

CIS 511
Formal Languages And Automata
Models of Computation, Computability
Basics of Recursive Function Theory

Jean Gallier
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
e-mail: jean@saul.cis.upenn.edu

© Jean Gallier

Please, do not reproduce without permission of the author

November 7, 2007

Contents

1	Introduction	5
2	Regular Languages, Minimization of DFA's	7
2.1	Right-Invariant Equivalence Relations on Σ^*	7
2.2	Finding minimal DFA's	15
2.3	State Equivalence and Minimal DFA's	16
3	Context-Free Grammars And Languages	25
3.1	Context-Free Grammars	25
3.2	Derivations and Context-Free Languages	26
3.3	Normal Forms for Context-Free Grammars	32
3.4	Regular Languages are Context-Free	39
3.5	Useless Productions in Context-Free Grammars	40
3.6	The Greibach Normal Form	42
3.7	Least Fixed-Points	43
3.8	Context-Free Languages as Least Fixed-Points	45
3.9	Least Fixed-Points and the Greibach Normal Form	49
3.10	Tree Domains and Gorn Trees	54
3.11	Derivations Trees	57
3.12	Ogden's Lemma	59
4	RAM Programs, Turing Machines	67
4.1	Introduction	67
5	Universal RAM Programs and the Halting Problem	69
5.1	Pairing Functions	69
5.2	Equivalence of Alphabets	71
5.3	Coding of RAM Programs	75
5.4	Kleene's T -Predicate	81
6	Elementary Recursive Function Theory	85
6.1	Acceptable Indexings	85
6.2	Undecidable Problems	88

6.3	Recursively Enumerable Sets	93
6.4	Reducibility and Complete Sets	97
6.5	The Recursion Theorem	101
6.6	Extended Rice Theorem	105
6.7	Creative and Productive Sets	107

Chapter 1

Introduction

The theory of computation is concerned with algorithms and algorithmic systems: their design and representation, their completeness, and their complexity.

The purpose of these notes is to introduce some of the basic notions of the theory of computation, including concepts from formal languages and automata theory, and the theory of computability and some basics of recursive function theory. Other topics such as correctness of programs and computational complexity will not be treated here (there just isn't enough time!).

The notes are divided into two parts. The first part is devoted to formal languages and automata. The second part deals with models of computation, recursive functions, and undecidability.

Chapter 2

Regular Languages and Equivalence Relations, The Myhill-Nerode Characterization, State Equivalence

2.1 Right-Invariant Equivalence Relations on Σ^*

The purpose of this section is to give one more characterization of the regular languages in terms of certain kinds of equivalence relations on strings. Pushing this characterization a bit further, we will be able to show how minimal DFA's can be found.

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. This motivates the following definition: 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 2.1.1 Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, we define the relation \simeq_D 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 equivalence 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 2.1.2 *Given any (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 .*

Proof. The fact that \simeq_D is an equivalence relation is a trivial verification. To prove that \simeq_D is right-invariant, we first prove by induction on the length of v that for all $u, v \in \Sigma^*$, for all $p \in Q$,

$$\delta^*(p, uv) = \delta^*(\delta^*(p, u), v).$$

Then, if $u \simeq_D v$, which means that $\delta^*(q_0, u) = \delta^*(q_0, v)$, we have

$$\delta^*(q_0, uw) = \delta^*(\delta^*(q_0, u), w) = \delta^*(\delta^*(q_0, v), w) = \delta^*(q_0, vw),$$

which means that $uw \simeq_D vw$. Thus, \simeq_D is right-invariant. We still have to prove that \simeq_D has index n . Define the function $f: \Sigma^* \rightarrow Q$ such that

$$f(u) = \delta^*(q_0, u).$$

Note that if $u \simeq_D v$, which means that $\delta^*(q_0, u) = \delta^*(q_0, v)$, then $f(u) = f(v)$. Thus, the function $f: \Sigma^* \rightarrow Q$ induces a function $\widehat{f}: \Pi \rightarrow Q$ defined such that

