CISC 4090
Theory of Computation

Finite state machines &
Regular languages

Professor Daniel Leeds
dleeds@fordham.edu
JMH 332

Stereotypical computer

Central processing unit (CPU) – performs all the instructions
Memory – stores data and instructions for CPU
Input – collects information from the world
Output – provides information to the world

Super-simple computers

Small number of potential inputs
Small number of potential outputs/actions

• Thermostat
• Elevator
• Vending machine
• Automatic door

Automatic door

Desired behavior
• Person approaches entryway, door opens
• Person goes through entryway, door stays open
• Person is no longer near entryway, door closes
• Nobody near entryway, door stays closed

Two states: Open, Closed
Two inputs: Front-sensor, Back-sensor

Finite state machine
Graph and table representations

<table>
<thead>
<tr>
<th>Front</th>
<th>Back</th>
<th>Neither</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>Open</td>
<td>Open</td>
<td>Closed</td>
</tr>
<tr>
<td>Open</td>
<td>Open</td>
<td>Open</td>
<td>Closed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

More finite state machine applications

- Text parsing
- Traffic light
- Pac-Man
- Electronic locks

Coding a combination lock

- A finite automaton M1 with 3 states
- Start state q1; accept state q2 (double circle)
- Example accepted string: 1101
- What are all strings that this model will accept?
  
  **String ending with 1 or string end with 1 followed by even number of 0’s**

Formal definition of Finite State Automaton

Finite state automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)

- \(Q\) is a finite set called states
- \(\Sigma\) is a finite set called the alphabet
- \(\delta: Q \times \Sigma \rightarrow Q\) is the transition function
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of accept states
Describe M1 using formal definition

M1 = (Q, Σ, δ, q₀, F)
- Q = \{q₁, q₂, q₃\}
- Σ = \{0, 1\}
- Start state: q₁
- F = \{q₂\}

δ =

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>q₁</td>
<td>q₂</td>
</tr>
<tr>
<td>q₂</td>
<td>q₂</td>
<td>q₂</td>
</tr>
<tr>
<td>q₃</td>
<td>q₂</td>
<td>q₂</td>
</tr>
</tbody>
</table>

Language of M1

If A is set of all strings accepted by M, A is language of M
- L(M) = A

A machine may accept many strings, but only one language
- M accepts a string
- M recognizes a language

Describe L(M1) = A
- A = \{w | w ends with 1 or w end with one 1 followed by even number of 0s\}

Describe M2 using formal definition

M2 = (Q, {0, 1}, δ, q₁, {q₂})
- Q = \{q₁, q₂\}
- Start state: q₁
- δ =

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>q₁</td>
<td>q₂</td>
</tr>
<tr>
<td>q₂</td>
<td>q₂</td>
<td>q₂</td>
</tr>
</tbody>
</table>
What is the language of M2?
L(M2)={w| w ends with at least one 1}

What is the language of M4?
L(M4)={w| w ends and begins with same letter (either a or b)}

CFG practice

G₁ -> AB
A -> xxA | y
B -> wBw | z

G₂ -> D
D -> nD | o
Perform modulo arithmetic

Let \( \Sigma = \{ \text{RESET}, 0, 1, 2 \} \)

Construct M5 to accept a string only if the sum of each input symbol is multiple of 3, and \( \text{RESET} \) sets the sum back to 0
(1.13, page 39)

More modulo arithmetic

Generalize M5 to accept if sum of symbols is a multiple of \( i \) instead of 3

\( \{(q0, q1, q2, q3, \cdots, q_{i-1}), \{0,1,2, \text{RESET}\}, \delta, q0, F\} \)

Definition of M accepting a string

Let \( M = (Q, \Sigma, \delta, q0, F) \) be a finite automaton and let \( w = w_1w_2 \cdots w_n \)

Then \( M \) accepts \( w \) if a sequence of states \( r_0, r_1, \cdots, r_n \) in \( Q \) exists with 3 conditions

- \( r_0 = q0 \)
- \( \delta(r_i, w_{i+1}) = r_{i+1} \) for \( i = 0, 1, \cdots, n - 1 \)
- \( r_n \in F \)
Regular languages

Definition: a language is called a regular language if some finite automaton recognizes it

equivalently

All of the strings in a regular language are accepted by some finite automaton

Designing finite automata (FAs)

• Determine what you need to remember
  • How many states needed for your task?
• Set start and finish states
• Assign transitions
• Check your solution
  • Should accept \( w \in L \)
  • Should reject \( w \notin L \)
  • Be careful about \( \epsilon \)!

