CPS615 - Theory of Computation

Instructor: Marcus Santos

Lectures: Weds. 14:00 to 16:00, Fri. 12:00 to 13:00

Laboratory: Labs start in the week of January 22. See your schedule.

In Person Office Hours: Fri. 13:00 to 14:00 in VIC 741

Virtual Office Hours: Mon 9:00 to 10:00 @ Google Hangouts (m3santos@ryerson.ca)

Online learning tools: D2L, Gradiance (free web-based tool)

Agenda

- Meet Marcus Santos: background, experience, and interests (related and unrelated to what we will learn in this course)
- Meet You (Homework Assignment 1 - Part 1)
- What I expect from you. Let's have a look at our Course Outline/Course Management Form
- What do you expect from me?
- Theory of computation: why study it, what is it, let's get started on this.

About this course

- In this course we study the theory of what can be computed and what cannot.
- We sketch theoretical frameworks that can inform us the design of programs to solve a wide variety of problems.
- But why bother with theory? Why we don’t just skip ahead and write the programs that we need?

Let’s see if we can provide a convincing answer to these questions.

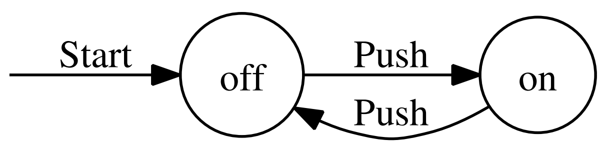
Theory – why study it

- Because there are mathematical properties, both of problems and of algorithms for solving problems, that are independent of the technology or the programming fashion *en vogue* today.
- Most of this theory is from the 70's. But it is still useful for two major reasons:
  - It provides a set of (hardware independent) abstract structures that are useful for solving certain classes of problems.
  - It defines limits to what can be computed, regardless of processor speed or memory size.
- Our focus will be on analyzing problems, rather than comparing solutions to problems. Our goal is to discover fundamental properties intrinsic to the problems themselves.

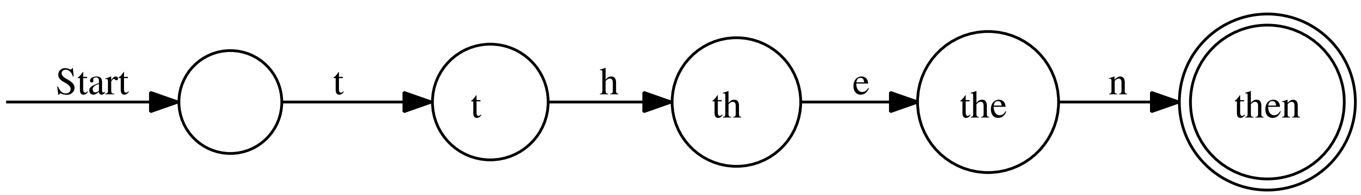
Abstractions: Finite State Machines

- There are many systems that are at all times in one of a finite number of “states”.
- A state is a relevant portion of a system’s history.
- Systems are carefully designed so that they remember what is important and forget what is not.

Example of a finite automaton modelling an on/off switch



Example of a finite automaton that recognizes the string “`then`”



Abstractions: Structural representations

Grammars: useful when specifying processes that handle data with a recursive structure. E.g.: mathematical expressions involving variables x and y

<expr> ::= <expr> <op> <expr>

<expr> ::= <var>

<op> ::= + | - | * | /

<var> ::= x | y

Regular Expressions: useful when specifying text strings. E.g.: (in Unix-Style notation) Any sequence of letters of the Latin alphabet that starts with a capital letter, and ends in a numerical digit.

$$[A - Z][a - z] * [0 - 9]$$

Applications of the Theory are Everywhere

The theory of computation has many applications in the design and construction of important kinds of sofware

- FSMs is used in software for designing digital circuits, and in interactive games.
- The design of programming languanges and compilers
- Natural languages are mostly context-free grammars. Speech understanding systems use probabilistic FSMs.
- Searching for keywords in a file or on the web.
- Verification of communication protocols.
- The theory of intractable problems can help us determine whether we are likely to write a program that solves a given problem, or if we have to find a simplified instance of the problem.