$$\widehat{f}([u]) = f(u),$$

for every equivalence class $[u] \in \Pi$, where $\Pi = \Sigma^* / \simeq$ is the partition associated with \simeq_D . However, the function $\widehat{f}: \Pi \rightarrow Q$ is injective (one-to-one), since $\widehat{f}([u]) = \widehat{f}([v])$ means that $\delta^*(q_0, u) = \delta^*(q_0, v)$, which means precisely that $u \simeq_D v$, i.e., $[u] = [v]$. Since Q has n states, Π has at most n blocks. Moreover, since every state is accessible, for every $q \in Q$, there is some $w \in \Sigma^*$ so that $\delta^*(q_0, w) = q$, which shows that $\widehat{f}([w]) = f(w) = q$. Consequently, \widehat{f} is also surjective. But then, being injective and surjective, \widehat{f} is bijective and Π has exactly n blocks.

Every equivalence class of Π is a set of strings of the form

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

for some $p \in Q$, which is accepted by the DFA obtained from D by changing F to $\{p\}$. Thus, every equivalence class is a regular language. Finally, $L(D)$ is the union of the equivalence classes corresponding to the final states in F . \square

The remarkable fact due to Myhill and Nerode, is that lemma 2.1.2 has a converse.

Lemma 2.1.3 *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, \dots, 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 $C_i w = \{uw \mid u \in C_i\}$.

For every $u \in C_i$, the string uw belongs to one and only one of the blocks of Π , say C_j . For any other string $v \in C_i$, since (by definition) $u \simeq v$, by right invariance, we get $uw \simeq vw$, but since $uw \in C_j$ and C_j is an equivalence class, we also have $vw \in C_j$. This proves the first claim.

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

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

If $C_1 w \subseteq C_i$, since $C_1 = [\epsilon]$, we have $\epsilon w = w \in C_i$. Conversely, if $w \in C_i$, for any $v \in C_1 = [\epsilon]$, since $\epsilon \simeq v$, by right invariance we have $w \simeq vw$, and thus $vw \in C_i$, which shows that $C_1 w \subseteq C_i$.

For every class C_k , let

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

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

It is easily shown by induction on $|w|$ that

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

Then, using the second claim, we have

$$\begin{aligned} L(D_k) &= \{w \in \Sigma^* \mid \delta^*(1, w) \in \{k\}\} \\ &= \{w \in \Sigma^* \mid \delta^*(1, w) = k\} \\ &= \{w \in \Sigma^* \mid C_1 w \subseteq C_k\} \\ &= \{w \in \Sigma^* \mid w \in C_k\} = C_k, \end{aligned}$$

proving that every block, C_k , is a regular language. \square



In general it is false that $C_i a = C_j$ for some block C_j , and we can only claim that $C_i a \subseteq C_j$.

We can combine lemma 2.1.2 and lemma 2.1.3 to get the following characterization of a regular language due to Myhill and Nerode:

Theorem 2.1.4 (*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 $h: D_1 \rightarrow D_2$ depends on the relationship between \simeq_{D_1} and \simeq_{D_2} . More specifically, we have the following lemma:

Lemma 2.1.5 *Given two DFA's D_1 and D_2 , the following properties hold.*

(1) *There is a DFA morphism $h: D_1 \rightarrow D_2$ iff*

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

(2) *There is a DFA F-map $h: D_1 \rightarrow D_2$ iff*

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

(3) *There is a DFA F^{-1} -map $h: D_1 \rightarrow D_2$ iff*

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

Furthermore, if D_1 is trim, h is surjective iff D_2 is trim.

