Introduction to the Theory of Computation

What’s a computer? a computation?
Ans1: What current machines are, and what they can do.
  Changes over time. Doesn’t give us LIMITS.
  Ex. Can machines ever......?

What problems are computers capable of solving?
  (Computability)

How easy/hard is a problem?
  (Complexity)

What resources are needed to solve a problem?
  (Computational Models)

CSE 105: Introduction to the Theory of Computation

Instructor: Professor Jeanne Ferrante, CSE
TA’s: Vagelis Hristidis, Min Qian, Dan Liu, Eric Wierwora

Class home page:
http://www-cse.ucsd.edu/classes/fa01/cse105_B

What is the class about?
How will it be run?
Why should you take it?

Finite automata, Ch. 1.1
Reading asst.: Ch 0, 1
Course Organization

Lectures: core material

Discussion sections: Questions

Readings: Text book and web

Exams:
3 scheduled in class, closed book quizzes (60%)
Final exam, Monday, Dec 3 (40%)

3 Homeworks (ungraded, solutions provided)

Class Web Pages: give essential info

WHY SHOULD YOU TAKE THIS COURSE?**>##?

1. It’s fun stuff.

2. It will give you new insights into computers/computation.

3. It will give you knowledge essential to a computer scientist.

4. It will stretch your brain.

5. Its essential for compilers (CSE 131A)
   Regular expressions, finite automata, context-free grammars
A computational model: Finite Automaton (Finite State Machine)

A finite automaton models a computer with no separate memory.

Ex: State diagram of finite automaton that accepts only the key word 'MAIN', alphabet {A-Z}

Kudo Machine

Warning: Accepts only nickels and dimes!

alphabet {5, 10, 15}
What are Finite Automata (FA) good for?

**Recognize patterns in strings**
- e.g. keywords, constants (integers, reals, strings,...),
- variable names, ... in programming languages

**Used in**

*Compilers, lexical analysis phase*
- string of symbols → tokens of language
  - \( \text{const } x = 5 \rightarrow \text{const } x = 5 \)

*Text Editors, ex. EMACS*

*Scripting languages, ex. Perl*

*Unix tools, ex. grep*

---

What are Finite Automata good for?

**Keeping track of limited state information**
- e.g. coin vending machine

**Used in**

*Hardware description languages*
- Ex. VHDL

*Control systems*
- Ex. control refrigerator light, automatic door

*Games Programming*
- Ex. Half-Life, Quake
But first, a break for some definitions....

**Alphabet** \( \Sigma \) : any finite set of symbols  
Ex. \( \{0,1\} \)

**String over** \( \Sigma \) : any finite sequence of symbols  
Exs. \( 01 \quad 0 \quad 111 \)

**Empty string** \( \varepsilon \)

**Concatenation of strings:** Given \( x \) and \( y \), \( xy \)
\( x \varepsilon = \varepsilon \quad x = x \)
Ex. \( x = \) dog, \( y = \) house, \( xy = \) doghouse  
\( x^2 = \) dogdog  
\( x^3 = \) dogdogdog

**Language over** \( \Sigma \) : any set of strings over \( \Sigma \)
Ex. \( \{\varepsilon, 0, 01, 10\} \)
Ex. set of all strings over \( \{0,1\} \)

---

**Formal Defintion of Finite Automaton**

A finite automaton \( M \) is a 5-tuple \( (Q, \Sigma, \delta, q_0, F) \) where,

1. \( Q \) is a finite set of states
2. \( \Sigma \) is a finite set called the alphabet
3. \( \delta : Q \times \Sigma \rightarrow Q \) the transition function
4. \( q_0 \) in \( Q \), the start state
5. \( F \subseteq Q \), set of accept (final) states

The language accepted by \( FA \ M \) is \( L(M) \). We say that \( M \) accepts \( L(M) \).

Previous ex:
\( Q = \{q_0, q_1, q_2, q_3, q_4, q_5\} \quad \Sigma = \{A, B, C, ... Z\} \)
\( \delta(q,a) = ... \quad q_0 \text{ start} \quad F = \{q_4\} \quad L(M) = \)
Formal Definition of Computation on FA

\[ M = (Q, \Sigma, \delta, q_0, F) \]
\[ w = w_1 w_2 w_3 \ldots w_n, \text{ a string over} \]

We say \( M \) accepts \( w \) if there is a sequence of states \( r_0, r_1, \ldots, r_n \) in \( Q \), such that

1. \( r_0 = q_0 \) \hspace{1cm} (starts correctly)
2. \( \delta (r_i, w_{i+1}) = r_{i+1}, \text{ for } i = 0, \ldots, n-1 \) \hspace{1cm} (moves correctly)
3. \( r_n \) is in \( F \) \hspace{1cm} (ends in accept state)

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

A language is **regular** if some FA accepts it.

Some examples of FA

1. \( Q = \{ q_0, q_1 \} \quad \Sigma = \{a, b\} \quad q_0 \text{ start } F = \{ q_1 \} \)
   \[ \delta \text{ given by} \]
   \[
   \begin{array}{c|cc}
   \delta & a & b \\
   \hline
   q_0 & q_0 & q_1 \\
   q_1 & q_1 & q_0 \\
   \end{array}
   \]

2. \( M = (\{ q_1, q_2 \}, \{0,1\}, \ 0, \ q_1, \{ q_2 \}) \quad (\text{Ex. } 1.2, \ p. \ 37) \)
   \[ \delta \text{ given by} \]
   \[
   \begin{array}{c|cc}
   \delta & 0 & 1 \\
   \hline
   q_1 & q_1 & q_2 \\
   q_2 & q_1 & q_2 \\
   \end{array}
   \]

State diagram? \( L(M) = \)

Change accept state of 2?
More examples of FA

3. \( Q = \{ q_0, q_1, q_2 \} \) \( \Sigma = \{ \text{reset}, 0, 1, 2 \} \) (Ex. 1.5, p. 39)

\[
\begin{align*}
q_0 & \xrightarrow{0} q_0 \\
q_0 & \xrightarrow{1} q_1 \\
q_0 & \xrightarrow{2, \text{reset}} q_2 \\
q_1 & \xrightarrow{0} q_0 \\
q_1 & \xrightarrow{1} q_1 \\
q_1 & \xrightarrow{2, \text{reset}} q_2 \\
q_2 & \xrightarrow{1} q_2 \\
q_2 & \xrightarrow{2} q_2
\end{align*}
\]

\( L(M) = \)

4. Construct a FA to accept the language over \{a, b\} which contains an odd number of b's.

What info is needed about input string in order to accept/reject?

Make states of FA to keep track of that info.

5. \( L = \{ w \mid w \text{ does not contain 3 b's as a substring} \} \)

2-bit saturating counter state transition diagram

\[ Q = \{ \text{SPT, WPT, SPN, WPN} \} \quad \Sigma = \{ T, N \} \]

Used in branch prediction

Keep track of last 2 branch outcomes, use to predict next

Final state(s) not important; start state can vary