Central concepts

Alphabet: Finite, nonempty set of symbols, e.g.:

- binary alphabet $\Sigma = \{0, 1\}$
- the set of all lower case letters $\Sigma = \{a, b, c, \cdots, z\}$

String: Finite sequence of symbols from an alphabet Σ, e.g.: 001101

Empty String: ϵ denotes the string with zero occurrences of symbols from Σ

Length of String: Number of positions for symbols in the string.

- $|w|$ denotes the length of the string w, e.g., $|0110| = 4, |\epsilon| = 0$

Central concepts (cont.)

Powers of an Alphabet: $\Sigma^k =$ the set of strings of lenght k with symbols from Σ. E.g.: Given $\Sigma = \{0,1\}$, then $\Sigma^1 = \{0,1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$, $\Sigma^0 = \{\epsilon\}$

The set of all strings over Σ is denoted $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$

Also:

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots$$

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$$

Central concepts (cont.)

- **Concatenation**: If x and y are strings, then xy is the string obtained by placing a copy of y immediately after a copy of x.
- Example: $x = 01101, y = 110, xy = 01101110$
- **Powers of a string**: To concatenate a string with itself many times we use the superscript notation

$$x^k \text{ is equivalent to } \overbrace{xx\cdots x}^{k}$$

- Note the difference (in interpretation) between the notation x^k, where x is a string, and Σ^k, where Σ is a set (an alphabet).

And another concept

Languages: If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a language

That is, a language is a set of strings. Examples:

- The set of all legal English words
- The set of all legal C programs
- The set of all binary strings with an equal number of 0's and 1's

$$\{\epsilon, 01, 10, 0011, 0101, 1001, \cdots\}$$

Compact ways to denote languages

It is common to use a "set-former" to denote a language

$$\{w \ : \ \text{something about } w\}$$

which reads: *"the set of words* w *such that..."*

Examples

- $\{w \ : \ w \text{ is a valid English word}\}$
- $\{x01y \ : \ x \text{ and } y \text{ are binary strings or } \epsilon\}$
- $\{0^n1^n \ : \ n \geq 1\}$

Finite Automata (FA)

- FA are simple "machines" that can recognize the first type of languages we will study: regular languages
- A finite automaton has a set of states, and a "control" that moves from state to state in response to external "inputs".
- Let’s create and simulate one in JFLAP
- There are two major classes of automata:

deterministic: on each input, there is only one state to which the automaton can transition from its current state

nondeterministic: on some input, there are more than one state to which the automaton can transition from its current state

Deterministic Finite Automaton (DFA)

A DFA is a quintuple

$$A = (Q, \Sigma, \delta, q_0, F)$$

- Q is a finite set of states
- Σ is a finite alphabet
- δ is a transition function defining the mapping: $Q \times \Sigma \rightarrow Q$
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is a set of final states

How a DFA processes strings

- A DFA is a machine that decides whether or not to accept an input string.
- The language of a DFA is the set of all strings it accepts.
- Given a string $a_1 a_2 \cdots a_n$
  1. We start out at the initial state, q_0
  2. We consult δ to find the state that the DFA enters, say $\delta(q_0, a_1) = q_1$
  3. We process the next input symbol a_2, by evaluating $\delta(q_1, a_2) = q_2$
  4. We continue, finding states $q_3, q_4, \cdots, q_n$, such that $\delta(q_{i-1}, a_i) = q_i$, for each i.
  5. If $q_n \in F$ then the DFA accepts the string; otherwise it "rejects" the string.

Specifying a DFA: example

- Suppose you are asked to specify a DFA that accepts all and only the binary strings that start with a 0 and end with a 1.
- The expression below formally specifies the language L of this particular DFA

$$L = \{0x1 \;:\; x \text{ is either a binary string or } \epsilon\}$$

- Now, let's obtain Σ, Q, δ, q_0 and F.

