Linguistics 106, lecture notes
Finite State Languages, Regular Languages

18 July 2002

1 Finite State Automata

1.1 Basic definitions

Finite State Automaton A Finite State Automaton (FSA) is a group of five things: \( (Q, \Sigma, \delta, q_0, \mathcal{F}) \), where:

1. \( Q \) is a finite set of objects called states.
2. \( \Sigma \) is a finite set of symbols, called the alphabet.
3. \( \delta \) is the transition function, a function which takes as input a pair \( (q_i, \sigma) \) \((q_i \in Q, \sigma \in \Sigma)\), and outputs a state \( q_j \) \((q_j \in (Q))\)
4. \( q_0 \) is the unique start state, and \( q_0 \in Q \).
5. \( \mathcal{F} \) is the set of accept states, \( \mathcal{F} \subseteq Q \).

Reading a string An FSA \( M = \{Q_M, \Sigma_M, \delta_M, q_0, \mathcal{F}_M\} \) can read any string over \( \Sigma_M \). To read a string \( \omega = \omega_1 \ldots \omega_n \) is to do the following:

- Let \( r_0 = q_0 \).
- For each consecutive value of \( i, i = 0, 1, \ldots n \), calculate \( r_{i+1}, \) where:
  \[ r_{i+1} = \delta_M(r_i, \omega_{i+1}) \]

Less formally: To read a string is to follow the path defined by the consecutive symbols of that string through the machine, starting with the first symbol and in the start state.

Accepting a string An FSA \( M \) accepts a string iff reading \( \omega \) leaves \( M \) in an accept state (i.e. if \( r_n \in \mathcal{F} \)).

\[ 1^\text{Thus to read a string is to calculate first } r_1 = \delta_M(q_0, \omega_1), \text{ then } r_2 = \delta_M(r_1, \omega_2), \text{ etc.} \]
**Derivation**  An FSA $M$ derives a string $\omega$ iff $M$ accepts $\omega$.

**Language of the machine (L($M$))**  The set of all strings accepted (derived) by an FSA $M$ is the **language** of $M$, $L(M)$.

**Generation**  $M$ generates the language $L(M)$.

**Finite State Language (FSL)**  A language $L$ is a **finite state language** iff there is an FSA which generates it.

**The graphical representation of FSAs:**

- FSAs are often represented as directed graphs, where:
  - Each states is a vertex of the graph,
  - The vertices are drawn as circles with their name ($q_0$, $q_1$, etc.) written in them.
  - Vertices that represent accept states are drawn as double circles, with their name written in them.
  - The edges of the graph are defined by $\delta$. If $\delta(q_x, \sigma_i) = q_z$, then there is an edge $(q_x, q_z)$.
  - The edges are labeled with symbols from $\Sigma$: If $\delta(q_x, \sigma_i) = q_z$, then the edge $(q_x, q_y)$ is labeled $\sigma_i$.

- **Example:**

  **Non-graphical definition of the machine:**
  Let: $M_1 = \{Q_1 = \{q_0, q_1\}, \Sigma_1 = \{a, b\}, \delta_1, q_0, \mathcal{F} = \{q_1\}\}$

  $\begin{align*}
  \begin{array}{c|c}
  (q_x, \sigma_n) & \delta_1 (q_x, \sigma_n) \\
  \hline
  (q_0, a) & q_1 \\
  (q_0, b) & q_0 \\
  (q_1, a) & q_1 \\
  (q_1, b) & q_0 \\
  \end{array}
  \end{align*}$

  **Graphical representation of $M_1$:**

  ![Graphical representation of $M_1$](image)
1.2 Exercise in building FSAs

For each language $L_n$ described in this section, present a deterministic FSA which generates that language.

1. $L_1 = \{ \omega \mid \omega = \epsilon, \text{or } \omega \text{ is just } a's, \text{or } \omega \text{ is just } b's \}$

2. $L_2 = \{ \text{the rednecks like frying squirrels, the hippies like flying squirrels } \}$
   (Assume: $\Sigma_2 = \{ \text{the, rednecks, hippies, like, flying, frying, squirrels } \}$. And in your labeling of the transitions in $M_2$, use “…” to mean ‘all other symbols’,)

3. $L_3 = \{ \text{the man is here, the men are here } \}$
   (Assume: $\Sigma_3 = \{ \text{the, man, men, is, are, here } \}$)

4. $L_4 = L_3$, plus man can be modified by old any number of times.

