# CSci 311, Models of Computation
# Chapter 4
# Properties of Regular Languages

H. Conrad Cunningham

29 December 2015

## Contents

Introduction . . . . . 1
4.1 Closure Properties of Regular Languages . . . . . 2
4.1.1 Mathematical Interlude: Operations and Closure . . . . . 2
4.1.2 Closure under Simple Set Operations . . . . . 3
4.1.3 Closure under Difference (Linz Example 4.1) . . . . . 5
4.1.4 Closure under Reversal . . . . . 5
4.1.5 Homomorphism Definition . . . . . 6
4.1.6 Linz Example 4.2 . . . . . 6
4.1.7 Linz Example 4.3 . . . . . 7
4.1.8 Closure under Homomorphism Theorem . . . . . 7
4.1.9 Right Quotient Definition . . . . . 7
4.1.10 Linz Example 4.4 . . . . . 8
4.1.11 Closure under Right Quotient . . . . . 9
4.1.12 Linz Example 4.5 . . . . . 10
4.2 Elementary Questions about Regular Languages . . . . . 12
4.2.1 Membership? . . . . . 12
4.2.2 Finite or Infinite? . . . . . 13
4.2.3 Equality? . . . . . 13

4.3 Identifying Nonregular Languages . . . . . 14
4.3.1 Using the Pigeonhole Principle . . . . . 14
4.3.2 Linz Example 4.6 . . . . . 14
4.3.3 Pumping Lemma for Regular Languages . . . . . 15
4.3.4 Linz Example 4.7 . . . . . 17
4.3.5 Using the Pumping Lemma (Viewed as a Game) . . . . . 18
4.3.6 Linz Example 4.8 . . . . . 19
4.3.7 Linz Example 4.9 . . . . . 20
4.3.8 Linz Example 4.10 . . . . . 20
4.3.9 Linz Example (Factorial Length Strings) . . . . . 21
4.3.10 Linz Example 4.12 . . . . . 22
4.3.11 Linz Example 4.13 . . . . . 22
4.3.12 Pitfalls in Using the Pumping Lemma . . . . . 24



**Acknowledgements**: MS student Eli Allen assisted in preparation of these notes. These lecture notes are for use with Chapter 4 of the textbook: Peter Linz. *Introduction to Formal Languages and Automata*, Fifth Edition, Jones and Bartlett Learning, 2012. The terminology and notation used in these notes are similar to those used in the Linz textbook. This document uses several figures from the Linz textbook.

**Advisory**: The HTML version of this document requires use of a browser that supports the display of MathML. A good choice as of December 2015 seems to be a recent version of Firefox from Mozilla.

## Introduction

The questions answered in this chapter include:

- What can regular languages *do*?
- What can regular languages *not do*?

The concepts introduced in this chapter are:

- Closure of operations on regular languages
- Membership, finiteness, and equality of regular languages
- Identification of nonregular languages

## 4.1 Closure Properties of Regular Languages

### 4.1.1 Mathematical Interlude: Operations and Closure

**Definition (Operation)**: An *operation* is a function $p : V \to Y$ where $V \in X_1 \times X_2 \times \cdots \times X_k$ for some sets $X_i$ with $0 \leq i \leq k$. $k$ is the number of operands (or arguments) of the operation.

- If $k = 0$, then $p$ is a *nullary* operation.
- If $k = 1$, then $p$ is a *unary* operation.
- If $k = 2$, then $p$ is a *binary* operation.
- etc.

We often use special notation and conventions for unary and binary operations. For example:

- a binary operation may be written in an *infix* style as in $x + y$ and $x \cdot y$
- a unary operation may be written in a *prefix* style as in $-x$, *suffix* style such as $x^*$, or special style such as $\sqrt{3}$ or $\bar{S}$
- a binary operation may be implied by the juxtaposition such as $3x$ for multiplication or (in a different context) $xy$ for string concatenation or implied by superscripting such as $x^2$ for exponentiation

Often we consider an operations *on a set*, where all the operands and the result are drawn from the same set.

**Definition (Closure)**: A set $S$ is *closed* under a unary operation $p$ if, for all $x \in S$, $p(x) \in S$. Similarly, a set $S$ is *closed* under a binary operation $\odot$ if, for all $x \in S$ and $y \in S$, $x \odot y \in S$.

Examples arithmetic on the set of natural numbers ($\mathbb{N} = \{0, 1, ...\}$)

- Binary operations addition ($+$) and multiplication ($*$ in programming languages) are closed on $\mathbb{N}$
  - $\forall x, y \in \mathbb{N}, x + y \in \mathbb{N}$
  - $\forall x, y \in \mathbb{N}, x * y \in \mathbb{N}$
- Binary operations subtraction ($-$) and division ($/$) are not closed on $\mathbb{N}$
  - $\exists x, y \in \mathbb{N}, x - y \notin \mathbb{N}$
    For example, $1 - 2$ is not a natural number.
  - $\exists x, y \in \mathbb{N}, x/y \notin \mathbb{N}$
    For example, $3/2$ is not a natural number.

- Unary operation negation (operator $-$ written in prefix form) is not closed on $\mathbb{N}$.

However, the set of *integers* is closed under subtraction and negation. But it is not closed under division or square root (as we normally define the operations).

Now, let's consider closure of the set of regular languages with respect to the simple set operations.

