Regular Languages

Eklavya Sharma

Contents

1 Strings and Languages **2**
1.1 Finite representation of languages . . . 2

2 Regular expressions **3**
2.1 Other closure results . . . 3

3 Finite automata **4**
3.1 DFA . . . 4
3.2 NFA . . . 4
3.3 NFA to DFA . . . 5

4 Finite automata and regular expressions **5**
4.1 Regular expression to finite automaton . . . 5
4.2 Finite automaton to regular expression . . . 5

5 Pumping Lemma **6**

6 DFA minimization **6**
6.1 Similarity of strings w.r.t. a language . . . 6
6.2 Similarity of strings w.r.t. a DFA . . . 6
6.3 Standard DFA of a language . . . 7
6.4 Minimal DFA is isomorphic to standard DFA . . . 7
6.5 DFA minimization . . . 8
6.6 Finding state equivalence classes . . . 9

7 Algorithms for Finite Automata **10**
7.1 NFA to DFA . . . 10
7.2 Regular expression to NFA . . . 10
7.3 NFA to regular expression . . . 10
7.4 DFA minimization . . . 11
7.5 Running an NFA . . . 11

1 Strings and Languages

Basic definitions:

- Alphabet
- String, Empty string (e), Length of a string ($|w|$)
- Concatenation of 2 strings
- Suffix, Prefix, Substring
- Reversal of a string (denoted as w^R)
- Language
- Lexicographical ordering of strings
- Shortlex ordering of strings

Basic theorems:

- **Theorem 1.** $(xy)^R = y^R x^R$

Operations on languages:

- Complement of language L on alphabet Σ: $\overline{L} = \Sigma^* - L$.
- Union and intersection of languages.
- Concatenation of languages: $L_1 L_2 = \{xy : x \in L_1 \land y \in L_2\}$.
- Power of a language: $L^0 = \{e\} \land L^k = L^{k-1}L$.
- Kleene-star of a language L: $L^* = \bigcup\limits_{i \geq 0} L^i$.

1.1 Finite representation of languages

Every string in Σ^* can be mapped to a number. That number is its rank in the shortlex order. Therefore, Σ^* is countably infinite. Since every language is a subset of Σ^*, every language is countable.

The set of all languages is 2^{Σ^*}. It can be proven using diagonalization that there is no bijection from Σ^* to 2^{Σ^*}. Therefore, all languages cannot be represented as a finite string.

2 Regular expressions

Let Σ be an alphabet. A regular expression R is a string over $\Sigma \cup \{\phi, \cup, \texttt{(}, \texttt{)}, ^*\}$ which represents a language over Σ. $L(R)$ is the language represented by R.

- ϕ is a regular expression, where $L(\phi) = \{\}$.
- $\forall a \in \Sigma$, a is a regular expression where $L(a) = \{a\}$.
- If R is a regular expression, then so is $\texttt{(}R\texttt{)}$, where $L(\texttt{(}R\texttt{)}) = L(R)$.
- If R_1 and R_2 are regular expressions, then so is $R_1 \cup R_2$, where $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$.
- If R_1 and R_2 are regular expressions, then so is R_1R_2, where $L(R_1R_2) = L(R_1)L(R_2)$.
- If R is a regular expression, then so is R^*, where $L(R^*) = L(R)^*$.

A language L' is defined to be regular iff there exists a regular expression R such that $L(R) = L'$.

2.1 Other closure results

Theorem 2. *If L is a regular language, then the following are regular:*

- pref(L)*: set of all prefixes of all strings in L.*
- suff(L)*: set of all suffixes of all strings in L.*
- subseq(L)*: set of all subsequences of all strings in L.*
- L^R*: reverse of all strings in L.*

They can all be proven to be regular by finding a regular expression for them.