5. $L_5 = \{ \omega \in \{a,b\}^* \mid \omega = (bab)^* \}$

6. $L_7 = \{ \omega \in \{a,b\}^* \mid \omega = \ldots aba \ldots \}$

1.3 Exercises in interpreting FSAs

For each FSA $M_n$ presented in this section, describe the language of $M_n(L(M_n))$.

1. $M_5$:

   ![Diagram](attachment:diagram.png)
2. $M_9$:

3. $M_{10}$:

4. $M_{11}$:

5. $M_{12}$:

2 Regular Grammars

2.1 Definition of grammar

- Grammars of rewrite rules provide another model for sets of strings (languages), besides automata.
• Formally, a grammar consists of:

  The terminal alphabet \( (V_T) \): A set of symbols.
  The non-terminal alphabet \( (V_N) \): A second, distinct set of symbols.
  The start symbol \( (S) \): A distinguished element of the non-terminal alphabet.
  The rewrite rules \( (R) \): A set of rules of the form: \( X \rightarrow Y \), where: \( X \) and \( Y \) are strings over the union of the terminal and non-terminal alphabets; and in at least one rule, the left-hand side of the rule \( (X) \) is the start symbol.
  The meaning of a rewrite rule \( X \rightarrow Y \) is: “\( X \) can be rewritten as \( Y \).”

• A grammar \( G \) represents a language \( L \) by deriving the strings in \( L \), its sentences.

  Derivation  Grammar \( G \) derives the string \( \omega \) iff the start symbol \( S \) can be rewritten as \( \omega \) through some application of rewrite rules of \( G \).

• The set of all strings derivable by grammar \( G \) is the language of \( G \), \( L(G) \). We will say that a grammar generates language.

Example: Let \( G_0 \) be specified as follows:

  terminal alphabet: \( V_T = \{a, b\} \)
  non-terminal alphabet: \( V_N = \{S, A, B\} \)
  start symbol: \( S = S \)

  set of rewrite rules = \[
  \{ S \rightarrow aB \\
  S \rightarrow bA \\
  A \rightarrow aB \\
  B \rightarrow bA \\
  A \rightarrow a \\
  B \rightarrow b \}
  \]
Sample derivations in $G_0$:

- $S \rightarrow bA$ (Rule: $S \rightarrow bA$)
- $abA \rightarrow baB$ (Rule: $B \rightarrow bA$)
- $aba \rightarrow baba$ (Rule: $A \rightarrow aB$)
- $babab \rightarrow (Rule: B \rightarrow b)$

The same derivations in tree form, a form we will often use:

- $S \rightarrow aB \rightarrow bA \rightarrow aB \rightarrow bA \rightarrow aB \rightarrow b$ (Rule: $S \rightarrow bA$)
- $S \rightarrow bA \rightarrow baB \rightarrow baba$ (Rule: $A \rightarrow aB$)
- $S \rightarrow bA \rightarrow baB \rightarrow baba$ (Rule: $A \rightarrow aB$)
- $S \rightarrow bA \rightarrow baB \rightarrow baba$ (Rule: $B \rightarrow b$)

*Question*: How would you characterize the language generated by $G_0$?

### 2.2 Definition of Regular Grammar

**Regular Grammar** Any grammar each of whose rules has the following three properties:

1. The lefthand side is a **single non-terminal symbol**.
2. The righthand side includes at most a **single non-terminal symbol**.
3. The righthand side includes a **single terminal symbol**.

- Thus any rule in a Regular Grammar (RG) will have one of the following shapes, where $X$ and $Y$ are single non-terminals, and $\sigma$ is a single terminal:

  - $X \rightarrow \sigma$
  - $X \rightarrow \sigma Y$

*Example*: The grammar $G_0$ defined above is a Regular Grammar.

**Regular Language (RL)** Any language which can be generated by a Regular Grammar.
• Interesting equivalence: Any RL is an FSL, and any FSL is an RL.

