Regular Languages

March 23, 2011
CS 81: Computability and Logic
**Recall**

1. Questions about computability can be reduced to *decision problems*.

2. Every *decision problem* can be rephrased as a question about membership in a *language*.
Determining What’s Computable

The Plan:
✓ Define a class of abstract “machines” that accept or reject strings

✓ See what languages this class of machines can recognize (i.e., what decision problems it can solve).

Note:
✓ Every machine corresponds to a language (its accepted strings)
✓ There may be many different machines accepting the same language
✓ If there are restrictions on the machines we can build (e.g., finite size), then not every language may have a machine.
“When the term ‘machine’ is used in ordinary discourse, it tends to evoke an unattractive picture. It brings to mind a big, heavy, complicated object which is noisy, greasy, and metallic; performs jerky repetitive, and monotonous motions; and has sharp edges that may hurt one if he does not maintain sufficient distance…”

Marvin Minsky, Computation: Finite and Infinite Machines
Our First Class of Machines: State Machines

Mathematically, a state machine consists of:

1. an alphabet $\Sigma$
2. a collection of states $Q$
3. a transition relation $\rightarrow \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$
   (where $q \xrightarrow{\sigma} q'$ means that $(q, \sigma, q')$ is in the relation)
4. one initial state $q_0 \in Q$.
5. a set of final/accepting states $F \subseteq Q$.

finite state machine:
$Q$ is finite

deterministic state machine:
a transition function $\delta : Q \times \Sigma \rightarrow Q$. 
Machine Behavior

✓ The machine starts in state $q_0$.
✓ It can change from state $q$ to state $q'$ on input $\sigma$ provided that $q \xrightarrow{\sigma} q'$.
✓ It can change from state $q$ to state $q'$ spontaneously provided that $q \xrightarrow{\epsilon} q'$.
✓ The machine accepts a string $w \in \Sigma^*$ if there is at least one path spelling out $w$, that starts at $q_0$ and ends at a state $\in F$.
What's Accepted?
What’s Accepted?
What’s Accepted? ( bb? b? aaab? ba? )
Finite State Machines

We care mostly about finite state machines, also known as “Finite Automata”

Terminology:

✓ DFA = Deterministic Finite Automaton = Deterministic FSM
✓ NFA = Nondeterministic Finite Automaton = Nondeterministic FSM
The following are equivalent:

✓ There is a DFA accepting the language \( L \)
✓ [Rabin and Scott] There is an NFA accepting \( L \)
✓ [Kleene] \( L \) is a regular set.
Digression: "Scotland Yard," the game
FROM NFA TO DFA: THE Subset Construction

Review
Regular Languages

An inductively-defined collection of sets!

✓ ∅ is a regular language.
✓ {a} is regular for any a ∈ Σ.
✓ If L and M are regular, then so is LM and L U M.
✓ If L is regular, then so is L*.
**Regular Languages**

An *inductively-defined collection of sets!*

✓ $\emptyset$ is a regular language.
✓ $\{a\}$ is regular for any $a \in \Sigma$.
✓ If $L$ and $M$ are regular, then so is $LM$ and $L \cup M$.
✓ If $L$ is regular, then so is $L^*$.

True or False?

1. $\Sigma^*$ is regular.
2. $\{\varepsilon\}$ is regular.
3. If $w \in \Sigma^*$, then $\{w\}$ is regular.
4. Every finite language is regular.
5. Every set is regular (since $\{w_1, w_2, \ldots\} = \{w_1\} \cup \{w_2\} \cup \cdots$).
Regular Expressions

An inductively-defined collection of expressions!

✓ $\emptyset$ is a regexp
✓ $\varepsilon$ is a regexp
✓ $a$ is a regexp for any $a \in \Sigma$.
✓ If $r_1$ and $r_2$ are regexps, then so is $(r_1r_2)$ and $(r_1|r_2)$.
✓ If $r$ is a regexp, then so is $(r^*)$.

Parenthesis Convention:

$$ab^*|c^* = (a(b^*)) | (c^*)$$
REGE M P  IN T ER PR ETAT I O N S

Regular expressions abbreviate regular languages.

✓ $L(\emptyset) = \emptyset$
✓ $L(\varepsilon) = \{\varepsilon\}$
✓ $L(a) = \{a\}$
✓ $L(r_1 r_2) = L(r_1) L(r_2)$
✓ $L(r_1 | r_2) = L(r_1) \cup L(r_2)$
✓ $L(r^*) = L(r)^*$

We say that “$r$ matches $w$” if $w \in L(r)$. True or False?

✓ $L(r_1) = L(r_2) \rightarrow r_1 = r_2$
✓ There is a regular expression $r$ with $L(r) = \Sigma^*$
Regular Expression Examples (\(\Sigma = \{0, 1\}\))

Describe the Language

1. \(0 \| 1\)
2. \((0|1)^*\)
3. \((0|1)\ 0^*\ 1^*\)
4. \(0^*110^*|1^*001^*\)

Find the regular expression

1. Strings where every 1 is followed by a 0.
2. Strings where no 1 is followed by a 0.
3. Strings where every 1 is preceded by and followed by 0.
From Regular Expression to NFA

Construct $\text{NFA}(r)$ by induction/recursion on the regular expression $r$.

✓ $\emptyset$ is a regexp
✓ $\varepsilon$ is a regexp
✓ $a$ is a regexp for any $a \in \Sigma$.
✓ If $r_1$ and $r_2$ are regexps, then so is $(r_1 r_2)$ and $(r_1 | r_2)$.
✓ If $r$ is a regexp, then so is $(r^*)$. 
Completing the Equivalence: Automata to Regular Expressions

Two approaches:

1. Solving equations
2. Generalized NFAs
Let $L_q$ be the set of strings are accepted when starting from state $q$.

✓ What is $L_{q_0}$, $L_{q_1}$, $L_{q_2}$, ...?

✓ How is $L_{q_1}$ related to $L_{q_2}$?
Automaton as a System of Equations

\[ L_A = \varepsilon L_B \cup bL_D \]

\[ L_B = \]

\[ L_C = \]

\[ L_D = \]
Solving Equations using Arden’s Rule

✓ The equation

\[ L = AL \cup B \]

has the solution

\[ L = A^*B \]

✓ This is the smallest solution

- If \( \varepsilon \notin A \), the unique solution
- Otherwise \( A^*C \) is a solution for any \( B \subseteq C \).

\[ \begin{align*}
L_A &= L_B \cup bL_D \\
L_B &= \varepsilon \cup bL_A \cup aL_C \\
L_C &= \varepsilon \cup aL_D \\
L_D &= (a \cup b)L_D \cup bL_C
\end{align*} \]
Generalized NFAs

Just like an NFA, but edges have regular expressions rather than single symbols.

Since regular expressions can be turned into NFAs, we aren’t adding any extra power.
**Regexp by Removing States**

The strategy:

- Make sure our NFA has
  - One start state, with edges only going out
  - One accept state, with edges only going in.
**Regexp by Removing States**

The strategy:

- Make sure our NFA has
  - One start state, with edges only going out
  - One accept state, with edges only going in.
**Regexp by Removing States**

The strategy:

✓ Make sure our NFA has
  ▶ One start state, with edges only going out
  ▶ One accept state, with edges only going in.

✓ Remove all the intermediate states (A–D), one at a time.

✓ In the end, we have one edge, labeled by our regexp.
**Removing States**

✓ When removing state $q$, replace every pair of in/out edges $U_1W_1$, $U_2W_2$ by a single edge $U_1V_1W_2$. 

![Diagram showing edge removal](image)
Example