Let $h : \Sigma \mapsto \Delta^*$ be a function. Extend h to $\Sigma^* \mapsto \Delta^*$ like this:

$$h(e) = e \wedge h(aw) = h(a)h(w) \text{ where } a \in \Sigma \text{ and } w \in \Sigma^*$$

This extended h is called a homomorphism. Also define $h(L) = \{h(w) : w \in L\}$.

Theorem 3. $\forall x, y \in \Sigma^*, h(xy) = h(x)h(y)$

Theorem 4. *If L is regular then $h(L)$ is regular.*

3 Finite automata

3.1 Deterministic Finite Automaton (DFA)

A DFA is a quintuple $M = (Q, \Sigma, \delta, s, F)$ where

- Q is the set of states.
- Σ is the alphabet.
- $\delta : Q \times \Sigma \mapsto Q$ is the transition function.
- $s \in Q$ is the start state.
- $F \subseteq Q$ are the final states.

We can extend δ to $Q \times \Sigma^* \mapsto Q$ like this: $\delta(q, e) = q$ and $\delta(q, aw) = \delta(\delta(q, a), w)$, where $a \in \Sigma$ and $w \in \Sigma^*$. Intuitively, $\delta(q, w)$ is the state reached by running M on w.

Theorem 5 (Can be proved using induction)**.**

$\delta(q, xy) = \delta(\delta(q, x), y)$ *where* $x, y \in \Sigma^*$

We define $L(M)$ as follows: $w \in L(M) \iff \delta(s, w) \in F$.

Automata M_1 and M_2 are equivalent iff $L(M_1) = L(M_2)$.

3.2 Non-deterministic Finite Automaton (NFA)

Let $\Sigma? = \Sigma \cup \{e\}$.

An NFA is a quntuple $M = (Q, \Sigma, \Delta, s, F)$. This is similar to a DFA, except that Δ is a subset of $Q \times (\Sigma?) \times Q$.

$E(q)$ is the set of states reachable from q by following only e transitions. For $K \subseteq Q$, $E(K) = \cup_{q \in K} E(q)$.

$w \in L(M) \iff$ there is a path from s to a final state which consumes w.

Define $\vdash_M$ as:

- $\forall w \in \Sigma^*, (p, w) \vdash_M (q, w) \iff (p, e, q) \in \Delta$.
- $\forall a \in \Sigma, \forall w \in \Sigma^*, (p, aw) \vdash_M (q, w) \iff (p, a, q) \in \Delta$.

Theorem 6. $(p, x) \vdash_M (q, y) \iff (p, xz) \vdash_M (q, yz)$

Define $\vdash_M^*$ as the reflexive-transitive closure of $\vdash_M$.

We define $L(M)$ as follows:

$w \in L(M) \iff (\exists f \in F, (s, w) \vdash_M^* (f, e))$

Theorem 7. *If h is a homomorphism, then $h(\Delta) = \{(p, h(a), q) : (p, h(a), q) \in \Delta\}$ is an NFA accepting $h(L)$.*

3.3 NFA to DFA

Theorem 8. *Let $M = (K, \Sigma, \Delta, s, F)$ be an NFA. Then a DFA $M' = (2^K, \Sigma, \delta, E(s), F')$ exists such that $L(M) = L(M')$ where*

$$F' = \{Q \subseteq K : Q \cap F \neq \{\}\}$$

$$\delta(Q, a) = \bigcup \{E(p) : q \in Q \wedge (q, a, p) \in \Delta\}$$

This can be proven by using this lemma:

Lemma 9.

$$\forall w \in \Sigma^*, \forall p, q \in K,$$

$$(q, w) \vdash_M^* (p, e) \iff (\exists P \supseteq \{p\}, (E(q), w) \vdash_{M'}^* (P, e))$$

This lemma can be proved by induction over $|w|$.

4 Finite automata and regular expressions

4.1 Regular expression to finite automaton

Theorem 10. *Let M, M_1 and M_2 be finite automata. Then it is possible to construct finite automata which accept:*

- $L(M_1) \cup L(M_2)$
- $L(M_1)L(M_2)$
- $L(M)^*$
- $\overline{L(M)}$
- $L(M_1) \cap L(M_2)$

This proves closure properties and the fact that a finite automata can be constructed for a regular language.