• This equivalence can be proven by showing that: (i) for any FSA with language $L$, there is a Regular Grammar with language $L$; and (ii) for any RG with language $L$, there is an FSA with language $L$.

A suggestion of how such a proof would proceed is given below, in the two tables.

Given an FSA/RG as specified in the left column, define an RG/FSA as specified in the right column. The resulting RG/FSA will generate the same language as the input FSA/RG—though we will not prove that here,

<table>
<thead>
<tr>
<th>Given FSA</th>
<th>Constructed RG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabet  = $\Sigma$</td>
<td>Terminals = $\Sigma$</td>
</tr>
<tr>
<td>States $\equiv \mathcal{Q}$</td>
<td>Non-terminals = $\mathcal{Q}$</td>
</tr>
<tr>
<td>Start state = $q_0$</td>
<td>Start symbol = $q_0$</td>
</tr>
</tbody>
</table>

| Transitions $\equiv \delta(q_i, \sigma) = q_k$ | Rewrite rules $\equiv q_i \rightarrow \sigma q_k$ |
| $\delta(q_i, \sigma) = q_k$, and $q_k \in \mathcal{F}$ | $q_i \rightarrow \sigma$ |

<table>
<thead>
<tr>
<th>Given RG</th>
<th>Constructed NFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminals = $V_T$</td>
<td>Alphabet = $V_T$</td>
</tr>
<tr>
<td>Start symbol = $S$</td>
<td>Start state = $S$</td>
</tr>
</tbody>
</table>

| Rewrite rules $\equiv X \rightarrow \sigma Y$ | Transitions $\equiv \delta(X, \sigma) = Y$ |
| $X \rightarrow \sigma$ | $\delta(X, \sigma) = Z$, and $Z \in \mathcal{F}$ |

| Accept states $\equiv$ The set all $Z$'s, $Z$ defined as above. | |

Non-terminals = $V_N$ Non-terminals = $V_N \cup \mathcal{F}$

### 2.3 Exercises in designing RGs

For each language $L_n$ in this section, give a Regular Grammar $G$ that generates $L_n$, $L(G) = L_n$.

1. $L_1 = \{ \omega \mid \omega = \epsilon, \text{ or } \omega \text{ is just a's, or } \omega \text{ is just b's } \}$

7
2. $L_3 = \{ \text{the man is here, the men are here} \}$

3. $L_6 = \{ \omega \in \{a, b\}^* | \omega \text{ starts and ends either with } bb \text{ or with } aa \}$

4. $L_7 = \{ \omega \in \{a, b\}^* | \omega = \ldots aba \ldots \}$

3  Regular Expressions

- We can concisely describe any regular language with a regular expression. A regular expression is a compact abstraction of every string in a regular language.
- A regular expression is built with some alphabet $\Sigma$, and three operations:
  - Concatenation ($\circ$): $a \circ b = ab$
  - Union ($\cup$): $a \cup b = \{a \cup b\}$
  - Kleene Star ($\ast$): $x^* = \{x^\ast | \text{x is any string, including } \epsilon, \epsilon x \}$
- We can also abbreviate a sequence of $n$ $X$'s as: $X^n$. For example: $aaaa = a^4$.
- Examples:
  - $L_1 = \{ \epsilon, 1, 11, 111, 1111, \ldots \}$
  - $L_2 = \{ 0, 1 \}$
  - $(ab)^n \circ (ba)^m$ $= (ab)^n(ba)^m = \{ab, ba, abba, abbbba, abbbba, \ldots \}$
  - $0^* \circ 1^* = 0^*1^* = \{\omega | \omega \text{ begins with any number of } 0\text{'s, followed by any number of } 1\text{'s} \}$
  - $(0 \cup 1)^* = \{ \epsilon \text{ all possible strings of } 1\text{'s and } 0\text{'s, in any order, including } \epsilon \}$
  - $\epsilon \cup (a \circ b) = \epsilon \cup ab^* = \{\epsilon, a, ab, abbb, \ldots \}$
  - $bab(10110)^*aba = \{ bababa, bab101aba, bab101101aba, bab1aba, bab110111aba, bab1111011011101aba, \ldots \}$