctness. DFAs can be implemented in real-life.

Thus, we use NFAs to capture patterns in string processors (grep, lexical analyzers), but convert to DFAs in finding patterns.

**Theorem.** For every NFA $N$, there exists a DFA $M$ with $L(M) = L(N)$.

**Proof idea.** Given $N$, $M$ will simulate the entire execution tree of $N$ in one execution.

\[
\begin{align*}
\Sigma & = \{0, 1\} \\
\delta_N & = \{ (q_0, q_1) \rightarrow q_0, (q_0) \rightarrow q_0, (q_1) \rightarrow q_1 \} \\
\delta_M & = \{ (q_0) \rightarrow \{q_0, q_1\}, (q_1) \rightarrow \{q_1\}, (q_0, q_1) \rightarrow \{q_0, q_1, q_2\} \}
\end{align*}
\]

**Trick.** A state in $M$ will correspond to a subset of $N$'s states.
FORMALLY

\[ \text{Given} \quad N = (Q_N, \Sigma, \delta_N, q_0, F_N) \]

\[ \text{Construct} \quad M = (Q_M, \Sigma, \delta_M, \{q_0\}, F_M) \]

\[ \text{Such that} \quad Q_M = 2^{\text{ALL SUBSETS OF} \quad Q_N} \]

\[ F_M = \{S \subseteq Q_N \mid S \cap F_N \neq \emptyset\} \]

WHAT ABOUT \( \delta_M \)?

\[ \delta_M (\{p_1, \ldots, p_k\}, a) = \delta_N (p_1, a) \cup \delta_N (p_2, a) \cup \ldots \cup \delta_N (p_k, a) \]

\[ \delta_M (S, a) = \bigcup_{p_i \in S} \delta_N (p_i, a) \]

MEANS \( \delta_M (S, a) \) IS THE SET OF STATES IN \( N \) REACHABLE FROM \( p_i \in S \) ON INPUT SYMBOL \( a \).

EXAMPLE

\[ \xymatrix{ \{q_0\} \ar[r] & \{q_0, q_1, q_2\} \ar[r] & \{q_0, q_1, q_2\} } \]
SOME STATES CAN'T BE REACHED FROM START STATE & HENCE CAN BE ELIMINATED AS BEING NON-ESSENTIAL STATES

DEAD STATE $\emptyset$ CAN NEVER LEAVE IT (STUCK)

SIMPIFIED DFA

![DFA Diagram]

**Lemma**

$\forall q \in Q_N \setminus \emptyset, \forall w \in \Sigma^*$

$\widehat{S}_M (\{q_0\}, w) = \widehat{S}_N (q_0, w)$

**Proof by induction on length of $w$.**

**But First** LET'S FINISH PROOF OF THEOREM.

$L(M) = \{ w \in \Sigma^* \mid \widehat{S}_M (\{q_0\}, w) \in F_M \} = \{ w \in \Sigma^* \mid \widehat{S}_M (\{q_0\}, w) \cap F_N \neq \emptyset \}$ (BY DEFINITION OF $F_M$)

$= \{ w \in \Sigma^* \mid \widehat{S}_N (q_0, w) \cap F_N \neq \emptyset \}$ (BY LEMMA)

$= L(N)$ (BY DEFINITION OF $L(N)$)
**Induction Lemma**

**Induction on Length of \( w \)**

**Basis** \( |w| = 0 \) or \( w = \varepsilon \)

\[
\hat{S}_M (\{q\}, \varepsilon) = \bigcup_{p \in \hat{S}_N (q, \varepsilon)} \hat{S}_N (q, \varepsilon)
\]

**Induction**

Assume lemma for \( |w| = n-1 \)

Show for \( |w| = n \).

**Consider** \( |w| = n \) and write \( w = x \cdot q \)

\( 1x1 = n-1 \), \( 1q1 = 1 \)

**Clearly**

\[
\hat{S}_M (\{q\}, x) = \hat{S}_N (q, x)
\]

(By I.H.)

**Now**

\[
\begin{align*}
\hat{S}_M (\{q\}, w) &= \hat{S}_M (\{q\}, xcq) \quad [w = xcq] \\
&= \hat{S}_M (\hat{S}_M (\{q\}, x), q) \\
&= \hat{S}_M (\hat{S}_N (q, x), q) \quad [\text{Defn of } \hat{S}_M] \\
&= \hat{S}_N (\hat{S}_N (q, x), q) \quad [\text{Defn of } \hat{S}_N] \\
&= \hat{S}_N (q, xq) \quad [\text{Defn of } \hat{S}_N] \\
&= \hat{S}_N (q, \varepsilon) \quad [\text{Defn of } \hat{S}_N] \\
&= \hat{S}_N (q, \varepsilon) \\
&= \hat{S}_N (q, w) \quad \text{Done}
\end{align*}
\]
EXAMPLE

\[ L = \{ \omega \mid \omega \text{ ends in } 001, 010 \text{ or } 100 \} \]

```
EXERCISE  CONVERT TO DFA

EXERCISE  TRY TO WRITE A DFA DIRECTLY.
```

```
OBSERVE  DFA IS QUITE COMPLEX & SECOND EXERCISE IS PRETTY TOUGH

BUT  WRITING NFA IS QUITE EASY & CONVERTING TO DFA IS AN AUTOMATED PROCESS
```