Intersection and union of DFAs can be constructed directly by taking the cross-product of states.

4.2 Finite automaton to regular expression

To construct a regular expression for an NFA, we consider a generalization of NFAs, called GNFAs. A GNFA can have an edge which is a regular expression.

Take an NFA and add a new start and final state, so that there is no transition into a start state and a single transition out of the start state and there is a single final state with only a single transition into it and no transition out of it.

Now iteratively remove nodes from the GNFA till only the start and final states remain. The edge between them is labeled with the required regular expression.

5 Pumping Lemma

Theorem 11. *Let L be a regular language. Then there exists a positive integer n, called a 'pumping length' of L, such that*

$$\forall w \in L \cap \Sigma^n\Sigma^*, \exists(x, y, z) \text{ such that}$$

$$w = xyz \wedge y \neq e \wedge |xy| \leq n \wedge (\forall i \geq 0, xy^iz \in L)$$

This can be proven by using the fact that if a DFA has n states, then a string of length $\geq n$ will encounter some state q more than once. The cycle from q to q can be pumped as many times as we want.

The pumping lemma can be used to prove that a language is irregular, by showing that for every pumping length, there is a string which cannot be pumped.

Another way of showing that a language is irregular is by showing that its intersection with a regular language is irregular (This works because regular languages are closed under intersection).

6 DFA minimization

6.1 Similarity of strings w.r.t. a language

Definition 1. *For a language L, $x \approx_L y \iff (\forall z \in \Sigma^*, L(xz) = L(yz))$.*

Theorem 12. *$\approx_L$ is an equivalence relation.*

Definition 2. *$[x]_L$ is the equivalence class of $\approx_L$ to which x belongs. $[\cdot]_L$ is the set of all equivalence classes of $\approx_L$. $|\approx_L|$ is the number of equivalence classes of $\approx_L$.*

Theorem 13. *$x \approx_L y \implies xz \approx_L yz$*

Corollary 13.1. *The function $f([x]_L) = [xz]_L$ is well-defined.*

6.2 Similarity of strings w.r.t. a DFA

Definition 3. *Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA. Then $x \sim_M y \iff \delta(s, x) = \delta(s, y)$.*

Theorem 14. *$\sim_M$ is an equivalence relation.*

Definition 4. *$[x]_M$ is the equivalence class of $\sim_M$ to which x belongs. $[\cdot]_M$ is the set of all equivalence classes of $\sim_M$. $|\sim_M|$ is the number of equivalence classes of $\sim_M$.*

Definition 5. $\delta(s, [x]_M) = \delta(s, x)$. *This definition is consistent because it does not depend on the choice of x because $[x]_M = [y]_M \Rightarrow x \sim_M y \Rightarrow \delta(s, x) = \delta(s, y)$.*

Without loss of generality, assume that all states in Q are reachable.

Theorem 15. *$\delta(s, [\cdot]_M)$ is a bijection from $[\cdot]_M$ to Q.*

Corollary 15.1. $| \sim_M | = |Q|$

Theorem 16. $x \sim_M y \implies xz \sim_M yz$

Theorem 17. $x \sim_M y \implies x \approx_{L(M)} y$

Corollary 17.1. $[x]_M \subseteq [x]_{L(M)}$

Therefore, $[\cdot]_M$ is a finer partitioning of $[\cdot]_{L(M)}$.

Corollary 17.2. $| \approx_{L(M)} | \leq | \sim_M | = |Q|$

6.3 Standard DFA of a language

Let L be a language where $| \approx_L |$ is finite. The standard DFA for L, $M(L) = (Q, \Sigma, \delta, s, F)$ is defined as follows:

- $Q = [\cdot]_L$.
- $s = [e]_L$.
- $F = \{[x]_L : x \in L\}$.
- $\delta([x]_L, a) = [xa]_L$.