Theorem 2.1.4 can also be used to prove that certain languages are not regular. A general scheme (not the only one) goes as follows: If L is not regular, then it must be infinite. Now, we argue by contradiction. If L was regular, then by Myhill-Nerode, there would be some equivalence relation, \simeq , which is right-invariant and of finite index and such that L is the union of some of the classes of \simeq . Because Σ^* is infinite and \simeq has only finitely many equivalence classes, there are strings $x, y \in \Sigma^*$ with $x \neq y$ so that

$$x \simeq y.$$

If we can find a third string, $z \in \Sigma^*$, such that

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

then we reach a contradiction. Indeed, by right invariance, from $x \simeq y$, we get $xz \simeq yz$. But, L is the union of equivalence classes of \simeq , so if $xz \in L$, then we should also have $yz \in L$, contradicting $yz \notin L$. Therefore, L is not regular.

For example, we prove that $L = \{a^n b^n \mid n \geq 1\}$ is not regular.

Assuming for the sake of contradiction that L is regular, there is some equivalence relation \simeq which is right-invariant and of finite index and such that L is the union of some of the classes of \simeq . Since the set

$$\{a, aa, aaa, \dots, a^i, \dots\}$$

is infinite and \simeq has a finite number of classes, two of these strings must belong to the same class, which means that $a^i \simeq a^j$ for some $i \neq j$. But since \simeq is right invariant, by concatenating with b^i on the right, we see that $a^i b^i \simeq a^j b^i$ for some $i \neq j$. However $a^i b^i \in L$,

and since L is the union of classes of \simeq , we also have $a^j b^i \in L$ for $i \neq j$, which is absurd, given the definition of L . Thus, in fact, L is not regular.

Here is another illustration of the use of the Myhill-Nerode Theorem to prove that a language is not regular. We claim that the language,

$$L = \{a^{n!} \mid n \geq 1\},$$

is not regular, where $n!$ (n factorial) is given by $0! = 1$ and $(n+1)! = (n+1)n!$.

Assume L is regular. Then, there is some equivalence relation \simeq which is right-invariant and of finite index and such that L is the union of some of the classes of \simeq . Since the sequence

$$a, a^2, \dots, a^n, \dots$$

is infinite, two of these strings must belong to the same class, which means that $a^p \simeq a^q$ for some p, q with $1 \leq p < q$. As $q! \geq q$ for all $q \geq 0$ and $q > p$, we can concatenate on the right with $a^{q!-p}$ and we get

$$a^p a^{q!-p} \simeq a^q a^{q!-p},$$

that is,

$$a^{q!} \simeq a^{q!+q-p}.$$

If we can show that

$$q! + q - p < (q+1)!$$

we will obtain a contradiction because then $a^{q!+q-p} \notin L$, yet $a^{q!+q-p} \simeq a^{q!}$ and $a^{q!} \in L$, contradicting Myhill-Nerode. Now, as $1 \leq p < q$, we have $q-p \leq q-1$, so if we can prove that

$$q! + q - p \leq q! + q - 1 < (q+1)!$$

we will be done. However, $q! + q - 1 < (q+1)!$ is equivalent to

$$q - 1 < (q+1)! - q!,$$

and since $(q+1)! - q! = (q+1)q! - q! = qq!$, we simply need to prove that

$$q - 1 < q \leq qq!,$$

which holds for $q \geq 1$.

There is another version of the Myhill-Nerode Theorem involving congruences which is also quite useful. An equivalence relation, \simeq , on Σ^* is *left and right-invariant* iff for all $x, y, u, v \in \Sigma^*$,

$$\text{if } x \simeq y \text{ then } uxv \simeq uyv.$$

An equivalence relation, \simeq , on Σ^* is a *congruence* iff for all $u_1, u_2, v_1, v_2 \in \Sigma^*$,

$$\text{if } u_1 \simeq v_1 \text{ and } u_2 \simeq v_2 \text{ then } u_1 u_2 \simeq v_1 v_2.$$

It is easy to prove that an equivalence relation is a congruence iff it is left and right-invariant, the proof is left as an exercise.