FA design practice!

• FA to accept language where number of 1’s is odd (page 43)

• FA to accept string with 001 as substring (page 44)

• FA to accept string with substring abab (next page!)
Regular operations

Let A and B be languages. We define 3 regular operations:
• **Union:** \( A \cup B = \{ x | x \in A \text{ or } x \in B \} \)
• **Concatenation:** \( A \cdot B = \{ xy | x \in A \text{ and } y \in B \} \)
• **Star:** \( A^* = \{ x_1 x_2 \cdots x_k | k \geq 0 \text{ and each } x_i \in A \} \)
  • Repeat a string 0 or more times

Examples of regular operations

Let \( A = \{ \text{good, bad} \} \) and \( B = \{ \text{boy, girl} \} \)

What is:
• \( A \cup B = \{ \text{good, bad, boy, girl} \} \)
• \( A \cdot B = \{ \text{goodboy, goodgirl, badboy, badgirl} \} \)
• \( A^* = \{ \epsilon, \text{good, bad, goodgood, goodbad, badgood, badbad, …} \} \)

Express CFG as RegEx?

\[ G_2 \rightarrow D \]
\[ D \rightarrow nD | o \]

\[ G_1 \rightarrow AB \]
\[ A \rightarrow xxA | y \]
\[ B \rightarrow wBw | z \]

None available

Closure

A collection of objects is closed under an operation if applying that operation to members of the collection returns an object in the collection

Regular languages are closed under \( \cup, \cdot, * \)
Closure of Union

Theorem 1.25: The class of regular languages is closed under the union operation

Proof by construction

Let’s consider two languages
L1: start with 0, end with 1
L2: start with 1, end with 0

Construct machines for each language
Construct machines M3 to recognize L1 U L2

Example union

A = {w | w starts with 0 and ends with 1}
M1

B = {w | w starts with 1 and ends with 0}
M2

Simulate M1 and M2 states
Closure of Union – Proof by Construction

Let us assume $M_1$ recognizes language $L_1$
- Define $M_1$ as $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$

Let us assume $M_2$ recognizes language $L_2$
- Define $M_2$ as $M_2 = (R, \Sigma, \delta_2, r_0, F_2)$

**Proof by construction**: Construct $M_3$ to recognize $L_3 = L_1 \cup L_2$
- Let $M_3$ be defined as $M_3 = (S, \Sigma, \delta_3, s_0, F_3)$

Closure of Union – Proof by Construction

- Let $M_3$ be defined as $M_3 = (S, \Sigma, \delta_3, s_0, F_3)$

Use each state of $M_3$ to simulate being in a state of $M_1$ and another state in $M_2$ simultaneously

$M_3$ states: $S = \{(q_i, r_j) | q_i \in Q \text{ and } r_j \in R\}$

Start state: $s_0 = (q_0, r_0)$

Accept state: $F_3 = \{(q_i, r_j) | q_i \in F_1 \text{ or } r_j \in F_2\}$

Transition function: $\delta_3((q_i, r_j), x) = (\delta_3(q_i, x), \delta_3(q_j, x))$

Closure of Concatenation

Theorem 1.26: The class of regular languages is closed under the concatenation operation

- If $A_1$ and $A_2$ are regular languages, then so is $A_1 \cdot A_2$
- Challenge: How do we know when $M_1$ ends and $M_2$ begins?

Determinism vs. non-determinism

Determinism: Single transition allowed given current state and given input

Non-determinism:
- multiple transitions allowed for current state and given input
- transition permitted for null input $\varepsilon$
NFA in action

• When there is a choice, follow all paths – like cloning
• If there is no forward arrow, path terminates and clone dies (no accept)
• NFA will “accept” if at least one path terminates at accept

Alternative thought:
• Magically pick best path from the set of options

The language of M10

```
• List some accepted strings
  110 – at third entry, we’re in states \{q_1, q_3, \text{and } q_4\}
• What is \(L(M10)\)?
  \{w \mid w \text{ contains 11 or 101}\}
```

NFA construction practice

Build an NFA that accepts all strings over \{0,1\} with 1 in the third position from the end

NFA -> DFA

Build an NFA that accepts all strings over \{0,1\} with 1 in the third position from the end