Example (cont.)

- From the specification of L, we obtain $\Sigma = \{0, 1\}$
- To obtain Q, and δ, we have to identify the possible states (and state transitions) of the scanning process of all strings in L.

$$
\begin{array}{l|l}
\delta(q_0, 0) = q_1 & \delta(q_0, 1) = q_e \\
\delta(q_1, 0) = q_1 & \delta(q_1, 1) = q_2 \\
\delta(q_2, 0) = q_1 & \delta(q_2, 1) = q_2 \\
\delta(q_e, 0) = q_e & \delta(q_e, 1) = q_e
\end{array}
$$

$Q = \{q_0, q_1, q_2, q_e\}$

- Inspect δ to obtain $F = \{q_2\}$

Simpler notations for DFAs

In summary, for the DFA seen in the last slides, we have:

- $Q = \{q_0, q_1, q_2, q_e\}$
- $\Sigma = \{0, 1\}$
- q_0
- $$\begin{array}{l|l} \delta(q_0, 0) = q_1 & \delta(q_0, 1) = q_e \\ \delta(q_1, 0) = q_1 & \delta(q_1, 1) = q_2 \\ \delta(q_2, 0) = q_1 & \delta(q_2, 1) = q_2 \\ \delta(q_e, 0) = q_e & \delta(q_e, 1) = q_e \end{array}$$
- $F = \{q_2\}$

There are simpler ways to describe a DFA by a transition diagram or a transition table.

Exercises

Give DFAs accepting the following languages over the alphabet $\Sigma = \{0, 1\}$

- $L = \{w : w$ contains at least two 0's$\}$
- The set of all strings with three consecutive 0's (not necessarily at the end)

Formally defining the language of a DFA

- The language L of a DFA is the set of all strings it recognizes.
- To formally obtain L, we extend δ to operate on strings.
- We define an *extended transition function* $\hat{\delta}$ that returns the state that an automaton reaches when starting in a given state p and processing a sequence of symbols w. $\hat{\delta}$ is inductively defined as follows:

  Basis: $\hat{\delta}(p, \epsilon) = p$

  Induction: suppose w is a string of the form xa; i.e., a is w's rightmost symbol and x is the rest of the symbols. Then, $\hat{\delta}(p, w) = \delta(\hat{\delta}(p, x), a)$
- Now let's see how $\hat{\delta}$ operates.

On how $\hat{\delta}$ operates

- Given the DFA below, verify that $\hat{\delta}(q_0, 110010) = q_1$
- We compute $\hat{\delta}(q_0, w)$ for each prefix w of 110010.

	0	1
$\to q_0$	q_2	q_0
$\star q_1$	q_1	q_1
q_2	q_2	q_1

$$
\begin{aligned}
\hat{\delta}(q_0, \epsilon) &= q_0 \\
\hat{\delta}(q_0, \epsilon 1) &= \delta(\hat{\delta}(q_0, \epsilon), 1) = \delta(q_0, 1) = q_0 \\
\hat{\delta}(q_0, 11) &= \delta(\hat{\delta}(q_0, 1), 1) = q_0 \\
\hat{\delta}(q_0, 110) &= \delta(\hat{\delta}(q_0, 11), 0) = q_2 \\
\hat{\delta}(q_0, 1100) &= \delta(\hat{\delta}(q_0, 110), 0) = q_2 \\
\hat{\delta}(q_0, 11001) &= \delta(\hat{\delta}(q_0, 1100), 1) = q_1 \\
\hat{\delta}(q_0, 110010) &= \delta(\hat{\delta}(q_0, 11001), 0) = q_1
\end{aligned}
$$

- The language of a DFA $A = (Q, \Sigma, q_0, F)$ is thus defined as

$$L(A) = \{w \ : \ \hat{\delta}(q_0, w) \in F\}$$

Exercise

Given the DFA below, use $\hat{\delta}$ to prove whether the DFAs below accept the string 001110