There is a version of Lemma 2.1.2 that applies to congruences and for this we define the relation \sim_D as follows: For any (trim) DFA, $D = (Q, \Sigma, \delta, q_0, F)$, for all $x, y \in \Sigma^*$,

$$x \sim_D y \quad \text{iff} \quad (\forall q \in Q)(\delta^*(q, x) = \delta^*(q, y)).$$

Lemma 2.1.6 *Given any (trim) DFA, $D = (Q, \Sigma, \delta, q_0, F)$, the relation \sim_D is an equivalence relation which is left and right-invariant and has finite index. Furthermore, if Q has n states, then the index of \sim_D is at most n^n and every equivalence class of \sim_D is a regular language. Finally, $L(D)$ is the union of some of the equivalence classes of \sim_D .*

Proof We leave the proof of Lemma 2.1.6 as an exercise. We just make the following remark: Observe that

$$\sim_D \subseteq \simeq_D,$$

since the condition $\delta^*(q, x) = \delta^*(q, y)$ holds for every $q \in Q$, so in particular for $q = q_0$. But then, every equivalence class of \simeq_D is the union of equivalence classes of \sim_D and since, by Lemma 2.1.2, L is the union of equivalence classes of \simeq_D , we conclude that L is also the union of equivalence classes of \sim_D . \square

Using Lemma 2.1.6 and Lemma 2.1.3, we obtain another version of the Myhill-Nerode Theorem.

Theorem 2.1.7 *(Myhill-Nerode, Congruence Version) 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 a congruence and has finite index.*

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

Lemma 2.1.8 *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) $ux^i v \in L(D)$, 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 .

Proof. Let m be the number of states in Q , and let $w = w_1 \dots w_n$. Since Q contains the start state q_0 , $m \geq 1$. Since $|w| \geq m$, we have $n \geq m$. Since $w \in L(D)$, let

$$(q_0, q_1, \dots, q_n),$$

be the sequence of states in the accepting computation of w (where $q_n \in F$). Consider the subsequence

$$(q_0, q_1, \dots, q_m).$$

This sequence contains $m + 1$ states, but there are only m states in Q , and thus, we have $q_i = q_j$, for some i, j such that $0 \leq i < j \leq m$. Then, letting $u = w_1 \dots w_i$, $x = w_{i+1} \dots w_j$, and $v = w_{j+1} \dots w_n$, it is clear that the conditions of the lemma hold. \square

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:

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

The pumping lemma can be stated as

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

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

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

Since

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

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 $w = u xv$ satisfying the constraints $x \neq \epsilon$ and $|ux| \leq m$, there is some $i \geq 0$ such that $ux^i v \notin L(D)$. When proceeding by contradiction, we have a language L that we are (wrongly) assuming to be regular, and we can use any DFA D accepting L . The creative part of the argument is to pick the right $w \in L$ (not making any assumption on $m \leq |w|$).

As an illustration, let us use the pumping lemma to prove that $L = \{a^n b^n \mid n \geq 1\}$ is not regular. The usefulness of the condition $|ux| \leq m$ lies in the fact that it reduces the number of legal decomposition $u xv$ of w . We proceed by contradiction. Thus, let us assume that $L = \{a^n b^n \mid n \geq 1\}$ is regular. If so, it is accepted by some DFA D . Now, we wish to contradict the pumping lemma. For every $m \geq 1$, let $w = a^m b^m$. Clearly, $w = a^m b^m \in L$ and $|w| \geq m$. Then, every legal decomposition u, x, v of w is such that

$$w = \underbrace{a \dots a}_u \underbrace{a \dots a}_x \underbrace{a \dots a b \dots b}_v$$

where $x \neq \epsilon$ and x ends within the a 's, since $|ux| \leq m$. Since $x \neq \epsilon$, the string $u x x v$ is of the form $a^n b^m$ where $n > m$, and thus $u x x v \notin L$, contradicting the pumping lemma.