**Can we construct a DFA for this?**
Formal definition of Nondeterministic Finite Automaton

Similar to DFA: a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)
- \(Q\) is a finite set called states
- \(\Sigma\) is a finite set called the alphabet
- \(\delta: Q \times \Sigma \epsilon \rightarrow \mathcal{P}(Q)\) is the transition function
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of accept states

Describe M10 using formal definition

\[
 M1 = (Q, \Sigma, \delta, q_0, F)
\]

\[
\begin{array}{c|ccc}
\delta & 0 & 1 & \epsilon \\
\hline
q_0 & q_0 & q_1 & F \\
q_1 & q_2 & q_3 \\
q_2 & q_0 & q_1 \\
q_3 & q_2 \\
\end{array}
\]

Consider NFA N1

Language:
\[L(N1) = \{w \mid w \text{ begins with } 0, \text{ ends with } 01, \text{ every } 1 \text{ in } w \text{ is preceded by a } 0\}\]

Convert NFA N1 to DFA M1

\[N1\]
Equivalence of NFAs and DFAs

NFAs and DFAs recognize the same class of languages

Two machines are equivalent if they recognize the same language

Every NFA has an equivalent DFA

Union Closure with NFAs

- Proofs by construction – fewer states!
- Any NFA proof applies to DFA

Given two regular languages $A_1$ and $A_2$ recognized by $N_1$ and $N_2$ respectively, construct $N$ to recognize $A_1 \cup A_2$

Let’s consider two languages

$L_1$: start with 0, end with 1
$L_2$: start with 1, end with 0

Construct machines for each language

Construct machines $N_3$ to recognize $L_1 \cup L_2$

Let’s consider two languages

$L_1$: start with 0, end with 1
$L_2$: start with 1, end with 0

Construct machines $N_1$ and $N_2$ to recognize $L_1$ and $L_2$ respectively.
Closure under concatenation

Given two regular languages $A_1$ and $A_2$ recognized by $N_1$ and $N_2$ respectively, construct $N$ to recognize $A_1 \cdot A_2$.

Closure under star

Prove if $A_1$ is regular, $A_1^*$ is also regular.

Star: $L_1^*$
Closure of regular languages under star

Let \( N_1 = (Q, \Sigma, \delta_1, q_0, F_1) \) recognize \( L_1 \)

\( N_3 = (Q_3, \Sigma, \delta_3, s_0, F_3) \) will recognize \( L_1^* \) iff

\( Q_3 = Q \cup \{s_0\} \)

Start state: \( s_0 \)

\( F_1 \cup \{s_0\} \)

\[ \delta_3(q, a) = \begin{cases} 
\delta_1(q, a) & \text{if } q \in Q \\
q_0 & \text{if } q = s_0 \text{ and } a = \epsilon \\
s_0 & \text{if } q \in F_1 \text{ and } a = \epsilon 
\end{cases} \]

Regular expressions

A regular expression is description of a set of possible strings using a single characters and possibly including regular operations

Examples:
- \((0 \cup 1)^*\)
- \((0 \cup 1)^+\)

Regular expressions – formal definition

\( R \) is a regular expression if \( R \) is
- \( a \), for some \( a \) in alphabet \( \Sigma \)
- \( \epsilon \)
- \( \emptyset \)
- \( R_1 \cup R_2 \), where \( R_1 \) and \( R_2 \) are regular expressions
- \( R_1 \cdot R_2 \), where \( R_1 \) and \( R_2 \) are regular expressions
- \( R_1^* \), where \( R_1 \) is a regular expression

This is a recursive definition

Examples of Regular Expressions
- \( 0^*10^* \)
- \( \Sigma^*1\Sigma^* \)
- \( 01 \cup 10 \)
- \( (0 \cup \epsilon)(1 \cup \epsilon) \)
FA can recognize any Regular Expression

Theorem: A language is regular if and only if some regular expression describes it

• Prove: If a language is described by a regular expression, then it is regular
• Prove: If a language is regular, then it is described by a regular expression

Prove if language described regular expression, it is regular (recognized by FSA)

Each regular expression is either

• Case 1: \( a \in \Sigma \)
• Case 2: \( \varepsilon \)
• Case 3: \( \emptyset \)
• Case 4: \( R_1 \cup R_2 \) – Theorem 1.45
• Case 5: \( R_1 \cdot R_2 \) – Theorem 1.47
• Case 6: \( R_1^* \) – Proven on slide 50

Converting from FSA to Regular Expression