4.5. Languages and Finite Automata

Note. This section addresses, in a general sense, language, the solvability of problems, and the complexity of problems. The main concern is the limitations of the language in which we are working (by it mathematical language or a programming language). We will take a language to be a collection of strings of symbols from some alphabet. This is addressed at this stage of the course because it is approached with the tools of sets and cardinality.

Definition 4.45/4.47. An *alphabet* Σ is a finite nonempty set. For nonnegative integer n, a *word* (or *string*) w of *length* n over the alphabet Σ is a function $w : \mathbb{N} \to \Sigma$.

Note/Definition. If word w maps $i \overset{w}{\mapsto} a_i$ for $1 \leq i \leq n$, then we denote w as $w = a_1 a_2 \dots a_n$. When $n = 0$ we have $w = \varnothing$ (because $\mathbb{N}_0 = \varnothing$) and w is the *empty word*, often denoted ε. The set of all words over Σ, including the empty word, is denoted Σ^*. For word $w \in \Sigma^*$, the length of w is denoted $|w|$.

Example 4.46/4.48. Examples of alphabets are $\{\spadesuit\}$, $\{a, b, c, \dots, z\}$, and $\{0, 1\}$. Some words of length three over these alphabets, respectively, are $\spadesuit\spadesuit\spadesuit$, zap, and 001.

Theorem 4.49. If Σ is an alphabet then Σ^* is countably infinite.

Definition 4.50. Let Σ be an alphabet. A subset $L \subseteq \Sigma^*$ is a *language* over Σ.

Theorem 4.51. Let Σ be any alphabet. Then there are uncountably many languages over Σ.

Note. Suppose we want to "describe" a language. We do so using a finite numbers of words (to describe it in a familiar way; we could also introduce a space and other punctuation as additional letters in the alphabet and then the whole description is simply a single, but long, word). But for alphabet K, the set of words K^* is countable (Theorem 4.49). However, the number of languages is uncountable (Theorem 4.51). So not all languages are describable. So the plan is to describe some languages, hopefully some of the useful ones! In this section, we imagine a mathematical model of a computer that reads strings over a given alphabet and "accepts" them or not according to whether they satisfy certain criteria (think of this as the computer determining if the statement is in some given language or not).

Note. We now describe the model of a machine designed for language recognition. Suppose it allows for the input of letters and words (through a keyboard, say). When the machine is first turned on, it is in its *initial state*, denoted q_0. The condition of the machine as it reacts to inputs determines the *state* of the machine at various times, and we denote the finite set of all states as S. As a symbol from Σ is input into the machine, its state changes depending on the symbol and

the current state of the machine. For the current state $q \in S$ and the entry of the symbol $\sigma \in \Sigma$, the ordered pair (q, σ) determines a unique new state denoted $\delta(q, \sigma)$ ("in a deterministic way," as Gerstein says on page 179). For language recognition, we require that there is a subset $F \subseteq S$ such that whenever the machine is in a state in set F, the machine has *accepts* the data entered. We formalize this in the next definition.

Definition 4.52. A *finite automaton* (or *finite-state machine*) is a five-tuple $M = (S, \Sigma, \delta, q_0, F)$ where

S is a finite nonempty set (the set of *state*);

Σ is a finite nonempty set (the *alphabet*);

δ is a function from $S \times \Sigma$ to S (the *transition function*);

$q_0 \in S$ (the *initial state*);

F is a subset of S (the set of *final state*).

Example 4.53(a). Consider an automaton M that reads alphabet $\Sigma = \{a, b\}$ and suppose the states are $S = \{q_0, q_1\}$. Let q_0 the the initial state and the only final state (so that $F = \{q_0\}$). Suppose that reading a changes the state of the machine (an unambiguous description, since there are only two states). Suppose that reading b does not change the state. Then the transition function δ can be described by the table given below. The current state q is given in the column on the left, the input σ is given in the row on the top, and the corresponding entry in the table is the new state $\delta(q, \sigma)$.

q \ σ	a	b
q_0	q_1	q_0
q_1	q_0	q_1

Example 4.53(b). This is a bit of a silly example, but it is more complicated that the example in 4.53(a). Quoting from Gerstein (page 180):

> "David wakes up in the morning and checks the weather. If it is sunny he goes fishing; if it is rainy he goes to the movies. He is happy if he sees a good movie or if he finds that the fish are biting; otherwise he is sad. Not every input condition will change David's state; for example, if he has gone to the movies or if he has just awakened, then he is unaffected by the activities of the fish. Once David's mood is set, it doesn't change. His criterion for 'accepting' the day is that the events make him happy."