	0	1
$\rightarrow q_0$	q_2	q_0
$\star q_1$	q_1	q_1
q_2	q_2	q_1

	0	1
$\rightarrow q_0$	q_0	q_1
$\star q_1$	q_1	q_0

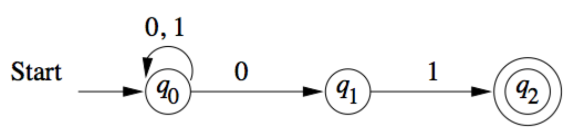
Exercise

Let A be a DFA and a a particular input symbol of A, such that for all states q of A we have $\delta(q, a) = q$.

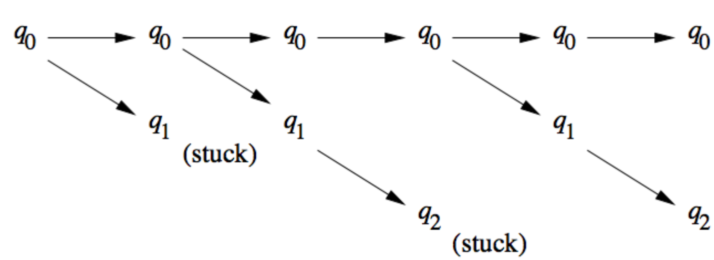
Show by induction on n that for all $n \geq 0, \hat{\delta}(q, a^n) = q$, where a^n is the string consisting of n a's.

Nondeterministic Finite Automata (NFA)

- NFAs are usually easier to "program" in.
- Accepts the same type of language accepted by DFAs (i.e., regular languages)
- An NFA can be in several states at once. Example: An automaton that accepts all and only strings ending in 01.
  - The automaton is able to “guess” when the final 01 has begun.



Here is what happens when the NFA processes the input 00101



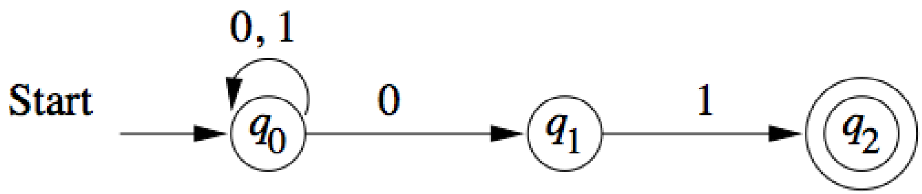
NFA: formally

A NFA is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where

- Q is the set of states
- Σ is the alphabet
- δ is the transition function $Q \times \Sigma \to R : R \subseteq Q$
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final states.

NFA: alternative notations

- State diagram:

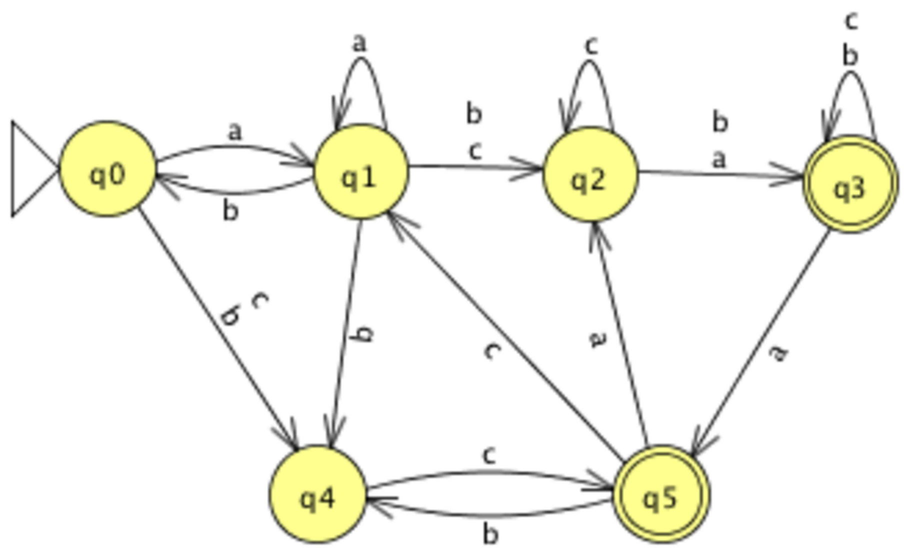


- The quintuple: $(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$
- State transition table:

