2.17 Right-Invariant Equivalence Relations on Σ^*

Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA. The DFA D may be redundant, for example, if there are states that are not accessible from the start state.

The set Q_r of *accessible or reachable states* is the subset of Q defined as

$$Q_r = \{p \in Q \mid \exists w \in \Sigma^*,\ \delta^*(q_0, w) = p\}.$$

The set Q_r can be easily computed by stages.

If $Q \neq Q_r$, we can "clean up" D by deleting the states in $Q - Q_r$ and restricting the transition function δ to Q_r.

This way, we get an equivalent DFA D_r such that $L(D) = L(D_r)$, where all the states of D_r are reachable. From now on, we assume that we are dealing with DFA's such that $D = D_r$ (called *reachable, or trim*).

Recall that an *equivalence relation* $\simeq$ on a set A is a relation which is *reflexive*, *symmetric*, and *transitive*.

Given any $a \in A$, the set

$$\{b \in A \mid a \simeq b\}$$

is called the *equivalence class of a*, and it is denoted as $[a]_{\simeq}$, or even as $[a]$.

Recall that for any two elements $a, b \in A$, $[a] \cap [b] = \emptyset$ iff $a \not\simeq b$, and $[a] = [b]$ iff $a \simeq b$. The set of equivalence classes associated with the equivalence relation $\simeq$ is a *partition* Π of A (also denoted as $A/\simeq$). This means that it is a family of nonempty pairwise disjoint sets whose union is equal to A itself.

The equivalence classes are also called the *blocks* of the partition Π. The number of blocks in the partition Π is called the *index* of $\simeq$ (and Π).

Given any two equivalence relations $\simeq_1$ and $\simeq_2$ with associated partitions Π_1 and Π_2,

$$\simeq_1 \subseteq \simeq_2$$

iff every block of the partition Π_1 is contained in some block of the partition Π_2. Then, every block of the partition Π_2 is the union of blocks of the partition Π_1, and we say that $\simeq_1$ is a *refinement* of $\simeq_2$ (and similarly, Π_1 is a refinement of Π_2). Note that Π_2 has at most as many blocks as Π_1 does.

We now define an equivalence relation on strings induced by a DFA. This equivalence is a kind of "observational" equivalence, in the sense that we decide that two strings u, v are equivalent iff, when feeding first u and then v to the DFA, u and v drive the DFA to the same state. From the point of view of the observer, u and v have the same effect (reaching the same state).

Definition 2.17.1 Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, we define the relation $\simeq_D$ (*Myhill-Nerode equivalence*) on Σ^* as follows: for any two strings $u, v \in \Sigma^*$,

$$u \simeq_D v \quad \text{iff} \quad \delta^*(q_0, u) = \delta^*(q_0, v).$$

We can figure out what the equivalence classes of $\simeq_D$ are for the following DFA:

	a	b
0	1	0
1	2	1
2	0	2

with 0 both start state and (unique) final state. For example

$$\begin{aligned} abbabbb &\simeq_D aa \\ ababab &\simeq_D \epsilon \\ bba &\simeq_D a. \end{aligned}$$

There are three equivalences classes:

$$[\epsilon]_{\simeq}, \quad [a]_{\simeq}, \quad [aa]_{\simeq}.$$

Observe that $L(D) = [\epsilon]_{\simeq}$. Also, the equivalence classes are in one–to–one correspondence with the states of D.

The relation $\simeq_D$ turns out to have some interesting properties. In particular, it is *right-invariant*, which means that for all $u, v, w \in \Sigma^*$, if $u \simeq v$, then $uw \simeq vw$.

Lemma 2.17.2 *Given any trim (accessible) DFA $D = (Q, \Sigma, \delta, q_0, F)$, the relation $\simeq_D$ is an equivalence relation which is right-invariant and has finite index. Furthermore, if Q has n states, then the index of $\simeq_D$ is n, and every equivalence class of $\simeq_D$ is a regular language. Finally, $L(D)$ is the union of some of the equivalence classes of $\simeq_D$.*

The remarkable fact due to Myhill and Nerode, is that lemma 2.17.2 has a converse.