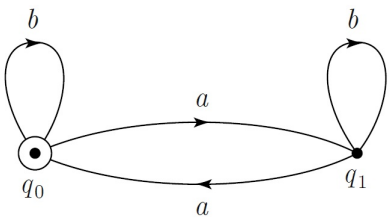
Let $M = (S, \Sigma, \delta, q_0, F)$ be the automaton representing David's situation, where

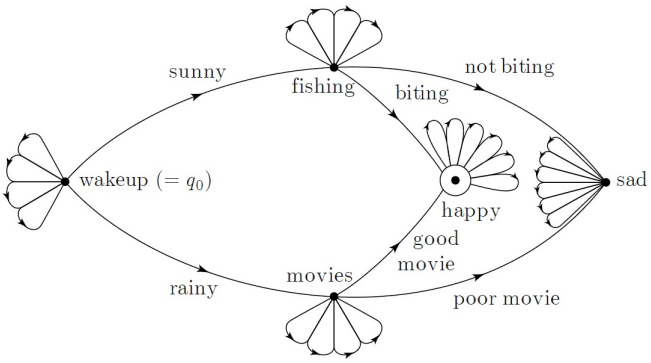
$$\begin{aligned} S &= \{\text{wakeup}, \text{fishing}, \text{movies}, \text{happy}, \text{sad}\} \\ \Sigma &= \{\text{sunny}, \text{rainy}, \text{biting}, \text{not biting}, \text{good movie}, \text{poor movie}\} \\ q_0 &= \text{wakeup} \\ F &= \{\text{happy}\} \end{aligned}$$

Notice that we are taking words here as the letters in alphabet Σ. We get the table for the transition function δ as given below.

q \ σ	sunny	rainy	biting	not biting	good move	poor movie
wakeup	fishing	movies	wakeup	wakeup	wakeup	wakeup
fishing	fishing	fishing	happy	sad	fishing	fishing
movies	movies	movies	movies	movies	happy	sad
happy	happy	happy	happy	happy	happy	happy
sad	sad	sad	sad	sad	sad	sad

Note/Definition. An easier way to visualize an automaton than the tables given in Example 4.53 is to consider a directed graph which represents the transitions. For automaton M, we create the directed graph, called the *transition diagram* or the *state graph*, by defining the *vertices* (or *nodes*) as the states in set S, and the *arcs* (or *directed edges*) are labeled with the symbols from Σ. We have an arc σ from vertex q_1 to vertex q_2 if M goes from state q_1 to state q_2 when σ is entered (that is, when $\delta(q_1, \sigma) = q_2$). We circle the vertices representing the final states. We then get the following transition diagrams for the two automata considered in Example 4.53 (where labels are omitted on arcs that do not cause a state change in the diagram for Example 4.53(b)).





Note. Previously, we considered a transition function δ based on the entry of a single symbol σ. We now extend this to transition functions of several symbols (namely, of words in Σ^*), so that δ maps $S \times \Sigma^* \to S$. This is really just dealt with by recursion using the transition function. When σ_1 is input then the machine goes from state q to state $\delta(q, \sigma_1)$. Then when σ_2 is input the machine goes to state $\delta(\delta(q, \sigma_1), \sigma_2)$, denoted $\delta(q, \sigma_1\sigma_2)$, and so forth. so we have the recursive definition $\delta(q, \sigma_1\sigma_2 \cdots \sigma_k\sigma_{k+1}) = \delta(\delta(q, \sigma_1\sigma_2 \cdots \sigma_k), \sigma_{k+1})$. To complete the recursive definition of $\delta : S \times \Sigma^* \to S$, we define $\delta(q, \varepsilon) = q$ for all $q \in S$ (so no input results in no change of state).