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

2.2 Finding 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 2.2.1 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 . This is because if $u \in L$ and $u \rho_L v$, by letting $w = \epsilon$ in the definition of ρ_L , we get

$$u \in L \quad \text{iff} \quad v \in L,$$

and since $u \in L$, we also have $v \in L$. This implies that if $u \in L$ then $[u]_{\rho_L} \subseteq L$ and so,

$$L = \bigcup_{u \in L} [u]_{\rho_L}.$$

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

Lemma 2.2.2 *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 $m \leq n$.*

Proof. By definition, $u \simeq_D v$ iff $\delta^*(q_0, u) = \delta^*(q_0, v)$. Since $w \in L(D)$ iff $\delta^*(q_0, w) \in F$, the fact that $u\rho_L v$ can be expressed as

$$\begin{aligned} \forall w \in \Sigma^*(uw \in L \text{ iff } vw \in L), & \quad \text{iff,} \\ \forall w \in \Sigma^*(\delta^*(q_0, uw) \in F \text{ iff } \delta^*(q_0, vw) \in F), & \quad \text{iff} \\ \forall w \in \Sigma^*(\delta^*(\delta^*(q_0, u), w) \in F \text{ iff } \delta^*(\delta^*(q_0, v), w) \in F), & \end{aligned}$$

and if $\delta^*(q_0, u) = \delta^*(q_0, v)$, this shows that $u\rho_L v$. Since the number of classes of \simeq_D is n and $\simeq_D \subseteq \rho_L$, the equivalence relation ρ_L has fewer classes than \simeq_D , and $m \leq n$. \square

Lemma 2.2.2 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 2.1.3 starting with the right-invariant equivalence relation ρ_L of finite index m , if L is the union of the classes C_{i_1}, \dots, C_{i_k} , the DFA

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

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 algorithm finds which states of D are equivalent.

2.3 State Equivalence and Minimal DFA's

The proof of lemma 2.2.2 suggests the following definition of an equivalence between states:

Definition 2.3.1 Given any DFA $D = (Q, \Sigma, \delta, q_0, F)$, the relation \equiv on Q , called *state equivalence*, is defined as follows: for all $p, q \in Q$,

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

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 $a \in \Sigma$.

The reader should check that states A and C in the DFA below are equivalent and that no other distinct states are equivalent.

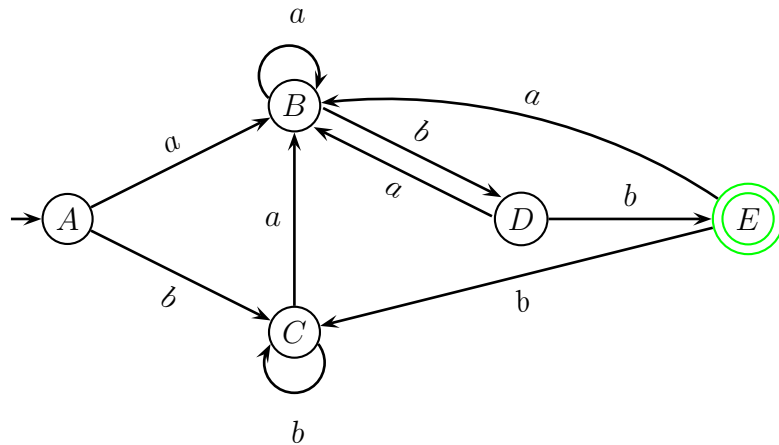


Figure 2.1: A non-minimal DFA for $\{a, b\}^*\{abb\}$

It might be illuminating to express state equivalence as the equality of two languages. Given the DFA $D = (Q, \Sigma, \delta, q_0, F)$, let $D_p = (Q, \Sigma, \delta, p, F)$ be the DFA obtained from D by redefining the start state to be p . Then, it is clear that

$$p \equiv q \quad \text{iff} \quad L(D_p) = L(D_q).$$

This simple observation implies that there is an algorithm to test state equivalence. Indeed, we simply have to test whether the DFA's D_p and D_q accept the same language and this can be done using the cross-product construction. Indeed, $L(D_p) = L(D_q)$ iff $L(D_p) - L(D_q) = \emptyset$ and $L(D_q) - L(D_p) = \emptyset$. Now, if $(D_p \times D_q)_{1-2}$ denotes the cross-product DFA with starting state (p, q) and with final states $F \times (Q - F)$ and $(D_p \times D_q)_{2-1}$ denotes the cross-product DFA also with starting state (p, q) and with final states $(Q - F) \times F$, we know that