Lemma 2.17.3 *Given any equivalence relation $\simeq$ on Σ^*, if $\simeq$ is right-invariant and has finite index n, then every equivalence class (block) in the partition Π associated with $\simeq$ is a regular language.*

Proof. Let $C_1, \ldots, C_n$ be the blocks of Π, and assume that $C_1 = [\epsilon]$ is the equivalence class of the empty string.

First, we claim that for every block C_i and every $w \in \Sigma^*$, there is a unique block C_j such that $C_i w \subseteq C_j$, where $C_i w = \{uw \mid u \in C_i\}$.

We also claim that for every $w \in \Sigma^*$, for every block C_i,

$$C_1 w \subseteq C_i \quad \text{iff} \quad w \in C_i.$$

For every class C_k, let

$$D_k = (\{1, \ldots, n\}, \Sigma, \delta, 1, \{k\}),$$

where $\delta(i, a) = j$ iff $C_i a \subseteq C_j$.

Using induction, we have

$$\delta^*(i, w) = j \quad \text{iff} \quad C_i w \subseteq C_j,$$

and using claim 2, it is immediately verified that $L(D_k) = C_k$, proving that every block C_k is a regular language. $\square$

We can combine lemma 2.17.2 and lemma 2.17.3 to get the following characterization of a regular language due to Myhill and Nerode:

Theorem 2.17.4 *(Myhill-Nerode) A language L (over an alphabet Σ) is a regular language iff it is the union of some of the equivalence classes of an equivalence relation $\simeq$ on Σ^*, which is right-invariant and has finite index.*

Given two DFA's D_1 and D_2, whether or not there is a morphism $h\colon D_1 \to D_2$ depends on the relationship between $\simeq_{D_1}$ and $\simeq_{D_2}$. More specifically, we have the following lemma:

Lemma 2.17.5 *Given two DFA's D_1 and D_2, with D_1 trim, the following properties hold.*

(1) There is a DFA morphism $h\colon D_1 \to D_2$ iff

$$\simeq_{D_1} \subseteq \simeq_{D_2} .$$

(2) There is a DFA F-map $h\colon D_1 \to D_2$ iff

$$\simeq_{D_1} \subseteq \simeq_{D_2} \quad \text{and} \quad L(D_1) \subseteq L(D_2);$$

(3) There is a DFA B-map $h\colon D_1 \to D_2$ iff

$$\simeq_{D_1} \subseteq \simeq_{D_2} \quad \text{and} \quad L(D_2) \subseteq L(D_1).$$

Furthermore, h is surjective iff D_2 is trim.

Theorem 2.17.4 can also be used to prove that certain languages are not regular. For example, we prove that $L = \{a^n b^n \mid n \geq 1\}$ is not regular.

The general method is to find three strings

$$x, y, z \in \Sigma^*$$

such that

$$x \simeq y$$

and

$$xz \in L \quad \text{and} \quad yz \notin L.$$

Another useful tool for proving that languages are not regular is the so-called *pumping lemma*.

Lemma 2.17.6 *Given any DFA* $D = (Q, \Sigma, \delta, q_0, F)$ *there is some* $m \geq 1$ *such that for every* $w \in \Sigma^*$*, if* $w \in L(D)$ *and* $|w| \geq m$*, then there exists a decomposition of* w *as* $w = uxv$*, where*

(1) $x \neq \epsilon$,

(2) $ux^iv \in L(D)$*, for all* $i \geq 0$*, and*

(3) $|ux| \leq m$.

Moreover, m *can be chosen to be the number of states of the DFA* D*.*

Typically, the pumping lemma is used to prove that a language is not regular. The method is to proceed by contradiction, i.e., to assume (contrary to what we wish to prove) that a language L is indeed regular, and derive a contradiction of the pumping lemma.

Thus, it would be helpful to see what the negation of the pumping lemma is, and for this, we first state the pumping lemma as a logical formula.