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$\star q_2$	$\emptyset$	$\emptyset$

Proving if an NFA accepts a string

Suppose we would like to prove whether the automaton below accepts the string $abccb$



It is apparent that we would need a modified version of $\hat{\delta}$ that works for NFAs...

Extended transition function $\hat{\delta}$

We inductively define $\hat{\delta}$ that returns the (set of) states that a NFA reaches as follows.

Basis: $\hat{\delta}(q, \epsilon) = \{q\}$

Induction: Suppose $w = xa$ and $\hat{\delta}(q, x) = \{p_1, p_2, \cdots, p_k\}$.

Let

$$\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \cdots .r_m\}$$

Then $\hat{\delta}(q, w) = \{r_1, r_2, \cdots .r_m\}$

Pseudo code:

$r := \{\ \}$;
for each $p_i \in \{p_1, p_2, \cdots, p_k\}$
$\quad r := r \cup \delta(p_i, a)$
$\hat{\delta}(p, w) = r$;

Formally, the language accepted by A is

$$L(A) = \{w \ : \ \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

Contrasting $\hat{\delta}$ for DFA and NFA

- DFA:

Basis: $\hat{\delta}(p, \epsilon) = p$

Induction: $\hat{\delta}(p, w) = \delta(\hat{\delta}(p, x), a)$

$$L(DFA) = \{w \; : \; \hat{\delta}(q_0, w) \in F\}$$

- NFA:

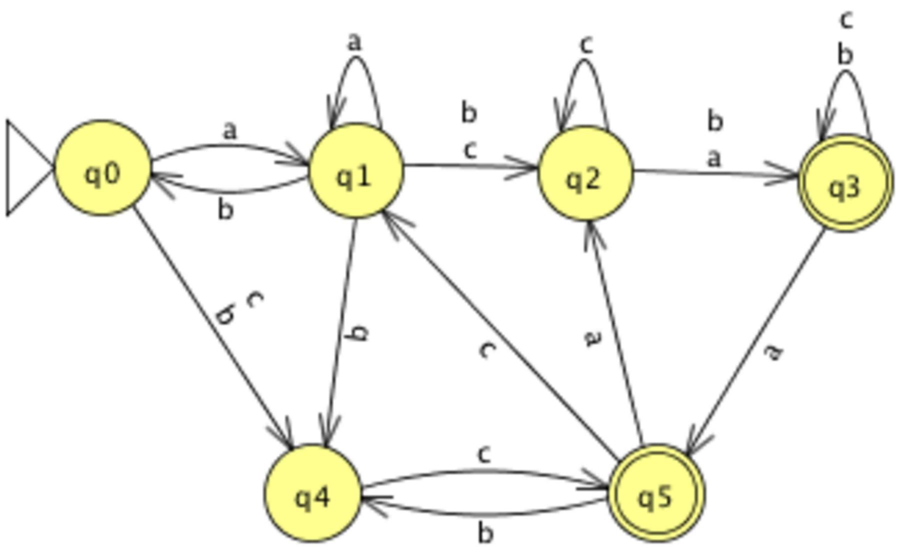
Basis: $\hat{\delta}(q, \epsilon) = \{q\}$

Induction: $\hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a)$

$$L(NFA) = \{w \; : \; \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

$\hat{\delta}$ for NFA: example

Example: Let's compute $\hat{\delta}(q_0, abc)$ for the NFA



(on the board)