$$L((D_p \times D_q)_{1-2}) = L(D_p) - L(D_q) \quad \text{and} \quad L((D_p \times D_q)_{2-1}) = L(D_q) - L(D_p),$$

so all we need to do if to test whether $(D_p \times D_q)_{1-2}$ and $(D_p \times D_q)_{2-1}$ accept the empty language. However, we know that this is the case iff the set of states reachable from (p, q) in $(D_p \times D_q)_{1-2}$ contains no state in $F \times (Q - F)$ and the set of states reachable from (p, q) in $(D_p \times D_q)_{2-1}$ contains no state in $(Q - F) \times F$. Actually, the graphs of $(D_p \times D_q)_{1-2}$ and $(D_p \times D_q)_{2-1}$ are identical, so we only need to check that no state in $(F \times (Q - F)) \cup (Q - F) \times F$ is reachable from (p, q) in that graph. This algorithm to test state equivalence is not the most efficient but it is quite reasonable (it runs in polyomial time).

If $L = L(D)$, the lemma below 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 as follows:

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

where

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

The 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 2.3.2 *For any (accessible) DFA $D = (Q, \Sigma, \delta, q_0, F)$ accepting the regular language $L = L(D)$, the function $\varphi: \Sigma^* \rightarrow Q$ defined such that*

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

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

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

Furthermore, we have

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

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

Proof. Since $\varphi(u) = \delta^*(q_0, u)$ and $\varphi(v) = \delta^*(q_0, v)$, the fact that $\varphi(u) \equiv \varphi(v)$ can be expressed as

$$\begin{aligned} \forall w \in \Sigma^* (\delta^*(\delta^*(q_0, u), w) \in F \quad \text{iff} \quad \delta^*(\delta^*(q_0, v), w) \in F), & \quad \text{iff} \\ \forall w \in \Sigma^* (\delta^*(q_0, uw) \in F \quad \text{iff} \quad \delta^*(q_0, vw) \in F), & \end{aligned}$$

which is exactly $u\rho_L v$. Therefore,

$$u\rho_L v \quad \text{iff} \quad \varphi(u) \equiv \varphi(v).$$

From the above, we see that the function $\varphi: \Sigma^* \rightarrow Q$ maps each equivalence class $[u]$ modulo ρ_L to the equivalence class $[\varphi(u)]$ modulo \equiv and so, the function $\widehat{\varphi}: \Sigma^*/\rho_L \rightarrow Q/\equiv$ given by

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

is well-defined. Moreover, $\widehat{\varphi}$ is injective, since $\widehat{\varphi}([u]) = \widehat{\varphi}([v])$ iff $\varphi(u) = \varphi(v)$ iff (from above) $u\rho_L v$ iff $[u] = [v]$. Since every state in Q is accessible, for every $q \in Q$, there is some $u \in \Sigma^*$

so that $\varphi(u) = \delta^*(q_0, u) = q$, so $\widehat{\varphi}([u]) = [q]_{\equiv}$ and $\widehat{\varphi}$ is surjective. Therefore, we have a bijection $\widehat{\varphi}: \Sigma^*/\rho_L \rightarrow Q/\equiv$.

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

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

and thus, $\delta(\varphi(u), a) \equiv \varphi(v)$ can be expressed as $\varphi(ua) \equiv \varphi(v)$. By the previous part, this is equivalent to $ua\rho_L v$, which is equivalent to

$$[u]_{\rho_L} a \subseteq [v]_{\rho_L}.$$

It is then easy to check (do it!) that $\widehat{\varphi}$ induces an F -map of DFA's which is an isomorphism (i.e., an invertible F -map whose inverse is also an F -map), $\widehat{\varphi}: D_{\rho_L} \rightarrow D/\equiv$. \square

Lemma 2.3.2 shows that the DFA D_{ρ_L} is isomorphic to the DFA D/\equiv obtained as the quotient of the DFA D modulo the equivalence relation \equiv on Q . Since D_{ρ_L} is a minimal DFA accepting L , so is D/\equiv .

