Regular Languages and Finite-State Automata

Formal Languages

- $\Sigma^*$ for the set of all words over $\Sigma$
- Languages = subsets of $\Sigma^*$
- Language-forming operations
  - $L_1 \cup L_2$
  - $L_1 \cdot L_2$
  - $L^*_1$ (Kleene closure)

Regular Expressions

- Inductive definition:
- Base cases:
  - $\emptyset$ is regular expression (over $\Sigma$) and denotes language $\emptyset$
  - $\epsilon$ is regular expression (over $\Sigma$) and denotes language $\{\epsilon\}$
  - $a$ in $\Sigma$ is regular expression (over $\Sigma$) and denotes language $\{a\}$. 
Regular Expressions

- Inductive cases:
  - Assume $r_1$ and $r_2$ are regular expressions and denote $L_1$ and $L_2$, resp.
  - $(r_1 | r_2)$ is a regular expression and denotes $L_1 \cup L_2$.
  - $(r_1 r_2)$ is a regular expression and denotes $L_1 L_2$.
  - $(r_1^*)$ is a regular expression and denotes $L_1^*$.

Regular Expressions Denote Languages

- $a^*$ denotes language $\{a^n | n \geq 0\}$
- $(a).b$ or just $ab$ denotes unit language $\{ab\}$
- $a^*b^*$ denotes $\{a^*b^m | n, m \geq 0\}$
- $a^*b^*$ denotes $\{aabb\}$
  - $a^*$ not regular expression
  - $a^* \ b^*$ denotes $\{a^n b^m | n, m \geq 0\}$
  - $a^*b^*$ denotes $\{bb, abb\}$

Regular Expressions with practical convention

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>alternatives</td>
<td>/[aeiou]/, /m[ae]n/</td>
</tr>
<tr>
<td>-</td>
<td>range</td>
<td>/[a-z]/</td>
</tr>
<tr>
<td>[^ ]</td>
<td>not</td>
<td>/[&quot;pbm]/, /[&quot;ox]s/</td>
</tr>
<tr>
<td>?</td>
<td>optionality</td>
<td>/Kath?mandu/</td>
</tr>
<tr>
<td>*</td>
<td>zero or more</td>
<td>/baa*!/</td>
</tr>
<tr>
<td>+</td>
<td>one or more</td>
<td>/baa+!/</td>
</tr>
<tr>
<td>.</td>
<td>any character</td>
<td>/cat[aeiou]/</td>
</tr>
<tr>
<td>^, $</td>
<td>start, end of line</td>
<td>|?^</td>
</tr>
<tr>
<td>\</td>
<td>not special character</td>
<td>/|?^</td>
</tr>
<tr>
<td></td>
<td></td>
<td>alternate strings</td>
</tr>
</tbody>
</table>
| )         | substring        | /cit\(yies\)/         | etc.
Regular Languages

- $L$ be a language over alphabet $\Sigma$, i.e., $L \subseteq \Sigma^*$. Then $L$ said to be regular language if $L$ denoted by some regular expression over $\Sigma$.
- $\Sigma$ finite alphabet and $L_1$ and $L_2$ regular languages over $\Sigma$. Then $L_1 \cup L_2$, $L_1 \cap L_2$, and $L_1^*$ are also regular.

Remarks

- $\Sigma$ finite alphabet and $w$ any word over $\Sigma$. Then unit language \{w\} regular.
- Any finite language over $\Sigma$ regular.

Deterministic Finite-State Automata

![Deterministic Finite-State Automata](image)
Determinism

• Determinism means that, within any state diagram for FSA, path labeled by given word $w$ unique: for word $w \in \Sigma^*$, there is exactly one path starting at $q_0$ and labeled by $w$.