δ is well-defined by corollary 13.1. $[x]_L = [y]_L \implies x \approx_L y \implies L(x) = L(y) \implies ([x]_L \in F \iff [y]_L \in F)$. Therefore, F is well-defined.

Lemma 18 (Can be proven using induction). $\delta([x]_L, y) = [xy]_L$

Theorem 19 (Myhill-Nerode theorem (part 1)). $L(M(L)) = L$

Proof. $x \in L(M(L)) \iff \delta([e]_L, x) \in F \iff [x]_L \in F \iff x \in L$ □

Theorem 20 (Myhill-Nerode theorem (part 2)). *L is regular iff $| \approx_L |$ is finite.*

Proof. If L is regular, there is a DFA M which accepts L. Therefore, $| \approx_L | \leq | \sim_M |$, so $| \approx_L |$ is finite. Conversely, $| \approx_L |$ is finite implies $M(L)$ is a DFA which accepts L, so L is regular. □

For any DFA M which decides L, $| \approx_L | \leq | \sim_M | = |Q|$. Since $M(L)$ has exactly $| \approx_L |$ states, it is a DFA with the minimum number of states.

6.4 Minimal DFA is isomorphic to standard DFA

Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA of L with $| \approx_L |$ states. Let $M' = (Q' = [\cdot]_L, \Sigma, \delta', s' = [e]_L, F')$ be the standard DFA of L.

We will prove that M is isomorphic to M', which means that there is a bijection $\phi : Q \mapsto Q'$ such that all of the following conditions are satisfied:

- $\phi(s) = s'$
- $\phi(F) = \{\phi(f) : f \in F\} = F'$
- $\forall q \in Q, \delta'(\phi(q), a) = \phi(\delta(q, a))$

$\sim_M$ is a refinement of $\approx_L$ and $| \sim_M | = |Q| = | \approx_L |$. Therefore, $\sim_M = \approx_L$. Therefore, $\forall x \in \Sigma^*, [x]_M = [x]_L$. Therefore, we will just write $[x]$.

Let ϕ be the inverse of $\delta(s, [\cdot]_M)$ (This means $\phi(q) = [x] \iff \delta(s, x) = q$). Therefore, ϕ is a bijection.

$$\delta(s, e) = s \implies \delta(s, [e]_M) = s \implies \phi(s) = [e]_M = [e]_L = s'$$

$$\begin{aligned}
&[x] \in F' \\
&\iff x \in L \\
&\iff \exists f \in F, \delta(s, x) = f \\
&\iff \exists f \in F, \delta(s, [x]) = f \\
&\iff \exists f \in F, \phi(f) = [x] \\
&\iff [x] \in \phi(F)
\end{aligned}$$

Therefore, $\phi(F) = F'$.

Let $\phi(p) = [x]$, $\phi(q) = [y]$.

$$\begin{aligned}
&\delta(p, a) = q \\
&\Rightarrow \delta(\delta(s, x), a) = \delta(s, y) \\
&\Rightarrow \delta(s, xa) = \delta(s, y) \\
&\Rightarrow xa \sim_M y \\
&\Rightarrow [xa] = [y] \\
&\Rightarrow \delta'([x], a) = [y] \\
&\Rightarrow \delta'(\phi(p), a) = \phi(q) = \phi(\delta(p, a))
\end{aligned}$$

Therefore, M is isomorphic to M'.

6.5 DFA minimization

Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA. Without loss of generality, assume all states in Q are reachable.