We will use the following abbreviations:

$$\begin{aligned} nat &= \{0, 1, 2, \ldots\}, \\ pos &= \{1, 2, \ldots\}, \\ A &\equiv w = uxv, \\ B &\equiv x \neq \epsilon, \\ C &\equiv |ux| \leq m, \\ P &\equiv \forall i \colon nat\, (ux^i v \in L(D)). \end{aligned}$$

The pumping lemma can be stated as

$$\forall D \colon \mathrm{DFA}\, \exists m \colon pos\, \forall w \colon \Sigma^*$$
$$((w \in L(D) \wedge |w| \geq m) \supset (\exists u, x, v \colon \Sigma^*\, A \wedge B \wedge C \wedge P)).$$

Recalling that

$$\neg(A \wedge B \wedge C \wedge P) \equiv \neg(A \wedge B \wedge C) \vee \neg P \equiv (A \wedge B \wedge C) \supset \neg P$$

and

$$\neg(R \supset S) \equiv R \wedge \neg S,$$

the negation of the pumping lemma can be stated as

$$\exists D\colon \mathrm{DFA}\ \forall m\colon pos\ \exists w\colon \Sigma^*$$
$$((w \in L(D) \wedge |w| \geq m) \wedge (\forall u, x, v\colon \Sigma^* (A \wedge B \wedge C) \supset \neg P)).$$

Since

$$\neg P \equiv \exists i\colon nat\ (ux^i v \notin L(D)),$$

in order to show that the pumping lemma is contradicted, one needs to show that for some DFA D, for every $m \geq 1$, there is some string $w \in L(D)$ of length at least m, such that for every possible decomposition $w = uxv$ satisfying the constraints $x \neq \epsilon$ and $|ux| \leq m$, there is some $i \geq 0$ such that $ux^i v \notin L(D)$.

We now consider an equivalence relation associated with a language L.

2.18 Minimal DFA's

Given any language L (not necessarily regular), we can define an equivalence relation ρ_L which is right-invariant, but not necessarily of finite index. However, when L is regular, the relation ρ_L has finite index. In fact, this index is the size of a smallest DFA accepting L. This will lead us to a construction of minimal DFA's.

Definition 2.18.1 Given any language L (over Σ), we define the relation ρ_L on Σ^* as follows: for any two strings $u, v \in \Sigma^*$,

$$u\rho_L v \quad \text{iff} \quad \forall w \in \Sigma^*(uw \in L \quad \text{iff} \quad vw \in L).$$

We leave as an easy exercise to prove that ρ_L is an equivalence relation which is right-invariant. It is also clear that L is the union of the equivalence classes of strings in L.

When L is also regular, we have the following remarkable result:

Lemma 2.18.2 *Given any regular language L, for any (accessible) DFA $D = (Q, \Sigma, \delta, q_0, F)$ such that $L = L(D)$, ρ_L is a right-invariant equivalence relation, and we have $\simeq_D \subseteq \rho_L$. Furthermore, if ρ_L has m classes and Q has n states, then $m \leq n$.*

Lemma 2.18.2 shows that when L is regular, the index m of ρ_L is finite, and it is a lower bound on the size of all DFA's accepting L.

It remains to show that a DFA with m states accepting L exists. However, going back to the proof of lemma 2.17.3 starting with the right-invariant equivalence relation ρ_L of finite index m, if L is the union of the classes $C_{i_1}, \ldots, C_{i_k}$, the DFA

$$D_{\rho_L} = (\{1, \ldots, m\}, \Sigma, \delta, 1, \{i_1, \ldots, i_k\}),$$

where $\delta(i, a) = j$ iff $C_i a \subseteq C_j$, is such that $L = L(D_{\rho_L})$. Thus, D_{ρ_L} is a minimal DFA accepting L.

In the next section, we give an algorithm which allows us to find D_{ρ_L}, given any DFA D accepting L. This algorithms finds which states of D are equivalent.

2.19 State Equivalence and Minimal DFA's

The proof of lemma 2.18.2 suggests the following definition of an equivalence between states.

Definition 2.19.1 Given any DFA $D = (Q, \Sigma, \delta, q_0, F)$, the relation $\equiv$ on Q, called *state equivalence*, is defined as follows: for all $p, q \in Q$,

$$p \equiv q \quad \text{iff} \quad \forall w \in \Sigma^*(\delta^*(p, w) \in F \quad \text{iff} \quad \delta^*(q, w) \in F).$$

When $p \equiv q$, we say that *p and q are indistinguishable*.

It is trivial to verify that $\equiv$ is an equivalence relation, and that it satisfies the following property:

$$\text{if } p \equiv q \text{ then } \delta(p, a) \equiv \delta(q, a),$$

for all $a \in \Sigma$.

In the DFA of Figure 2.18, states A and C are equivalent. No other two states are equivalent.