Definition 4.57. Let $M = (S, \Sigma, \delta, q_0, F)$ be a finite automaton, and let $w \in \Sigma^*$. We say that M *accepts* w if $\delta(q_0, w) \in F$. Define the *language accepted by* M to be the language $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

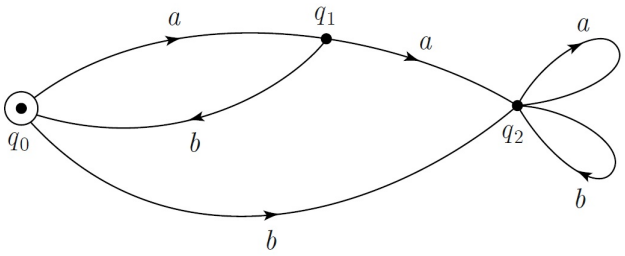
Note. Notice from the previous definition that if a word w results in M going from its initial state q_0 to some final state in F, then we have that "M accepts w." You might think of the unacceptable words as those for which the machine reaches no conclusion.

Example 4.58. (a) Let M be automaton of Example 4.53(a) (with the transition diagram given above). Since input b does not change the state, but input a changes the state from q_0 to q_1 (or q_1 to q_0), then any word with an even number of a's will result in the machine in the final state q_0 (remember, M starts in state q_0). So the

language accepted by M, $L(M)$, is the set of all words with an even number of a's.

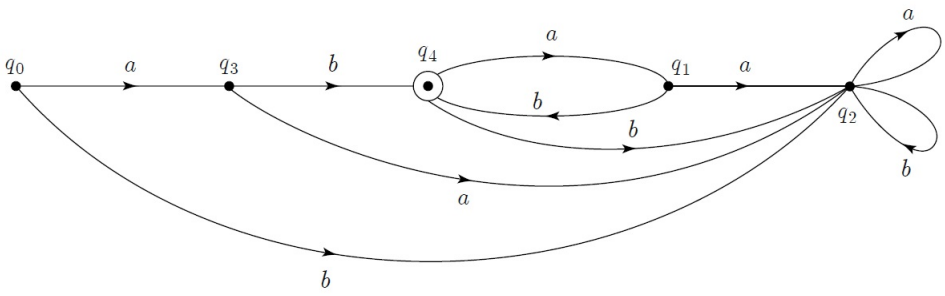
(c) At the extremes, consider $M = (S, \Sigma, \delta, q_0, F)$. If $F = S$ then $L(M) = \Sigma^*$ since every state is a final state. If $F = \varnothing$, then $L(M) = \varnothing$ (since there is no final state).

(d) Let $\Sigma = \{a, b\}$ and suppose M has the following transition diagram:



A word in $L(M)$, other then $\varnothing$, must end with b to reach state $q_0 \in F$ from state q_1. The only way to reach q_0 if from q_1 having read symbol b, and the only way to reach q_1 is from q_0 having read symbol a. So the only words in the language accepted by M, $L(M)$, are $\varnothing$ and words starting with a, ending with b and alternating between a's and b's: $ababab \cdots abab$.

(e) With this example, we take on the problem of finding machine M such that the language it accepts is given. We take as $L(M)$ the set of words accepted in (d), *except* for the empty string. We start with the transition diagram from (d), but instead of having q_0 as a final state, we label the final state as q_4 (in place of the location of q_0 in (d)) and introduce two new vertices q_0 and q_3:



Starting at q_0, we see that if we have two consecutive a's then we get "trapped" at state q_2 (as is the case if we start with b). So we must start with ab which puts M in state q_4 where it then accepts the words accepted in (d). That is, this is a transition diagram of a machine M with $L(M)$ consisting precisely of all words starting with a, ending with b and alternating between a's and b's, $ababab\cdots abab$ (and excludes the word $\varnothing$), as desired.

Note. In any transition diagram, for each vertex there must be exactly on arc emanating from it corresponding to each symbol. If we have a transition diagram for a language, then it is easy to test a particular string to see if it is in $L(M)$. We now give a name to those languages which are "realizable" as the language accepted by some machine M.