### 4.1.2 Closure under Simple Set Operations

**Linz Theorem 4.1 (Closure under Simple Set Operations)**: If $L_1$ and $L_2$ are regular languages, then so are $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 L_2$, $\bar{L_1}$, and $L_1^*$.

That is, we say that the family of regular languages is *closed* under *union*, *intersection*, *concatenation*, *complementation*, and *star-closure*.

**Proof of $L_1 \cup L_2$**

Let $L_1$ and $L_2$ be regular languages.

---

$L_1 \cup L_2$
$=$ { Th. 3.2: there exist regular expressions $r_1$, $r_2$ }
$L(r_1) \cup L(r_2)$
$=$ { Def. 3.2, rule 4 }
$L(r_1 + r_2)$

---

Thus, by Theorem 3.1 (regular expressions describe regular languages), the union is a regular language.

Thus $L_1 \cup L_2$ is a regular language. QED.

**Proofs of $L_1 L_2$ and $L_1^*$**

Similar to the proof of $L_1 \cup L_2$.

**Proof of $\bar{L_1}$**

Strategy: Given a dfa $M$ for the regular language, construct a new dfa $\widehat{M}$ that *accepts everything rejected* and *rejects everything accepted* by the given dfa.

---

$L_1$ is a regular language on $\Sigma$.
$\equiv$ { Def. 2.3 }
$\exists$ dfa $M = (Q, \Sigma, \delta, q_0, F)$ such that $L(M) = L_1$.

---

Thus

---

$\omega \in \Sigma^*$

$\Rightarrow$ { by the properties of dfas and sets }

Either $\delta^*(q_0, \omega) \in F$ or $\delta^*(q_0, \omega) \in Q - F$

$\Rightarrow$ { Def. 2.2: language accepted by dfa }

Either $\omega \in L(M)$ or $\omega \in L(\widehat{M})$ for some dfa $\widehat{M}$

---

Let's construct dfa $\widehat{M} = (Q, \Sigma, \delta, q_0, Q - F)$.

Clearly, $L(\widehat{M}) = \bar{L_1}$. Thus $\bar{L_1}$ is a regular language. QED.

**Proof of $L_1 \cap L_2$**

Strategy: Given two dfas for the two regular languages, construct a new dfa that accepts a string if and only if both original dfas accept the string.

Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for dfas:

$$M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$$

$$M_2 = (P, \Sigma, \delta_2, p_0, F_2)$$

Construct $\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\delta}, (q_0, p_0), \widehat{F})$, where

$$\widehat{Q} = Q \times P$$

$$\widehat{\delta}((q_i, p_j), a) = (q_k, p_l) \textbf{ when } \delta_1(q_i, a) = q_k \quad \delta_2(p_j, a) = p_l$$

$$\widehat{F} = \{(q, p) : q \in F_1, p \in F_2\}$$

Clearly, $\omega \in L_1 \cap L_2$ if and only if $\omega$ accepted by $\widehat{M}$.

Thus, $L_1 \cap L_2$ is regular. QED.

The previous proof is *constructive*.

- It establishes desired result.
- It provides an *algorithm* for building an item of interest (e.g., dfa to accept $L_1 \cap L_2$).

Sometimes nonconstructive proofs are shorter and easier to understand. But they provide no algorithm.

**Alternate (nonconstructive) proof for $L_1 \cap L_2$**

| | |
|---|---|
| | $L_1$ and $L_2$ are regular. |
| $\equiv$ | { previously proved part of Theorem 4.1 } |
| | $\bar{L_1}$ and $\bar{L_2}$ are regular. |
| $\Rightarrow$ | { previously proved part of Theorem 4.1 } |
| | $\bar{L_1} \cup \bar{L}$ is regular |
| $\Rightarrow$ | { previously proved part of Theorem 4.1 } |
| | $\overline{\bar{L_1} \cup \bar{L_2}}$ is regular |
| $\equiv$ | { deMorgan's Law for sets } |
| | $L_1 \cap L_2$ is regular |

QED.

### 4.1.3 Closure under Difference (Linz Example 4.1)

Consider the *difference* between two regular languages $L_1$ and $L_2$, written $L_1 - L_2$.

But this is just set difference, which is defined $L_1 - L_2 = L_1 \cap \bar{L_2}$.

From Theorem 4.1 above, we know that regular languages are closed under both complementation and intersection. Thus, regular languages are closed under difference as well.

### 4.1.4 Closure under Reversal

**Linz Theorem 4.2 (Closure under Reversal):** The family of regular languages is closed under *reversal*.

**Proof (constructive)**

Strategy: Construct an nfa for the regular language and then reverse all the edges and exchange roles of the initial and final states.

Let $L_1$ be a regular language. Construct an nfa $M$ such that $L_1 = L(M)$ and $M$ has a single final state. (We can add $\lambda$ transitions from the previous final states to create a single new final state.)

Now construct a new nfa $\hat{M}$ as follows.

- Make the initial state of $M$ the final state of $\hat{M}$.
- Make the final state of $M$ the initial state of $\hat{M}$.
- Reverse the direction of all edges of $M$ keeping the same labels and add the edges to $\hat{M}$.

Thus nfa $\hat{M}$ accepts $\omega^R \in \Sigma^*$ if and only if the original nfa accepts $\omega \in \Sigma^*$. QED.