There are other characterizations of the regular languages. Among those, the characterization in terms of right derivatives is of particular interest because it yields an alternative construction of minimal DFA's.

Definition 2.3.3 Given any language, $L \subseteq \Sigma^*$, for any string, $u \in \Sigma^*$, the *right derivative of L by u* , denoted L/u , is the language

$$L/u = \{w \in \Sigma^* \mid uw \in L\}.$$

Theorem 2.3.4 *If $L \subseteq \Sigma^*$ is any language, then L is regular iff it has finitely many right derivatives. Furthermore, if L is regular, then all its right derivatives are regular and their number is equal to the number of states of the minimal DFA's for L .*

It is easy to check that

$$L/u = L/v \quad \text{iff} \quad u\rho_L v.$$

The above shows that ρ_L has a finite number of classes, say m , iff there is a finite number of right derivatives, say n , and if so, $m = n$. If L is regular, then we know that the number of equivalence classes of ρ_L is the number of states of the minimal DFA's for L , so the number of right derivatives of L is equal to the size of the minimal DFA's for L .

Conversely, if the number of derivatives is finite, say m , then ρ_L has m classes and by Myhill-Nerode, L is regular. It remains to show that if L is regular then every right derivative is regular.

Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting L . If $p = \delta^*(q_0, u)$, then let

$$D_p = (Q, \Sigma, \delta, p, F),$$

that is, D with p as start state. It is clear that

$$L/u = L(D_p),$$

so L/u is regular for every $u \in \Sigma^*$. Also observe that if $|Q| = n$, then there are at most n DFA's D_p , so there is at most n right derivatives, which is another proof of the fact that a regular language has a finite number of right derivatives. \square

If L is regular then the construction of a minimal DFA for L can be recast in terms of right derivatives. Let $L/u_1, L/u_2, \dots, L/u_m$ be the set of all the right derivatives of L . Of course, we may assume that $u_1 = \epsilon$. We form a DFA whose states are the right derivatives, L/u_i . For every state, L/u_i , for every $a \in \Sigma$, there is a transition on input a from L/u_i to $L/u_j = L/(u_i a)$. The start state is $L = L/u_1$ and the final states are the right derivatives, L/u_i , for which $\epsilon \in L/u_i$.

We leave it as an exercise to check that the above DFA accepts L . One way to do this is to recall that $L/u = L/v$ iff $u\rho_L v$ and to observe that the above construction mimics the construction of D_{ρ_L} as in the Myhill-Nerode Proposition (Proposition 2.1.3). This DFA is minimal since the number of right derivatives is equal to the size of the minimal DFA's for L .

We now return to state equivalence. 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 2.3.5 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*.

Since state equivalence \equiv is defined such that

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

we note that testing the condition

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

for all strings in Σ^* is equivalent to testing the above condition for all strings of length at most i for all $i \geq 0$, i.e.

$$p \equiv q \quad \text{iff} \quad \forall i \geq 0 \forall w \in \Sigma^*, |w| \leq i (\delta^*(p, w) \in F \quad \text{iff} \quad \delta^*(q, w) \in F).$$

Since \equiv_i is defined such that

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

we conclude that

$$p \equiv q \quad \text{iff} \quad \forall i \geq 0 (p \equiv_i q).$$

This identity can also be expressed as

$$\equiv = \bigcap_{i \geq 0} \equiv_i .$$

If we assume that $F \neq \emptyset$ and $F \neq Q$, observe that \equiv_0 has exactly two equivalence classes F and $Q - F$, since ϵ is the only string of length 0, and since the condition

$$\delta^*(p, \epsilon) \in F \quad \text{iff} \quad \delta^*(q, \epsilon) \in F$$

is equivalent to the condition

$$p \in F \quad \text{iff} \quad q \in F.$$

It is also obvious from the definition of \equiv_i that

$$\equiv \subseteq \cdots \subseteq \equiv_{i+1} \subseteq \equiv_i \subseteq \cdots \subseteq \equiv_1 \subseteq \equiv_0 .$$

