Regular Expressions

The *regular expressions* over an alphabet Σ are exactly all strings over the alphabet $\Sigma \cup \{(,), \emptyset, \cup, *\}$ that can be obtained as follows:

1. $\emptyset$, ϵ, and the members of Σ are regular expressions.

2. If R and S are regular expressions, then so is $R \cdot S$ and so is $R \cup S$.

3. If R is a regular expression, then so is R^*.

4. If R is a regular expression, then so is (R).

$*$ has the highest precedence, followed by $\cdot$, followed by $\cup$. Both $\cdot$ and $\cup$ are associative.

Some Abbreviations and Alternatives

- We sometimes write $+$ for $\cup$ (sometimes $|$ is used as well, but we won't use this.)

- We someimtes write RS for $R \cdot S$

- We write R^+ to denote RR^*

Languages Represented by Regular Expressions

Every regular expression represents a language. We define L to be a function which maps any regular expression to a language, as follows:

- $L(\emptyset) = \emptyset$, and $L(a) = \{a\}$ for all $a \in \Sigma$.
- If R and S are regular expressions, then $L(RS) = L(R)L(S)$, and $L(R \cup S) = L(R) \cup L(S)$.
- If R is a regular expression, then $L(R^*) = (L(R))^*$.
- If R is a regular expression, then $L((R)) = L(R)$
- We often don't write the L but just put the E to refer to the language corresponding to E.

Examples of Languages Defined by Regular Expressions

- $\{w \in \{0,1\}^* : w$ ends with an $1\}$.
- $\{w \in \{0,1\}^* : w$ does not contain the substring $10\}$.

Equivalence of FA's and Regular Expressions

- *Theorem: A language is regular $\Leftrightarrow$ if and only if some regular expression describes it.*

- **Proof:** We show each direction separately.

- First, we show that if a language is described by a regular expression R then there is an NFA which recognizes it.

Base Case

- $R = \emptyset$.
- $R = \epsilon$.
- $R = a, \quad a \in \Sigma$.

Induction

Suppose S, T are regular expressions. We assume for induction that there are NFAs M_S, M_T such that $L(S) = L(M_S)$ and $L(T) = L(M_T)$. For each case below, we need to construct a NFA M_R such that $L(M_R) = L(R)$.

1. $R = S \cup T$
2. $R = S \cdot T$
3. $R = S^*$
4. $R = (S)$

Example – Regular Expression to NFA

Consider the regular expression $(ab \cup aba)^*$

1. NFA's for ab and aba:

2. NFA for $(ab \cup aba)$

Example – Regular Expression to NFA

1. NFA for $(ab \cup aba)^*$

Mapping NFA's to Regular Expressions

($\Rightarrow$ We will now show that if a language A is regular, then it is described by a regular expression R such that $L(R) = L(A)$.

Observation: Given an NFA M we can construct an equivalent NFA with:

1. exactly one accept state;
2. no arrows into the start state;
3. no arrows out of the accept state and the accept state is not the same as the start state.

Building a Regular Expression

Given a NFA in the prescribed form, we can create a *generalized non-deterministic FA (GNFA)* $(Q, \Sigma, \delta, q_{start}, q_{accept})$ where

- Edges are labeled by regular expressions (i.e. $\delta : Q \times Q \to \text{RegExp}(\Sigma)$)
- There are no arrows into q_{start} and no arrows out q_{accept} (and there is only one accept state).
- **Note**: Missing arrows are equivalent to arrows labeled by $\emptyset$
- A GNFA accepts a string w if we can write $w = w_1 w_2 w_3 ... w_k$ where $w_i \in \Sigma^*$ and there is a sequence of states $q_0, q_1, q_2, ..., q_k$ such that $w_i \in L(R_i)$ and $R_i = \delta(q_{i-1}, q_i)$ and $q_0 = q_{start}$ and $q_k = q_{accept}$.

What is the language of a GNFA with only two states?

Getting an Equivalent GFNA with Two States

Let $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$ be a GNFA containing at least one state q_t, where $q_t \neq q_{start}, q_{accept}$. We define $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$ as follows

1. $Q' \leftarrow Q - \{q_t\}$

2. For every $q_i \in Q' - \{q_{accept}\}$, $q_j \in Q' - \{q_{start}\}$, let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$$

   where $R_1 = \delta(q_i, q_t)$, $R_2 = \delta(q_t, q_t)$,$R_3 = \delta(q_t, q_j)$, and $R_4 = \delta(q_i, q_j)$.

3. **Recall**: $\emptyset^* = \epsilon$, and for any R, $\epsilon \cup R = R \cup \epsilon = R$ and $\emptyset R = R\emptyset = \emptyset$

$L(G') = L(G)$

Lemma: The language accepted by G' equals the language accepted by G.

Proof:

- $L(G) \subseteq L(G')$: Every string accepted by G along a path which didn't pass through q_t is unaffected. Otherwise, if we remove q_t from an accepting sequence of states in G, we get an accepting sequence of states in G': say we have $\ldots, q_i, q_t, q_t, \ldots, q_t, q_j, \ldots$ in G. By the construction, $\delta'(q_i, q_j)$ will include any substring recognized in this subsequence, so $\ldots, q_i, q_j, \ldots$ will be an accepting computation in G'

- $L(G') \subseteq L(G)$: if G' accepts a string it must have been accepted by G since a label matching a subsequence of a string accepted by G' corresponds to the concatenation of labels on a path in G.

Removal of all Intermediate States

Theorem: The regular expression on the label of the arrow from q_{start} to q_{accept} after all other states have been removed describes the language of the original machine G.

Proof is by induction on the number of states.

Example – NFA to Regular Expression

$N = (Q, \Sigma, \delta, q_1, F)$ where
$Q = \{q_1, q_2\}$
$\Sigma = \{a, b\}$,
$F = \{q_2\}$, and
δ is specified by the following transition table:

q	a	b
q_1	$\{q_1\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$