Definition 6. *For $p, q \in Q$,*

$$p \equiv q \iff (\forall z \in \Sigma^*, \delta(p, z) \in F \iff \delta(q, z) \in F)$$

Theorem 21. *By taking $|z| = 0$, we get $p \equiv q \implies (p \in F \iff q \in F)$.*

Theorem 22. *$\equiv$ is an equivalence relation.*

Definition 7. *For $q \in Q$, $[q]$ is the equivalence class of $\equiv$ containing q.*

Theorem 23. $\delta(s, x) \equiv \delta(s, y) \iff x \approx_{L(M)} y$

Therefore, if we clump together all states which are equivalent, then all strings reaching that clumped state are $\approx_L$-equivalent and all strings which are $\approx_L$-equivalent will reach the same clumped state. Therefore, the clumped states correspond exactly to the equivalence classes of $\approx_L$. Therefore, $|\equiv| = |\approx_L|$.

We will now construct a DFA $M' = (Q', \Sigma, \delta', s', F')$ which has exactly $|\equiv|$ states:

- $Q' = [\cdot]_\equiv$
- $s' = [s]$
- $F' = \{[f] : f \in F\}$
- $\delta'([q], a) = [\delta(q, a)]$

Theorem 24. $f \in F \iff [f] \in F'$

Therefore, F' is well-defined.

Theorem 25. $\forall w \in \Sigma^*, p \equiv q \implies \delta(p, w) \equiv \delta(q, w)$

Corollary 25.1. $\forall a \in \Sigma, p \equiv q \implies \delta(p, a) \equiv \delta(q, a)$

Therefore, δ' is well-defined.

Theorem 26. $\delta'([q], w) = [\delta(q, w)]$

Theorem 27. $x \in L(M') \iff x \in L(M)$

6.6 Finding state equivalence classes

Definition 8. *For* $p, q \in Q$,

$$p \equiv_n q \iff (\forall |z| \le n, \delta(p, z) \in F \iff \delta(q, z) \in F)$$

Theorem 28. *For* $|z| = 0$, *we get* $p \equiv q \iff (p \in F \iff q \in F)$.

Therefore, equivalence classes of $\equiv_0$ are F and $Q - F$.

Theorem 29. $\forall n \ge 0, p \equiv q \Rightarrow p \equiv_n q$

Theorem 30. $\forall i < j, p \equiv_j q \Rightarrow p \equiv_i q$

Therefore, increasing n refines $\equiv_n$ and $\equiv$ is finer than all $\equiv_n$.

Theorem 31. $(\forall a \in \Sigma, \delta(p, a) \equiv_{n-1} \delta(q, a))$
$\iff (\forall 1 \le |z| \le n, \delta(p, z) \in F \iff \delta(q, z) \in F)$

Theorem 32. $p \equiv_n q \iff (p \equiv_{n-1} q \wedge (\forall a \in \Sigma, \delta(p, a) \equiv_{n-1} \delta(q, a)))$

Theorem 33. $(\forall p, q \in Q, p \equiv_n q \iff p \equiv_{n+1} q)$
$\implies (\forall p, q \in Q, p \equiv_{n+1} q \iff p \equiv_{n+2} q)$

Since $\equiv_{n+1}$ is a refinement of $\equiv_n$, $(\equiv_n) \neq (\equiv_{n+1}) \implies |\equiv_n| < |\equiv_{n+1}|$. Since $|\equiv_n| \le |Q|$, there will be a value of n such that $(\equiv_n) = (\equiv_{n+1}) = (\equiv_{n+2}) = \ldots = (\equiv)$.

7 Algorithms for Finite Automata

7.1 NFA to DFA

Let $M = (Q, \Sigma, \Delta, s, F)$ be an NFA and $M' = (Q', \Sigma, \delta', E(s), F')$ be the corresponding DFA, where $Q' \subseteq 2^Q$, $F' = \{K \subseteq Q : K \cap F \neq \{\}\}$ and $\delta'(K, a) = E\left(\bigcup_{q \in K} \Delta(q, a)\right)$ where $\Delta(q, a) = \{p : (q, a, p) \in \Delta\}$.

It is assumed that intersection or union of 2 subsets of Q can be computed in $O(|Q|)$ time. This is because both sets can be expressed as $|Q|$-length bitsets.