If this sequence was strictly decreasing for all $i \geq 0$, the partition associated with \equiv_{i+1} would contain at least one more block than the partition associated with \equiv_i and since we start with a partition with two blocks, the partition associated with \equiv_i would have at least $i + 2$ blocks. But then, for $i = n - 1$, the partition associated with \equiv_{n-1} would have at least $n + 1$ blocks, which is absurd since Q has only n states. Therefore, there is a smallest integer, $i_0 \leq n - 2$, such that

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

Thus, 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 2.3.6 *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*

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

Proof. By the definition of the relation \equiv_i ,

$$p \equiv_{i+1} q \quad \text{iff} \quad \forall w \in \Sigma^*, |w| \leq i+1 (\delta^*(p, w) \in F \quad \text{iff} \quad \delta^*(q, w) \in F).$$

The trick is to observe that the condition

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

holds for all strings of length at most $i+1$ iff it holds for all strings of length at most i and for all strings of length between 1 and $i+1$. This is expressed as

$$\begin{aligned} p \equiv_{i+1} q \quad \text{iff} \\ \forall w \in \Sigma^*, |w| \leq i (\delta^*(p, w) \in F \quad \text{iff} \quad \delta^*(q, w) \in F) \quad \text{and} \\ \forall w \in \Sigma^*, 1 \leq |w| \leq i+1 (\delta^*(p, w) \in F \quad \text{iff} \quad \delta^*(q, w) \in F). \end{aligned}$$

Obviously, the first condition in the conjunction is $p \equiv_i q$, and since every string w such that $1 \leq |w| \leq i+1$ can be written as au where $a \in \Sigma$ and $0 \leq |u| \leq i$, the second condition in the conjunction can be written as

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

However, $\delta^*(p, au) = \delta^*(\delta(p, a), u)$ and $\delta^*(q, au) = \delta^*(\delta(q, a), u)$, so that the above condition is really

$$\forall a \in \Sigma (\delta(p, a) \equiv_i \delta(q, a)).$$

Thus, we showed that

$$p \equiv_{i+1} q \quad \text{iff} \quad p \equiv_i q \quad \text{and} \quad \forall a \in \Sigma (\delta(p, a) \equiv_i \delta(q, a)).$$

Thus, if $\equiv_{i+1} = \equiv_i$ for some $i \geq 0$, using induction, we also have $\equiv_{i+j} = \equiv_i$ for all $j \geq 1$. Since

$$\equiv = \bigcap_{i \geq 0} \equiv_i, \quad \equiv_{i+1} \subseteq \equiv_i,$$

and since we know that there is a smallest index say i_0 , such that $\equiv_j = \equiv_{i_0}$, for all $j \geq i_0 + 1$, we have $\equiv = \equiv_{i_0}$. \square

Using lemma 2.3.6, 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.

Note that if $F = Q$ or $F = \emptyset$, then $\equiv = \equiv_0$, and the inductive characterization of Lemma 2.3.6 holds trivially.

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 illustrate the above method. Consider the following DFA accepting $\{a, b\}^* \{abb\}$:

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

The start state is A , and the set of final states is $F = \{E\}$. (This is the DFA displayed in Figure 2.1.)

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.

B	\square			
C	\square	\square		
D	\square	\square	\square	
E	\times	\times	\times	\times
	A	B	C	D

After the first round, we have

B	\square			
C	\square	\square		
D	\times	\times	\times	
E	\times	\times	\times	\times
	A	B	C	D

After the second round, we have

B	\times			
C	\square	\times		
D	\times	\times	\times	
E	\times	\times	\times	\times
	A	B	C	D

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”, following an idea of Knuth. Such an algorithm is related to a fast unification algorithm.