### 4.1.5 Homomorphism Definition

In mathematics, a homomorphism is a mapping between two mathematical structures that *preserves* the essential structure.

**Linz Definition 4.1 (Homomorphism):** Suppose $\Sigma$ and $\Gamma$ are alphabets. A function

$$h : \Sigma \to \Gamma^*$$

is called a *homomorphism.*

In words, a homomorphism is a substitution in which a single letter is replaced with a string.

We can extend the domain of a function $h$ to strings in an obvious fashion. If

$$w = a_1 a_2 \ \cdots \ a_n \text{ for } n \geq 0$$

then

$$h(w) = h(a_1)h(a_2) \ \cdots \ h(a_n).$$

If $L$ is a language on $\Sigma$, then we define its *homomorphic image* as

$$h(L) = \{h(w) : w \in L\}.$$

Note: The homomorphism function $h$ *preserves* the essential structure of the language. In particular, it preserves operation concatenation on strings, i.e., $h(\lambda) = \lambda$ and $h(uv) = h(u)h(v)$.

### 4.1.6 Linz Example 4.2

Let $\Sigma = \{a, b\}$ and $\Gamma = \{a, b, c\}$.

Define $h$ as follows:

$$h(a) = ab,$$

$$h(b) = bbc$$

Then $h(aba) = abbbcab$.

The homomorphic image of $L = \{aa, aba\}$ is the language $h(L) = \{abab, abbbcab\}$.

If we have a regular expression $r$ for a language $L$, then a regular expression for $h(L)$ can be obtained by simply applying the homomorphism to each $\Sigma$ symbol of $r$. We show this in the next example.

### 4.1.7 Linz Example 4.3

For $\Sigma = \{a, b\}$ and $\Gamma = \{b, c, d\}$, define $h$:

$$h(a) = dbcc$$

$$h(b) = bdc$$

If $L$ is a regular language denoted by the regular expression

$$r = (a + b^*)(aa)^*$$

then

$$r_1 = (dbcc + (bdc)^*)(dbccdbcc)^*$$

denotes the regular language $h(L)$.

The general result on the closure of regular languages under any homomorphism follows from this example in an obvious manner.

### 4.1.8 Closure under Homomorphism Theorem

**Linz Theorem 4.3 (Closure under Homomorphism)**: Let $h$ be a homomorphism. If $L$ is a regular language, then its homomorphic image $h(L)$ is also regular.

Proof: Similar to the argument in Example 4.3. See Linz textbook for full proof.

The family of regular languages is therefore closed under arbitrary homomorphisms.

### 4.1.9 Right Quotient Definition

**Linz Definition 4.2 (Right Quotient)**: Let $L_1$ and $L_2$ be languages on the same alphabet. Then the *right quotient* of $L_1$ with $L_2$ is defined as

$$L_1/L_2 = \{x : xy \in L_1 \text{ for some } y \in L_2\}$$

### 4.1.10 Linz Example 4.4

Given languages $L_1$ and $L_2$ such that

$$L_1 = \{a^n b^m : n \geq 1, m \geq 0\} \cup \{ba\}$$

$$L_2 = \{b^m : m \geq 1\}$$

Then

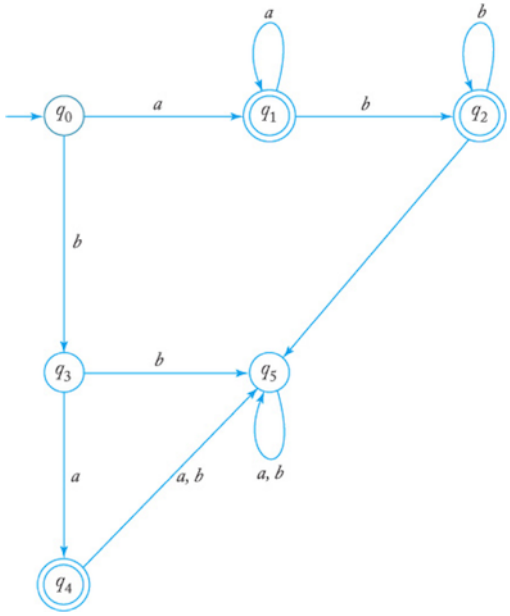
$$L_1/L_2 = \{a^n b^m : n \geq 1, m \geq 0\}.$$

The strings in $L_2$ consist of one or more $b$'s. Therefore, we arrive at the answer by removing one or more $b$'s from those strings in $L_1$ that terminate with at least one $b$ as a suffix.

Note that in this example $L_1$, $L_2$, and $L_1/L_2$ are regular.

Can we construct a dfa for $L_1/L_2$ from dfas for $L_1$ and $L_2$?

Linz Figure 4.1 shows a dfa $M_1$ that accepts $L_1$.




