Lecture 5: Myhill - Nerode Theorem

Instructor: Ketan Mulmuley
Scriber: Yuan Li

January 20, 2015

1 Myhill - Nerode Theorem

Recall the theorem we have stated in the last class, and we will give a proof in this lecture. Equivalence relation $\sim$ on $\Sigma^* \times \Sigma^*$ is called *right-invariant* if

$$x \sim y \Rightarrow xw \sim yw \;\; \forall w \in \Sigma^*.$$

Theorem 1.1. *The followings are equivalent:*

(1) $L \subseteq \Sigma^*$ is accepted by some DFA (i.e., L is regular).

(2) L is the union of some equivalence classes of a right-invariant equivalence relation on $\Sigma^* \times \Sigma^*$ of finite index. (We say equivalent relation is of finite index if the number of equivalence classes is finite.)

(3) Define $x \sim_L y$ if and only if $(\forall z \in \Sigma^*)(xz \in L$ iff $yz \in L)$. Then $\sim_L$ has finite index.

Proof. (1) $\Rightarrow$ (2). Say $L = L(M)$, where $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA. Define equivalence relation $\sim_M$ as: $x \sim_M y$ if and only if x and y take q_0 to the same state. It is easy to see $\sim_M$ is a right-invariant equivalence relation, and L is the union of equivalence classes of $\sim_M$ corresponding to the states in F.

(2) $\Rightarrow$ (3). Fix a right-invariant equivalence relation R of finite index such that L is the union of some equivalence classes of R. We want to show if $x \sim_R y$, then $x \sim_L y$, which will imply $\sim_L$ has finite index. Fix some $x \sim_R y$. For any $z \in \Sigma^*$, if $xz \in L$, then xz belongs to some equivalence class C of

R. Since R is right-invariant, we claim yz belongs to the same class C of R, which implies $yz \in L$.

In words, equivalent classes of $\sim_L$ are obtained by coalescing some equivalent classes in R.

(3) $\Rightarrow$ (1). Assign a state to each equivalence class C (finitely many). Let $\delta(C, a) = D$, where $a \in \Sigma$, and D is the equivalence class of xa for any x in class C. (The choice of x does *not* matter.) Let DFA $M = (Q, \Sigma, q_0, \sigma, F)$, where Q is the set of all equivalence classes, F is the set of all equivalence classes contained in L. □

By (3) in the above theorem, we know there exists a DFA accepting L with m states, where m is the number of equivalence classes of $\sim_L$ as defined in (3). And for any DFA M such that $L(M) = L$, M induces a equivalence relation which is a refinement of $\sim_L$, and thus M has at least m states. The uniqueness of the minimal DFA follows from the uniqueness of the minimal equivalence relation refining $\sim_L$, that is, $\sim_L$ itself.

Now we prove that, given any regular language L, there exists a *unique* minimal DFA M such that $L(M) = L$. The next problem is to find such M efficiently (= in polynomial time). This problem, in my view, is the first nontrivial problem we have encountered so far.

2 Minimization of DFA

Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, we want to construct the (unique) minimal DFA N such that $L(M) = L(N)$.

Let us maintain a table $L(p, q)$, where $p, q \in Q$. Put a cross at $L(p, q)$ if states p and q are distinguishable. Initially, $L(p, q) = \times$ for all ($p \in F$ and $q \notin F$) or ($p \notin F$ and $q \in F$). Then, for every location (p, q) without a cross, for every $a \in \Sigma$, if $L(\delta(p, a), \delta(q, a)) = \times$, then put a cross at $L(p, q)$. Repeat this process until there is no updates.

Let us prove the correctness of this algorithm. If $L(p, q) = \times$, then it is clear that p, q belongs to different equivalent classes of $\sim_L$. On the other hand, we need to prove if p and q are not equivalent, i.e., p, q belongs to different equivalent classes, then $L(p, q) = \times$. Since p, q are not equivalent, there exists some $x \in \Sigma^*$ such that px goes to an accepting state, and qx goes to a non-accepting state (or px goes to a non-accepting state, and qx goes to an accepting state). Consider such x with *minimal* length. We can prove by induction on the length of x, which is left to the readers as an exercise.