- Calculate $E(q)$ for all $q \in Q$: This can be done with a DFS from every node. Time complexity: $O(|Q|^2)$.
- Calculate $\Delta(q, a)$ for all $(q, a) \in Q \times \Sigma$: Create a table of size $|Q||\Sigma|$, each cell of which stores an empty list. For each $(q, a, p) \in \Delta$, add p to cell at (q, a). Time complexity: $O(|Q||\Sigma| + |\Delta|)$.
- Calculate $\delta'(K, a)$ for all $(K, a) \in Q' \times \Sigma$: For each (K, a), take the union of $|K|$ sets and then $|Q|$ sets. Time complexity: $O(|Q'||\Sigma||Q|^2)$.
- Calculate F': Intersect each state in Q' with F. Time complexity: $O(|Q'||Q|)$.

Since $\Delta \subseteq Q \times \Sigma \times Q$, $|\Delta| \leq |Q|^2|\Sigma|$.

Therefore, total time complexity is $O(|Q'||Q|^2|\Sigma|)$, where $|Q'| \leq 2^{|Q|}$.

7.2 Regular expression to NFA

The NFA for a regular expression R will have at most $2|R|$ states and at most $4|R|^2$ transitions. The cost of each operation is proportional to the number of states and transitions added. Therefore, running time is $O(|R|^2)$.

7.3 NFA to regular expression

If each edge on a GNFA has length at most l, removing a vertex will change the length to at most $4l + 10$. Therefore, an NFA with n states will have a regular expression of length at most $(10n + 1)4^n$.

Time taken by naive algorithm, which just concatenates regular expressions:

$$O\left(\sum_{i=1}^{n}(n + 2 - i)^2 4^i (10i + 1)\right) = O(n^3 4^n)$$

Space complexity of this algorithm: $O(n^3 4^n)$.

Concatenation of 2 strings takes time proportional to the length of the 2 strings. Therefore, concatenating multiple strings by repeated pairwise concatenation leads to unnecessarily repeated work. We can optimize this by using lazy concatenation: Instead of producing a new string object, create a 'lazy-string' object, which just stores references to the strings being concatenated. Doing this repeatedly will give us a concatenation tree (actually a DAG, but we can safely ignore this detail). To convert a lazy string tree to a non-lazy string, do an in-order traversal of the tree and print each leaf node.

A lazy-string tree is a full binary tree. Therefore, the number of nodes (after expanding the DAG to a tree) is equal to twice the number of leaf nodes. Therefore, time taken for traversal is linearly proportional to the size of the output.

Creating a lazy string from an NFA takes time $O(n^3)$. Converting that lazy string to a non-lazy string takes time $O(n4^n)$. Therefore, time taken by this algorithm is $O(n4^n)$. Since the DAG has at most $O(n^3)$ entries, space complexity is $O(n^3)$.

7.4 DFA minimization

The algorithm proceeds in phases. In each phase, we compute $\equiv_i$ from $\equiv_{i-1}$. There can be at most $|Q| - 1$ phases. To compute $\equiv_i$ from $\equiv_{i-1}$, a crude algorithm would consider every pair in every set in $\equiv_{i-1}$ and it will check in $O(|\Sigma|)$ time whether elements of that pair are $\equiv_i$-equivalent. Therefore, a rough upper bound on running time is $O(|\Sigma||Q|^3)$.

A better algorithm: to split a set, take the first element and find out which elements are similar to it. Move all those elements to a different set and repeat. Running time: $O(|\Sigma||Q|^2)$.

Hopcroft invented an $O(|\Sigma||Q| \log |Q|)$-time algorithm.

7.5 Running an NFA

The algorithm for running an NFA is similar to the algorithm for NFA to DFA construction. We keep track of the set of states the NFA can be in. Running time is $O(n|Q|^2)$, where n is the length of the input string.