**Linz Fig. 4.1: DFA for Example 4.4 $L_1$**

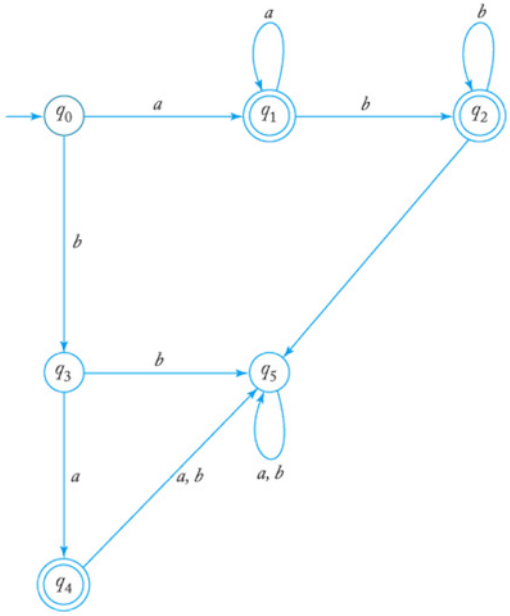
An automaton for $L_1/L_2$ must accept any $x$ such that $xy \in L_1$ and $y \in L_2$.

For all states $q \in M_1$, if there exists a walk labeled $v$ from $q$ to a final state $q_f$ such that $v \in L_2$, then make $q$ a final state of the automaton for $L_1/L_2$.

In this example, we check states to see if there is $bb^*$ walk to any of the final states $q_1$, $q_2$, or $q_4$.

- $q_1$ and $q_2$ have such walks.
- $q_0$, $q_3$, and $q_4$ do not.

The resulting automaton is shown in Linz Figure 4.2.




**Linz Fig. 4.2: DFA for Example 4.4 $L_1/L_2$ EXCEPT $q_4$ NOT FINAL**

The next theorem generalizes this construction.

### 4.1.11 Closure under Right Quotient

**Linz Theorem 4.4 (Closure under Right Quotient)**: If $L_1$ and $L_2$ are regular languages, then $L_1/L_2$ is also regular. We say that the family of regular languages is *closed under right quotient* with a regular language.

**Proof**

Let dfa $M = (Q, \Sigma, \delta, q_0, F)$ such that $L(M) = L_1$.

Construct dfa $\widehat{M} = (Q, \Sigma, \delta, q_0, \widehat{F})$ for $L_1/L_2$ as follows.

For all $q_i \in Q$, let dfa $M_i = (Q, \Sigma, \delta, q_i, F)$. That is, dfa $M_i$ is the same as $M$ except that it starts at $q_i$.

- From Theorem 4.1, we know $L(M_i) \cap L_2$ is regular. Thus we can construct the intersection machine as show in the proof of Theorem 4.1.
- If there is any path in the intersection machine from its initial state to a final state, then $L(M_i) \cap L_2 \neq \emptyset$. Thus $q_i \in \widehat{F}$ in machine $\widehat{M}$.

Does $L(\widehat{M}) = L_1/L_2$?

First, let $x \in L_1/L_2$.

- By definition, there must be $y \in L_2$ such that $xy \in L_1$.
- Thus $\delta^*(q_0, xy) \in F$.
- There must be some $q$ such that $\delta^*(q_0, x) = q$ and $\delta^*(q, y) \in F$.
- Thus, by construction, $q \in \widehat{F}$. Hence, $\widehat{M}$ accepts $x$.

Now, let $x$ be accepted by $\widehat{M}$.

- $\delta^*(q_0, x) = q \in \widehat{F}$.
- Thus, by construction, we know there is a $y \in L_2$ such that $\delta^*(q, y) \in F$.

Thus $L(\widehat{M}) = L_1/L_2$, which means $L_1/L_2$ is regular.

#### 4.1.12 Linz Example 4.5

Find $L_1/L_2$ for

$$L_1 = L(a^*baa^*)$$

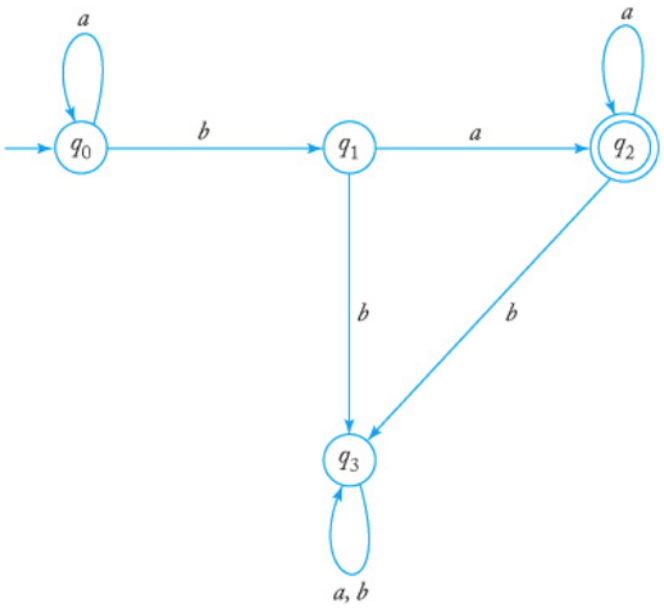
$$L_2 = L(ab^*)$$

We apply the construction (algorithm) used in the proof of Theorem 4.4.

Linz Figure 4.3 shows a dfa for $L_1$.

Let $M = (Q, \Sigma, \delta, q_0, F)$.