For the ruining time of the algorithm, each iteration takes $O(|Q|^2|\Sigma|)$ steps, and there could be at most $|Q|^2$ iterations (because there are $|Q|^2$ entries, and each time at least one entry will be updated). Putting everything together, the total running time is $O(|Q|^4|\Sigma|) = O(|Q|^4)$ assuming $|\Sigma|$ is a constant.

Exercise 2.1. *Improve the above algorithm to reduce the running time to* $O(n^2)$*, where* n *is the input size.*

Theorem 2.2. *Given a DFA* N*, the smallest* M *such that* $L(M) = L(N)$ *can be constructed in time polynomial in the (specification) size of* N*.*

3 Context-Free Languages

Consider the following example, which is the definition of an expression in some programming language.

- $\langle exp \rangle \to \langle exp \rangle + \langle exp \rangle$
- $\langle exp \rangle \to \langle exp \rangle - \langle exp \rangle$
- $\langle exp \rangle \to \langle exp \rangle * \langle exp \rangle$
- $\langle exp \rangle \to (\langle exp \rangle)$
- $\langle exp \rangle \to$ identifier (variable or constant)

For example, $exp := (x + y) * z$ is a valid expression, which has the following derivation tree.

Formally, a Context-Free Grammar (CFG) is $G = (V, T, P, S)$, where V is the set of vertices, T the set of terminals, P the set of production rules and S the start symbol. Each production rule is of the form $A \to \alpha \in (V \cup T)^*$, where $A \in V$. For the above example, $V = \{E\}$, $S = \{E\}$, $T = \{\text{identifier}\}$, and

$$P = \{E \to E + E \mid E - E \mid E * E \mid (E) \mid \text{id}\}.$$

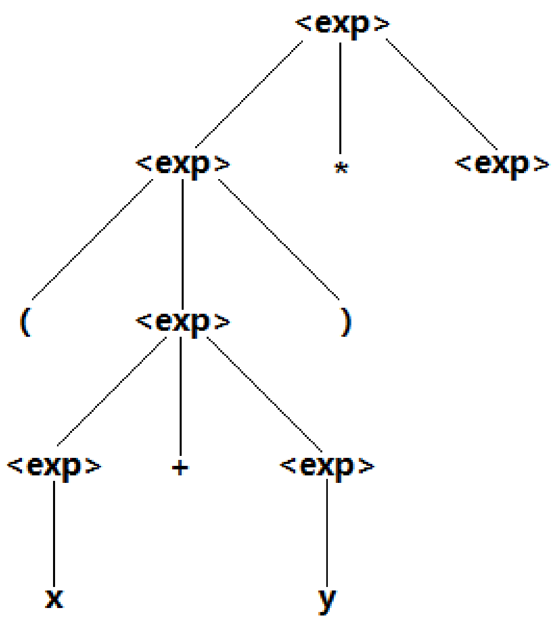

Figure 1: Derivation tree for $(x + y) * z$.

If $A \Rightarrow_G B$ is a production, then $xAy \Rightarrow_G xBy$, for any $x, y \in (V \cup T)^*$. Define $\Rightarrow_G^*$ to be the reflexive transitive closure of $\Rightarrow_G$. Then,

$$L(G) = \{w \in T^* : S \Rightarrow_G^* w\}.$$

String $\alpha \in (V \cup T)^*$ is a *sentential form* if $S \Rightarrow_G^* \alpha$.

Let $L \subseteq \{0, 1\}^*$ be the set of strings containing equal number of 0's and 1's. We will show L is context-free language. Let $G = (V, T, P, S)$, where $V = \{S, A, B\}$, $T = \{0, 1\}$. The production rules are:

- $S \to 0A \mid 1B$
- $A \to 1 \mid 1S \mid 0AA$
- $B \to 0 \mid 0S \mid 1BB$

S represents the string with equal number of 0's and 1's, A represents the string with one more 1, and B represents the string with one more 0.