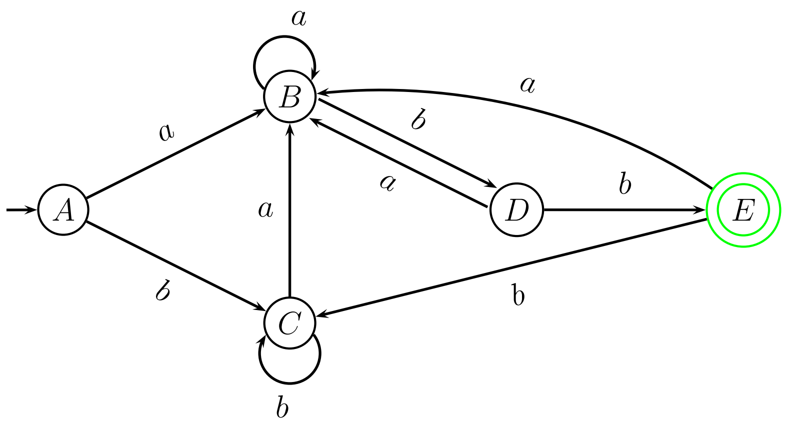

Figure 2.18: A non-minimal DFA for $\{a, b\}^*\{abb\}$

If $L = L(D)$, the following lemma shows the relationship between ρ_L and $\equiv$ and, more generally, between the DFA D_{ρ_L} and the DFA, $D/\equiv$, obtained as the quotient of the DFA D modulo the equivalence relation $\equiv$ on Q and defined such that

$$D/\equiv \,=\, (Q/\equiv, \Sigma, \delta/\equiv, [q_0]_\equiv, F/\equiv),$$

where

$$\delta/\equiv([p]_\equiv, a) = [\delta(p, a)]_\equiv.$$

The minimal DFA $D/\equiv$ is obtained by merging the states in each block of the partition Π associated with $\equiv$, forming states corresponding to the blocks of Π, and drawing a transition on input a from a block C_i to a block C_j of Π iff there is a transition $q = \delta(p, a)$ from any state $p \in C_i$ to any state $q \in C_j$ on input a.

The start state is the block containing q_0, and the final states are the blocks consisting of final states.

Lemma 2.19.2 *For any (accessible) DFA $D = (Q, \Sigma, \delta, q_0, F)$ accepting the regular language $L = L(D)$, the function $\varphi \colon \Sigma^* \to Q$ defined such that*

$$\varphi(u) = \delta^*(q_0, u)$$

induces a bijection $\widehat{\varphi} \colon \Sigma^*/\rho_L \to Q/\equiv$, defined such that

$$\widehat{\varphi}([u]_{\rho_L}) = [\delta^*(q_0, u)]_{\equiv}.$$

Furthermore, we have

$$[u]_{\rho_L} a \subseteq [v]_{\rho_L} \quad \textit{iff} \quad \delta(\varphi(u), a) \equiv \varphi(v).$$

Consequently, $\widehat{\varphi}$, induces an isomorphism of DFA's, $\widehat{\varphi} \colon D_{\rho_L} \to D/\equiv$ (an invertible F-map whose inverse is also an F-map; from a homework problem, such a map must be an invertible proper homomorphism whose inverse is also a proper homomorphism).

The DFA $D/\equiv$ is isomorphic to the minimal DFA D_{ρ_L} accepting L, and thus, it is a minimal DFA accepting L.

Note that if $F = \emptyset$, then $\equiv$ has a single block (Q), and if $F = Q$, then $\equiv$ has a single block (F). In the first case, the minimal DFA is the one state DFA rejecting all strings. In the second case, the minimal DFA is the one state DFA accepting all strings.

When $F \neq \emptyset$ and $F \neq Q$, there are at least two states in Q, and $\equiv$ also has at least two blocks, as we shall see shortly.

It remains to compute $\equiv$ explicitly. This is done using a sequence of approximations. In view of the previous discussion, we are assuming that $F \neq \emptyset$ and $F \neq Q$, which means that $n \geq 2$, where n is the number of states in Q.