Thus if we construct the sequence of machines $M_i$




**Linz Fig. 4.3: DFA for Example 4.5** $L_1$

$$L(M_0) \cap L_2 = \emptyset$$

$$L(M_1) \cap L_2 = \{a\} \neq \emptyset$$

$$L(M_2) \cap L_2 = \{a\} \neq \emptyset$$

$$L(M_3) \cap L_2 = \emptyset$$

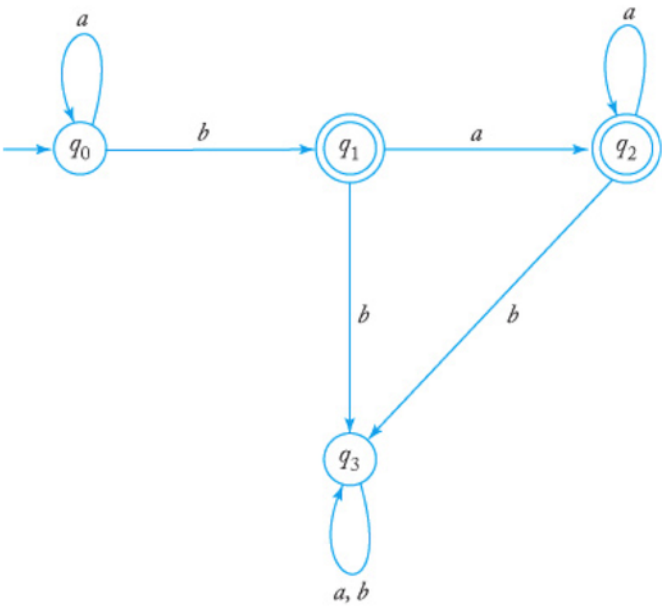
then the resulting dfa for $L_1/L_2$ is shown in Linz Figure 4.4.

The automaton shown in Figure 4.4 accepts the language denoted by the regular expression

$$a^*b + a^*baa^*$$

which can be simplified to

$$a^*ba^*$$




**Linz Fig. 4.4: DFA for Example 4.5** $L_1/L_2$

## 4.2 Elementary Questions about Regular Languages

### 4.2.1 Membership?

Fundamental question: Is $w \in L$?

It is difficult to find a *membership* algorithm for languages in general. But it is relatively easy to do for regular languages.

A regular language is given in a *standard representation* if and only if described with one of:

- a dfa or nfa
- a regular expression
- a regular grammar

**Linz Theorem 4.5 (Membership)**: Given a standard representation of any regular language $L$ on $\Sigma$ and any $w \in \Sigma^*$, there exists an algorithm for determining whether or not $w$ is in $L$.

**Proof**

We represent the language by some dfa, then test $w$ to see if it is accepted by this automaton. QED.

### 4.2.2 Finite or Infinite?

**Linz Theorem 4.6 (Finiteness)**: There exists an algorithm for determining whether a regular language, given in standard representation, is empty, finite, or infinite.

**Proof**

Represent $L$ as a transition graph of a dfa.

- If simple path exists from the initial state to any final state, then it is *not empty*. Otherwise, it is *empty*.
- If any vertex on a cycle is in a path from the initial state to any final state, then the language is *infinite*. Otherwise, it is *finite*.

QED.

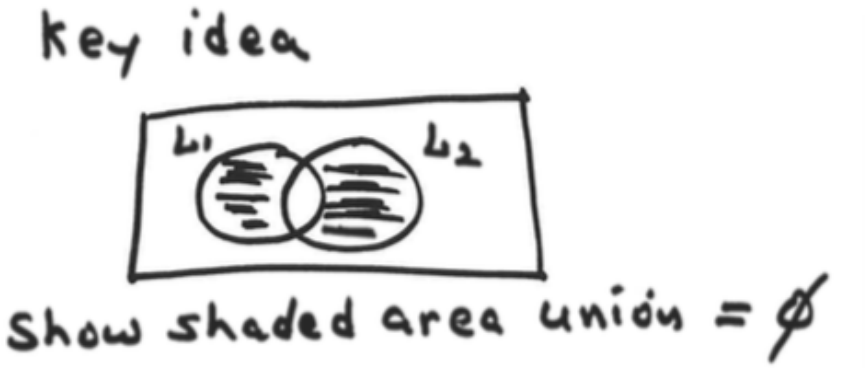
### 4.2.3 Equality?

Consider the question $L_1 = L_2$?

This is practically important. But it is a difficult issue because there are many ways to represent $L_1$ and $L_2$.

**Linz Theorem 4.7 (Equality)**: Given a standard representation of two regular languages $L_1$ and $L_2$, there exists an algorithm to determine whether or whether not $L_1 = L_2$.

**Proof**




Let $L_3 = (L_1 \cap \bar{L_2}) \cup (\bar{L_1} \cap L_2)$.

By closure, $L_3$ is regular. Hence, there is a dfa $M$ that accepts $L_3$.

Because of Theorem 4.6, we can determine whether $L_3$ is empty or not.

But from Excerise 8, Section 1.1, we see that $L_3 = \emptyset$ if and only if $L_1 = L_2$. QED.

## 4.3 Identifying Nonregular Languages

A regular languages may be *infinite*

- but it is accepted by an automaton with *finite* "memory"
- which imposes restrictions on the language.

In processing a string, the amount of information that the automaton must "remember" is strictly limited (finite and bounded).

### 4.3.1 Using the Pigeonhole Principle

In mathematics, the *pigeonhole principle* refers to the following simple observation:

> If we put $n$ objects into $m$ boxes (pigeonholes), and, if $n > m$, at least one box must hold more than one item.

This is obvious, but it has deep implications.

### 4.3.2 Linz Example 4.6

Is the language $L = \{a^n b^n : n \geq 0\}$ regular?

