Regular Languages

We will introduce a number of classes of languages. One of the simplest, and most important, classes is the class of *regular* languages. Many classes are defined by the complexity of their membership problems. A regular language is one whose membership problem can be decided by a machine with finitely many states, meaning finite memory.

Abstract Machines

An abstract machine is a mathematical object. A physical machine, such as your laptop, can be modeled by an abstract machine, but an abstract machine might not be equivalent to any physical machine. In this course, all machines are abstract.

An abstract machine M has *states* and operates by *steps.* It can read *input* and write *output.* It has an initial state. At each step, M

- reads input, which must be a finite (possibly empty) string over an alphabet Σ, the input alphabet of M.
- changes state.
- writes output, which is a (possibly empty) string
- possibly halts.

We say a machine M *accepts* a language L is, given an input string w, M **may** eventually halt if and only if $w \in L$.

We say a machine M *decides* a language L if, given an input string w, M always halts, and outputs "1" if $w \in L$, and "0" if $w \notin L$.

A machine could have infinitely many states, sometimes called configurations. Some machines have finitely many states: there are called *finite automata.* (Plural automata, singular outomaton.)

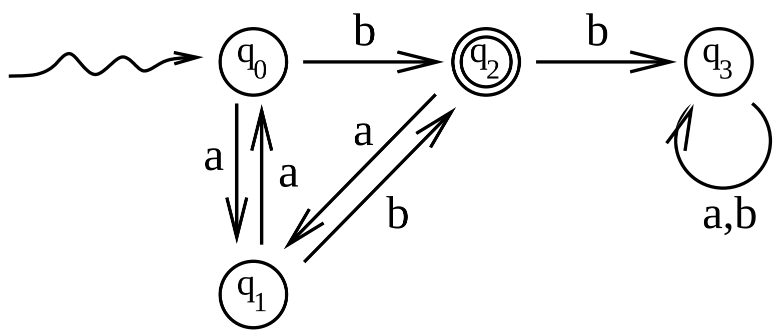
A language is *regular* if and only if it is be decided by a finite automaton. It is also true that a language is regular if and only if it is *accepted* by some finite automaton. This is because every non-deterministic finite automaton, or NFA, is equivalent to some deterministic finite automaton, or DFA.

In other words, the membership problem of L can be answered by a machine with only finite memory.

Not Regular. Let L be the language of all strings over $\{a, b\}$ which have equal numbers of each symbol. For example, the empty string ε is a member of L, as are $ab, ba, aabb, abab, abba$. L is infinite. How would you write a program to decide L? Your program would read a string w one at a time, and would maintain a number m. Initially $m = 0$. e Each time M read an a, it would increment m, and each time it read a b it would decrement m. When the input is fully processed, M announces that $w \in L$ if $m = 0$, and that $w \notin L$ if $w \neq 0$.

The problem is that the program would have to be able to store arbitrarily large integers. Since m could have any length, storing m could overflow M's finite memory.

Example



This example illustrates a DFA, which we call M. M has four states, q_0, q_1, q_2, and q_3. At the start of a computation, M is in the start state, q_0. The input for a computation is a string $w \in \Sigma^*$, where $\Sigma = \{a, b\}$. A step of M is defined as follows.

1. If the input file is empty, jump to step 4. otherwise, proceed to step 2.
2. Read the first symbol in the input, then proceed to step 3.
3. Move to a new state by following the arrow whose label is the symbol just read, then jump to step 1.
4. If the current state is *final* (indicated by a double circle) halt and accept. (That is, output "1.") Otherwise, halt and reject. (That is, output "0.")

M has a dead state, namely q_3. If a computation reaches a dead state, the string will be rejected regardless of the remaining symbols.

You can walk through a computation with a given input string. You can determine that M accepts b, ab, aab, and $abab$. We write $L(M)$ to mean the language accepted by M. Can you describe $L(M)$ in a few words?

Two machines M_1 and M_2 are *equivalent* if $L(M_1) = L(M_2)$. Can you find a DFA with fewer states that is equivalent to M?

Operations on Languages

Operations on languages include union, intersection, concatenation, Kleene closure, complementation, and homomorphism. You already know about union and intersection; remember that a language is a set. However, we frequently write union of languages with "+" rather than "$\cup$"

Concatenation. If u and v are strings, we write uv to indicate concatenation. For example, if $u = abaa$ and $v = ba$, then $uv = abaaba$.

If L_1 and L_2 are languages, we write L_1L_2 for the concatenation of those languages, defined to be the set of all concatenations of a member of L_1 with a member of L_2.

For example, let $L_1 = \{a, ba, bab\}$ and $L_2 = a, b, bb$. Then $L_1L_2 = \{aa, ab, abb, baa, bab, babb, baba, babbb\}$.

We let L^2 be the concatenation LL. We define L^3, L^4, similarly.

Questions

1. Since $|L_1| = |L_2| = 3$, we would expect that $|L_1L_2| = 9$. But it's only 8. Why?
2. Recall that $\emptyset$ is the empty language. If L is some language, what is the concatenation $\emptyset L$?
3. Let $L_1 = \{\lambda\}$. the language consisting of only the empty string. If L_2 is some other language, what is the concatenation L_1L_2?
4. Given two languages L_1 and L_2, is the equation $L_1L_2 = L_2L_1$ always true?
5. What is L^0?
6. Is the equation $L_1(L_2 + L_3) = SL_1L_2 + L_1L_3)$ always true?

Kleene Closure If L is any language, L^* is the *Kleene closure* of L defined to be the set of all strings which are formed by concatenation of any number of copies of members of L in any order. We can also write $L^* = L^0 + L^1 + L^2 + L^3 + \cdots$ Fore example, if $L + \{a, ab\}$ then L^* is infinite, but the first few members are $\{\lambda, a, aa, ab, aaa, aab, aba, aaab, aaba, abaa, abab, \ldots\}$

Complementation

If Σ is the alphabet of L, we define the *complement* of L to be $\{w \in \Sigma^* : w \notin L\}$.

Questions

1. What is $\emptyset^*$, the Kleene closure of the empty language?
2. What is L^{**}?
3. Is the union of two regular languages always regular?
4. Is the intersection of two regular languages always regular?
5. Is the complement of a regular language always regular?
6. Is the Kleene closure of a regular language always regular?

Regular Expressions. Given any alphabet Σ, the set of regular expressions over Σ is a context-free language, but not a regular language. In fact, it is an algebraic language, where the operators are union, concatenation, and Kleene closure. If $\Sigma = \{a, b\}$, the alphabet of the language of regular exprssions over Σ is $\{a, b, \emptyset, \lambda, +, ^*, (,)\}$. The concatenation operator is indicated by concatenation. Each regular expression describes a regular language. Examples:

1. $\emptyset$ means the empty language.
2. a means the language $\{a\}$
3. λ means the language $\{\lambda\}$
4. $a + b$ means the language $\{a, b\}$.
5. $a(a + b)$ means the language $\{aa, ab\}$.
6. a^* means the language $\{\lambda, a, aa, aaa, \ldots\}$.
7. What is $(a + \lambda)b$?
8. Is it true that $(a + b)^* = a^* + b^*$?

Another Language for Regular Expressions

```
 PYTHON has built-functions for analyzing regular expressions, but they
 are written in different language.

 Example in Python:

import re
regex = re.compile('^([0-9][0-9]:[0-9][0-9]).* \(([a-zA-Z0-9_]*)\)
 .*\[sessid: (.*?)\].* (GET|POST) (.*)$')
```

For example: `[0-9][0-9]:[0-9][0-9]` is a time.