Transition Functions
Formal Definition

- FSA is quintuple \( \langle \Sigma, Q, q_{init}, F, \delta_M \rangle \)
- \( \Sigma \) is input alphabet
- \( Q \) is finite, nonempty set of states
- \( q_{init} \in Q \) initial state or start state
- \( F \subseteq Q \) is a (possibly empty) set of accepting or terminal states
- \( \delta_M : Q \times \Sigma \rightarrow Q \) transition function (total)

Word Acceptance

- Deterministic finite-state automaton \( M \) accepts word \( w \in \Sigma^* \) if unique path starting at \( q_{init} \) and labeled by \( w \) leads to some member of \( F \), i.e., to some accepting state of \( M \).

Language Acceptance

- The language accepted by \( M \) is the set of all and only those words over \( \Sigma \) that are accepted by \( M \).
- \( L(M) \) for the language accepted by \( M \).
- FSAs are language acceptors only
A Nondeterministic Machine

Figure 9.3.3

Nondeterminism

- $\delta_M: Q \times \Sigma \rightarrow Q$ is a transition mapping
- Assumed to be total ("fully defined") but permitted to be multivalued

Word Acceptance

- Word $w \in \Sigma^*$ accepted by $M$ provided there exists path, labeled by $w$, in the state diagram of $M$ leading from $q_{init}$ to terminal state
- Compare deterministic case
Word Acceptance

- *Word* $w \in \Sigma^*$ accepted by $M$ provided there exists path, labeled by $w$, in the state diagram of $M$ leading from $q_{init}$ to terminal state
- Compare deterministic case

Language Acceptance

- The *language accepted by nondeterministic* is set of words accepted by $M$.

Figure 9.3.4

Nondeterministic Design Often Easier

*Figure 9.3.4*

The nondeterministic finite-state automation: acceptor of regular expression $(ba^*)^*$
Goal

• Suppose given nondeterministic $M$ that accepts $L$. We seek algorithm for constructing, on basis of $M$, a new deterministic $M'$ that accepts $L$.

Subset Construction

• States of nondeterministic $M'$ will correspond to nonempty sets of states of deterministic $M$.
• Where $q_0$ is start state of $M$, use $\{q_0\}$ as start state of $M'$.
• Accepting states of $M'$ will be those state-sets containing at least one accepting state of $M$.

Subset Construction (cont.)

• For each state-set $S$ and for each $s$ in $M'$s alphabet, we draw an arc labeled $s$ from state $S$ to that state-set (call it $S_{s,suc}$) consisting of all and only the $s$-successors of members of $S$.
• Eliminate any state-set, as well as all arcs incident upon it, such that there is no path leading to it from $\{q_0\}$.
Example

- $M$ has 4 states. So $M'$ will have $2^4 - 1$ state sets.
- Terminal state(-sets) marked by bullet
- $a$-arc from state $\{q_0, q_1\}$ will lead to state $\{q_1, q_2, q_3\}$ in new state diagram for $M'$

Theorem (Kleene)

- Let $M$ be nondeterministic FSA accepting $L$. Then there exists deterministic finite-state automaton $M'$ that accepts $L$ as well.

Theorem

- Any finite language is FSA-acceptable
- Example $L = \{abba, abb, abab\}$
Finite-State Automata with $\varepsilon$

- Executing arcs labeled $\varepsilon$ do not advance input
- $\varepsilon$-arcs may or may not introduce nondeterminism

Example

This FSA accepts the language $L(a^*b^*c^*)$.

Figure 9.7.1

Equivalence Result

- Let $M$ be FSA with $\varepsilon$-moves. Then there exists FSA $M'$ with no $\varepsilon$-moves such that $L(M) = L(M')$
Equivalence of FA and RE

- Finite Automata and Regular Expressions are equivalent. To show this:
  - Show we can express a DFA as an equivalent RE
  - Show we can express a RE as an $\varepsilon$-NFA. Since the $\varepsilon$-NFA can be converted to a DFA and the DFA to an NFA, then RE will be equivalent to all the automata we have described.