The answer is *no*, as we show below.

**Proof that $L$ is not regular**

Strategy: Use proof by contradiction. Assume that what we want to prove is false. Show that this introduces a contradiction. Hence, the original assumption must be true.

Assume $L$ is regular.

Thus there exists a dfa $M = (Q, \{a, b\}, \delta, q_0, F)$ such that $L(M) = L$.

Machine $M$ has a specific number of states. However, the number of $a$'s in a string in $L(M)$ is finite but *unbounded* (i.e., no maximum value for the length). If $n$ is larger than the number of states in $M$, then, according to the pigeonhole principle, there must be some state $q$ such that

$$\delta^*(q_0, a^n) = q$$

and

$$\delta^*(q_0, a^m) = q$$

with $n \neq m$. But, because $M$ accepts $a^n b^n$,

$$\delta^*(q, b^n) = q_f \in F$$

for some $q_f \in F$.

From this we reason as follows:

| | |
|---|---|
| | $\delta^*(q_0, a^m b^n)$ |
| $=$ | $\delta^*(\delta^*(q_0, a^m), b^n)$ |
| $=$ | $\delta^*(q, b^n)$ |
| $=$ | $q_f$ |

But this contradicts the assumption that $M$ accepts $a^m b^n$ only if $n = m$. Therefore, $L$ cannot be regular. QED

We can use the pigeonhole principle to make "finite memory" precise.

### 4.3.3 Pumping Lemma for Regular Languages

**Linz Theorem 4.8 (Pumping Lemma for Regular Languages)**: Let $L$ be an infinite regular language. There exists some $m > 0$ such that any $w \in L$ with $|w| \geq m$ can be decomposed as

$$w = xyz$$

with

$$|xy| \leq m$$

and

$$|y| \geq 1$$

such that

$$w_i = xy^i z$$

is also in $L$ for all $i \geq 0$.

That is, we can break every sufficiently long string from $L$ into three parts in such a way that an arbitrary number of repetitions of the middle part yields another string in $L$.

We can "pump" the middle string, which gives us the name *pumping lemma* for this theorem.

**Proof**

Let $L$ be an infinite regular language. Thus there exists a dfa $M$ that accepts $L$. Let $M$ have states $q_0, q_1, q_2, \cdots\ q_n$.

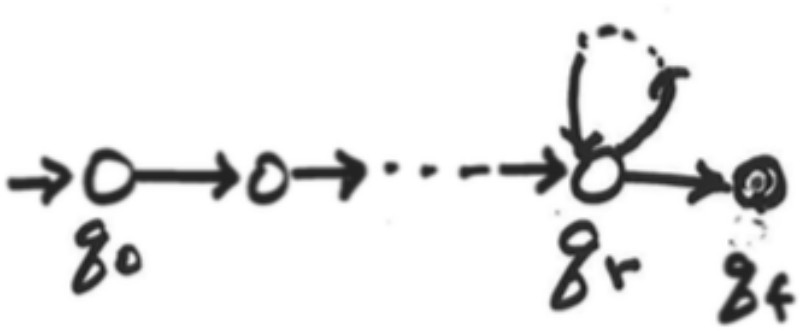
Consider a string $w \in L$ such that $|w| \geq m = n + 1$. Such a string exists because $L$ is infinite.

Consider the set of states $q_0, q_i, q_j, \cdots\ q_f$ that $M$ traverses as it processes $w$.

The size of this set is exactly $|w| + 1$. Thus, according to the pigeonhole principle, at least one state must be repeated, and such a repetition must start no later than the $n$th move.

Thus the sequence is of the form

$$q_0, q_i, q_j, \cdots, q_r, \cdots, q_r, \cdots, q_f.$$




Then there are substrings $x$, $y$, and $z$ of $w$ such that

$$\delta^*(q_0, x) = q_r$$

$$\delta^*(q_r, y) = q_r$$

$$\delta^*(q_r, z) = q_f$$

with $|xy| \leq n + 1 = m$ and $|y| \geq 1$. Thus, for any $k \geq 0$,

$$\delta^*(q_0, xy^k z) = q_f$$

QED.

We can use the pumping lemma to show that languages are *not regular*. Each of these is a proof by contradiction.

### 4.3.4 Linz Example 4.7

Show that $L = \{a^n b^n : n \geq 0\}$ is not regular.

Assume that $L$ is regular, so that the Pumping Lemma must hold.

If, for some $n \geq 0$ and $i \geq 0$, $xyz = a^n b^n$ and $xy^i z$ are both in $L$, then $y$ must be all $a$'s or all $b$'s.

We do not know what $m$ is, but, whatever $m$ is, the Pumping Lemma enables us to choose a string $w = a^m b^m$. Thus $y$ must consist entirely of $a$'s.

Suppose $k > 0$. We must decompose $w = xyz$ as follows for some $p + k \leq m$:

$$x = a^p$$

$$y = a^k$$

$$z = a^{m-p-k} b^m$$

From the Pumping Lemma

$$w_0 = a^{m-k} b^m.$$

Clearly, this is not in $L$. But this contradicts the Pumping Lemma.

Hence, the assumption that $L$ is regular is false. Thus $\{a^n b^n : n \geq 0\}$ is not regular.

### 4.3.5 Using the Pumping Lemma (Viewed as a Game)

The Pumping Lemma guarantees the *existence* of $m$ and decomposition $xyz$ for any string in a regular language.