Definition 2.19.3 Given any DFA $D = (Q, \Sigma, \delta, q_0, F)$, for every $i \geq 0$, the relation $\equiv_i$ on Q, called *i-state equivalence*, is defined as follows: for all $p, q \in Q$,

$$p \equiv_i q \quad \text{iff} \quad \forall w \in \Sigma^*,\ |w| \leq i$$
$$(\delta^*(p, w) \in F \quad \text{iff} \quad \delta^*(q, w) \in F).$$

When $p \equiv_i q$, we say that
p and q are i-indistinguishable.

It remains to compute $\equiv_{i+1}$ from $\equiv_i$, which can be done using the following lemma. The lemma also shows that

$$\equiv = \equiv_{i_0} .$$

Lemma 2.19.4 *For any (accessible) DFA* $D = (Q, \Sigma, \delta, q_0, F)$, *for all* $p, q \in Q$,

$p \equiv_{i+1} q$ *iff* $p \equiv_i q$ *and* $\delta(p, a) \equiv_i \delta(q, a)$, *for every* $a \in \Sigma$.

Furthermore, if $F \neq \emptyset$ *and* $F \neq Q$, *there is a smallest integer* $i_0 \leq n - 2$, *such that*

$$\equiv_{i_0+1} = \equiv_{i_0} = \equiv .$$

Note that if $F = Q$ or $F = \emptyset$, then $\equiv = \equiv_0$, and the inductive characterization of Lemma 2.19.4 holds trivially.

Using lemma 2.19.4, we can compute $\equiv$ inductively, starting from $\equiv_0 = (F, Q - F)$, and computing $\equiv_{i+1}$ from $\equiv_i$, until the sequence of partitions associated with the $\equiv_i$ stabilizes.

There are a number of algorithms for computing $\equiv$, or to determine whether $p \equiv q$ for some given $p, q \in Q$.

A simple method to compute $\equiv$ is described in Hopcroft and Ullman. It consists in forming a triangular array corresponding to all unordered pairs (p, q), with $p \neq q$ (the rows and the columns of this triangular array are indexed by the states in Q, where the entries are below the descending diagonal).

Initially, the entry (p, q) is marked iff p and q are **not** 0-equivalent, which means that p and q are not both in F or not both in $Q - F$. Then, we process every unmarked entry on every row as follows: for any unmarked pair (p, q), we consider pairs $(\delta(p, a), \delta(q, a))$, for all $a \in \Sigma$. If any pair $(\delta(p, a), \delta(q, a))$ is already marked, this means that $\delta(p, a)$ and $\delta(q, a)$ are inequivalent, and thus p and q are inequivalent, and we mark the pair (p, q).

We continue in this fashion, until at the end of a round during which all the rows are processed, nothing has changed. When the algorithm stops, all marked pairs are inequivalent, and all unmarked pairs correspond to equivalent states.

Let us illustrates the above method. Consider the following DFA accepting $\{a, b\}^*\{abb\}$.

$$\begin{array}{ccc} & a & b \\ A & B & C \\ B & B & D \\ C & B & C \\ D & B & E \\ E & B & C \end{array}$$

The start state is A, and the set of final states is $F = \{E\}$.

The initial (half) array is as follows, using $\times$ to indicate that the corresponding pair (say, (E, A)) consists of inequivalent states, and $\square$ to indicate that nothing is known yet.

$$\begin{array}{ccccc}
B & \square & & & \\
C & \square & \square & & \\
D & \square & \square & \square & \\
E & \times & \times & \times & \times \\
 & A & B & C & D
\end{array}$$

After the first round, we have

$$\begin{array}{ccccc}
B & \square & & & \\
C & \square & \square & & \\
D & \times & \times & \times & \\
E & \times & \times & \times & \times \\
 & A & B & C & D
\end{array}$$

After the second round, we have

$$\begin{array}{ccccc}
B & \times & & & \\
C & \square & \times & & \\
D & \times & \times & \times & \\
E & \times & \times & \times & \times \\
 & A & B & C & D
\end{array}$$

Finally, nothing changes during the third round, and thus, only A and C are equivalent, and we get the four equivalence classes

$$(\{A, C\}, \{B\}, \{D\}, \{E\}).$$

There are ways of improving the efficiency of this algorithm, see Hopcroft and Ullman for such improvements.

Fast algorithms for testing whether $p \equiv q$ for some given $p, q \in Q$ also exist. One of these algorithms is based on "*forward closures*", an idea due to Knuth. Such an algorithm is related to a fast unification algorithm.