Turning a DFA into a RE

- Theorem: If $L=L(A)$ for some DFA $A$, then there is a regular expression $R$ such that $L=L(R)$.
- Proof
  - Construct GNFA, Generalized NFA
  - State Elimination
    - We’ll see how to do this next, easier than inductive construction, there is no exponential number of expressions

DFA to RE: State Elimination

- Eliminates states of the automaton and replaces the edges with regular expressions that includes the behavior of the eliminated states.
- Eventually we get down to the situation with just a start and final node, and this is easy to express as a RE
**State Elimination**

- Consider the figure below, which shows a generic state $s$ about to be eliminated. The labels on all edges are regular expressions.
- To remove $s$, we must make labels from each $q_i$ to $p_1$ up to $p_m$ that include the paths we could have made through $s$.

![State Elimination Diagram]

Note: $q$ and $p$ may be the same state!

---

**DFA to RE via State Elimination (1)**

1. Starting with intermediate states and then moving to accepting states, apply the state elimination process to produce an equivalent automaton with regular expression labels on the edges.
   - The result will be a one or two state automaton with a start state and accepting state.

---

**DFA to RE State Elimination (2)**

2. If the two states are different, we will have an automaton that looks like the following:

![DFA to RE Diagram]

We can describe this automaton as: $(R+SU*T)*SU*$
DFA to RE State Elimination (3)

3. If the start state is also an accepting state, then we must also perform a state elimination from the original automaton that gets rid of every state but the start state. This leaves the following:

We can describe this automaton as simply $R^*$. 

DFA to RE State Elimination (4)

4. If there are $n$ accepting states, we must repeat the above steps for each accepting states to get $n$ different regular expressions, $R_1, R_2, \ldots R_n$. For each repeat we turn any other accepting state to non-accepting. The desired regular expression for the automaton is then the union of each of the $n$ regular expressions: $R_1 \cup R_2 \ldots \cup R_n$

DFA $\xrightarrow{\rightarrow}$ RE Example

- Convert the following to a RE

- First convert the edges to RE’s:
DFA $\rightarrow$ RE Example (2)

- Eliminate State 1:

  ![Diagram of DFA with State 1 eliminated]

  - To:
    - Note edge from 3$\rightarrow$3

  Answer: $(0+10)^*11(0+1)^*$

---

Second Example

- Automata that accepts even number of 1’s

  ![Diagram of automata accepting even number of 1’s]

- Eliminate state 2:

  ![Diagram of automata with state 2 eliminated]

---

Second Example (2)

- Two accepting states, turn off state 3 first

  ![Diagram of automata with state 3 turned off]

This is just $0^*$; can ignore going to state 3 since we would “die”
Second Example (3)

• Turn off state 1 second:

![Diagram of automaton]

This is just $0^*10^*1(0+10^*1)^*$

Combine from previous slide to get

$0^* + 0^*10^*1(0+10^*1)^*$

Converting a RE to an Automata

• We have shown we can convert an automata to a RE. To show equivalence we must also go the other direction, convert a RE to an automaton.

• We can do this easiest by converting a RE to an $\varepsilon$-NFA
  – Inductive construction
  – Start with a simple basis, use that to build more complex parts of the NFA

RE to $\varepsilon$-NFA

• Basis:

  $R=\varepsilon$  
  $R=\emptyset$

Next slide: More complex RE's
RE to \( \varepsilon \)-NFA Example

- Convert \( R = (ab + a)^* \) to an NFA
  - We proceed in stages, starting from simple elements and working our way up
    
    - **a**
      
    - **b**
      
    - **ab**

RE to \( \varepsilon \)-NFA Example (2)

- **ab + a**
  
- **(ab + a)^***
What have we shown?

• Regular expressions and finite state automata are really two different ways of expressing the same thing.
• In some cases you may find it easier to start with one and move to the other
  – E.g., the language of an even number of one’s is typically easier to design as a NFA or DFA and then convert it to a RE