- But we *do not know* what $m$ and $xyz$ are.
- We *do not have contradiction* if the Pumping Lemma is violated for some specific $m$ or $xyz$.

The Pumping Lemma holds for all $w \in L$ and for all $i \geq 0$ (i.e., $xy^iz \in L$ for all $i$).

- We *do have a contradiction* if the Pumping Lemma is violated for some $w$ or $i$.

We can thus conceptualize a proof as a game against an opponent.

- Our goal: Establish a contradiction of the Pumping Lemma.
- Opponent's goal: Stop us.
- Moves:
  1. The opponent picks $m$.
  2. Given $m$, we pick a string $w$ in $L$ of length equal or greater than $m$. We are free to choose any $w$, subject to requirement $w \in L$ and $|w| \geq m$.
  3. The opponent chooses the decomposition $xyz$, subject to $|xy| \leq m$ and $|y| \geq 1$. We have to assume that the opponent makes the choice that will make it hardest for us to win the game.
  4. We try to pick $i$ in such a way that the pumped string $w_i$, as defined in $w_i = xy^iz$, is not in $L$. If we can do so, we win the game.

Strategy:

- Choose $w$ in step 2 carefully. So that, regardless of the $xyz$ choice, contradiction can be established.

### 4.3.6 Linz Example 4.8

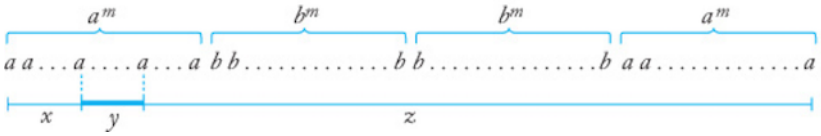
Let $\Sigma = \{a, b\}$. Show that

$$L = \{ww^R : w \in \Sigma^*\}$$

is not regular.

We use the Pumping Lemma and assume $L$ is regular.

Whatever $m$ the opponent picks in step 1 (of the "game"), we can choose a $w$ as shown below in step 2.




**Linz Fig. 4.5**

Because of this choice, and the requirement that $|xy| \leq m$, in step 3 the opponent must choose a $y$ that consists entirely of $a$'s. Consider

$$w_i = xy^iz$$

that must hold because of the Pumping Lemma.

In step 4, we use $i = 0$ in $w_i = xy^iz$. This string has fewer $a$'s on the left than on the right and so cannot be of the form $ww^R$.

Therefore, the Pumping Lemma is violated. $L$ is not regular.

Warning: Be careful! There are ways we can go wrong in applying the Pumping Lemma.

- If we choose $w$ too short in step 2 of this example (i.e., where the first $m$ symbols include two or more $b$'s), then the opponent can choose a $y$ having an even number of $b$'s. In that case, we could not have reached a violation of the pumping lemma on the last stap.
- If we choose a string $w$ consisting of all $a$'s, say

  $$w = a^{2m}$$

  which is in $L$. To defeat us, the opponent need only pick

  $$y = aa$$

Now $w_i$ is in $L$ for all $i$, and we lose. ˆ

- We must assume the opponent does not make mistakes. If, in the case where we pick $w = a^{2m}$, the opponent picks

  $$y = a$$

  then $w_0$ is a string of odd length and therefore not in $L$. But any argument is incorrect if it assumes the opponent fails to make the best possible choice (i.e., $y = aa$).

#### 4.3.7 Linz Example 4.9

For $\Sigma = \{a, b\}$, show that the language

$$L = \{w \in \Sigma^* : n_a(w) < n_b(w)\}$$

is not regular.

We use the Pumping Lemma to show a contradiction. Assume $L$ is regular.

Suppose the opponent gives us $m$. Because we have complete freedom in choosing $w \in L$, we pick $w = a^m b^{m+1}$. Now, because $|xy|$ cannot be greater than $m$, the opponent cannot do anything but pick a $y$ with all $a$'s, that is,

$$y = a^k \text{ for } 1 \leq k \leq m.$$

We now pump up, using $i = 2$. The resulting string

$$w_2 = a^{m+k} b^{m+1}$$

is not in $L$. Therefore, the Pumping Lemma is violated. $L$ is not regular.

#### 4.3.8 Linz Example 4.10

Show that

$$L = \{(ab)^n a^k : n > k, k \geq 0\}$$

is not regular

We use the Pumping Lemma to show a contradiction. Assume $L$ is regular.

Given some $m$, we pick as our string

$$w = (ab)^{m+1}a^m$$

which is in $L$.

The opponent must decompose $w = xyz$ so that $|xy| \leq m$ and $|y| \geq 1$. Thus both $x$ and $y$ must be in the part of the string consisting of $ab$ pairs. The choice of $x$ does not affect the argument, so we can focus on the $y$ part.

If our opponent picks $y = a$, we can choose $i = 0$ and get a string not in $L((ab)^*a^*)$ and, hence, not in $L$. (There is a similar argument for $y = b$.)

If the opponent picks $y = ab$, we can choose $i = 0$ again. Now we get the string $(ab)^m a^m$, which is not in $L$. (There is a similar argument for $y = ba$.)

In a similar manner, we can counter any possible choice by the opponent. Thus, because of the contradiction, $L$ is not regular.