Definition 4.59. A language L is *regular* if $L = L(M)$ for some finite automaton M.

Note. Some regular languages are given in Example 4.58. It can be shown that every finite language is regular. This is called Kleene's Theorem, which follows from Kleene's algorithm given in Stephen C. Kleene's "Representation of Events in Nerve Nets and Finite Automate," *Automata Studies, Annals of Mathematical Studies* Princeton University Press **34**, 1–44 (1956); see Section 9, accessed 2/13/2022. Exercise 4.5.5 gives some idea of he technique of proof. An infinite language may not be regular. The Pumping Lemma (Lemma 4.60, below) is useful in proving that some languages are not regular. We need one more definition.

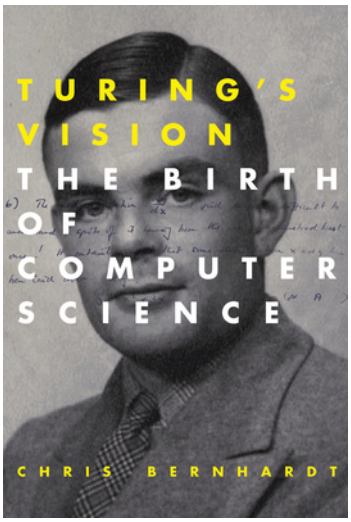
Definition. If Σ is an alphabet and x and y are words in Σ^*, then the *concatenation* of x and y, denoted xy, is the word obtained by following x with y.

Lemma 4.60. Pumping Lemma.
Let Σ be an alphabet, and let $L \subseteq \Sigma^*$ be an infinite regular language. Then there are strings x, y, z in Σ^*, with $y \neq \varepsilon$, such that for every $n \in \mathbb{N}$ the string $xy^n z$ belongs to L.

Example 4.61. (a) Consider the language of words of $\Sigma = \{a, b\}$, consisting of all strings formed by following a string of a's with a string of b's of the same length: $L = \{a^n b^n \mid n > 0\}$. We claim that L is *not* regular and show this by contradiction. ASSUME L is regular. Then by the Pumping Lemma (Lemma 4.60) there exist strings $a, y, z \in \{a, b\}^*$, with $y \neq \varnothing$, such that $xy^n z \in L$ for all $n > 0$. Now y cannot be a string of only a's, since then the string $xy^n z$ would have more a's than b's when $n > 1$, contradicting the definition of L. Similarly, y cannot be a string of b's only. If y has both a's and b's, then $xy^2 z$ has some b's preceding some a's and so $xy^2 z$ is not in L. Each possibility produces a CONTRADICTION, so the assumption that L is regular is false and hence L is not regular.

Note. According to Gerstein (page 187): "...automata also have important applications in other aspects of computer science, particularly in the analysis of algorithms and in the study of the field called *computational complexity*."

Note. Chris Bernhardt's popular-level book, *Turing's Vision: The Birth of Computer Science*, MIT Press (2016), gives descriptions of finite automata, the λ-calculus, and Turing machines. In his Chapter 3, "Finite Automata," he uses much the same notation as we have seen here to describe several examples of finite automata. He considers strings of 0's and 1's and gives a finite automata that (1) accept a string containing an odd number of 1's, (2) accept a string that ends with "01," and (3) accept a string containing and even number of 1's (to illustrate how the negation of a question can be addressed). He also give two examples of problems that cannot be solved by finite automata. One is the question of whether a binary string has the same number of 0's and 1's. The second example, related to the last example (though requiring a slightly different alphabet), is the consideration a string of left and right parentheses to see if they are properly balanced. For example, in "((()(()))" the parentheses are balanced, but in "())(" they are not (even though in "())(" there are the same number of left and right parentheses).



For more information on Turing machines, see Bernhardt's Chapter 4 and my online notes on Computational Complexity. Computational Complexity is not an ETSU Department of Mathematics and Statistics class (though the Department of

Computer Science has a class "Formal Languages and Computational Complexity," CSCI 5610); these online notes (in preparation as of summer 2022) are meant for self-study and as a supplement to the graph-theoretic algorithm material of Mathematical Modeling Using Graph Theory (MATH 5870).

Revised: 7/4/2022