### 4.3.9 Linz Example (Factorial Length Strings)

Note: This example is adapted from an earlier edition of the Linz textbook.

Show that

$$L = \{a^{n!} : n \geq 0\}$$

is not regular.

We use the Pumping Lemma to show a contradiction. Assume $L$ is regular.

Given the opponent's choice for $m$, we pick $w$ to be the string $a^{m!}$ (unless the opponent picks $m < 3$, in which case we can use $a^{3!}$ as $w$).

The possible decompositions $w = xyz$ (such that $|xy| \leq m$) differ only in the lengths of $x$ and $y$. Suppose the opponent picks $y$ such that

$$|y| = k \leq m.$$

According to the Pumping Lemma, $xz = a^{m!-k} \in L$. But this string can only be in $L$ if there exists a $j$ such that

$$m! - k = j!.$$

But this is impossible, because for $m \geq 3$ and $k \leq m$ we know (see argument below) that

$$m! - k > (m-1)!.$$

Therefore, $L$ is not regular.

Aside: To see that $m! - k > (m-1)!$ for $m \geq 3$ and $k \leq m$, note that

$$m! - k \geq m! - m = m(m-1)! - m = m((m-1)! - 1) > (m-1)!.$$

### 4.3.10 Linz Example 4.12

Show that the language

$$L = \{a^n b^k c^{n+k} : n \geq 0, k \geq 0\}$$

is not regular.

Strategy: Instead of using the Pumping Lemma directly, we show that $L$ is related to another language we already know is nonregular. This may be an easier argument.

In this example, we use the closure property under homomorphism (Linz Theorem 4.3).

Let $h$ be defined such that

$$h(a) = a, h(b) = a, h(c) = c.$$

Then

| | | |
|---|---|---|
| $h(L)$ | $=$ | $\{a^{n+k}c^{n+k} : n + k \geq 0\}$ |
| | $=$ | $\{a^i c^i : i \geq 0\}$ |

But we proved this language was not regular in Linz Example 4.6. Therefore, because of closure under homomorphism, $L$ cannot be regular either.

**Alternative proof by contradiction**

Assume $L$ is regular.

Thus $h(L)$ is regular by closure under homomorphism (Linz Theorem 4.3).

But we know $h(L)$ is not regular, so there is a contradiction.

Thus, $L$ is not regular.

### 4.3.11 Linz Example 4.13

Show that the language

$$L = \{a^n b^l : n \neq l\}$$

is not regular.

We use the Pumping Lemma, but this example requires more ingenuity to set up than previous examples.

Assume $L$ is regular.

Choosing a string $w \in L$ with $m = n = l + 1$ or $m = n = l + 2$ will not lead to a contradiction.

In these cases, the opponent can always choose a decomposition $w = xyz$ (with $|xy| \leq m$ and $|y| \geq 1$) that will make it impossible to pump the string out of the language (that is, pump it so that it has an equal number of $a$'s and $b$'s). For $w = a^{l+1}b^l$, the opponent can chose $y$ to be an even number of $a$'s. For $w = a^{l+2}b^l$, the opponent can chose $y$ to be an odd number of $a$'s greater than 1.

We must be more creative. Suppose we choose $w \in L$ where $n = m!$ and $l = (m + 1)!$.

If the opponent decomposes $w = xyz$ (with $|xy| \leq m$ and $|y| = k \geq 1$), then $y$ must consist of all $a$'s.

If we pump $i$ times, we generate string $xy^iz$ where the number of $a$'s is $m!+(i-1)k$

We can contradict the Pumping Lemma if we can pick $i$ such that

$$m! + (i - 1)k = (m + 1)!.$$

But we can do this, because it is always possible to choose

$$i = 1 + mm!/k.$$

For $1 \leq k \leq m$, the expression $1 + mm!/k$ is an integer.

Thus the generated string has $m! + ((1 + mm!/k) - 1)k$ occurrences of $a$.

$$\begin{array}{ll} & m! + ((1 + mm!/k) - 1)k \\ = & m! + mm! \\ = & m!(m + 1) \\ = & (m + 1)! \end{array}$$

This introduces a contradiction of the Pumping Lemma. Thus $L$ is not regular.

**Alternative argument (more elegant)**

Suppose $L = \{a^n b^l : n \neq l\}$ is regular.

Because of complementation closure, $\bar{L}$ is regular.

Let $L_1 = \bar{L} \cap L(a^*b^*)$.

But $L(a^*b^*)$ is regular and thus, by intersection closure, $L_1$ is also regular.

But $L_1 = \{a^n b^n : n \geq 0\}$, which we have shown to be nonregular. Thus we have a contradiction, so $L$ is not regular.

### 4.3.12 Pitfalls in Using the Pumping Lemma

The Pumping Lemma is difficult to understand and, hence, difficult to *apply*.

Here are a few suggestions to avoid pitfalls in use of the Pumping Lemma.

- Do not attempt to use the Pumping Lemma to show a language is regular. Only use it to show a language is *not regular*.
- Make sure you start with a string that is in the language.
- Avoid invalid assumptions about the decomposition of a string $w$ into $xyz$. Use only that $|xy| \leq m$ and $|y| \geq 1$.

Like most interesting "games", knowledge of the rules for use of the Pumping Lemma is necessary, but it is not sufficient to become a master "player". To master the use of the Pumping Lemma, one must work problems of various difficulties. Practice, practice, practice.