

Warm-Up Problem

- What are the bitwise operations involved in encoding a word with this shape:
 - (6 bits) x
 - (5 bits) y
 - (5 bits) z
 - (16 bits) i
- Where x , y , z are unsigned, and i is signed.

CS241 Lecture 6


Deterministic Finite Automata

With thanks to Brad Lushman, Troy Vasiga, Kevin Lanctot,
and Carmen Bruni

Formal Languages

We begin with a few definitions

Intentionally vague



Definition

An **alphabet** is a non-empty, finite set of symbols, often denoted by Σ (capital sigma).

Definition

A **string** (or **word**) w is a finite sequence of symbols chosen from Σ . The set of all strings over an alphabet Σ is denoted by Σ^* .

Definition

A **language** is a set of strings.

Definition

The **length of a string** w is denoted by $|w|$.

Stacking

- Since an alphabet is a set of “symbols” (which is vague), and a language is a set of words, a language can be the alphabet of another language
- For instance, the English lexicon (dictionary) is a language in which the alphabet is letters, but English’s alphabet is words from that lexicon
 - (Or at least, that’s one valid way of looking at it!)

Examples

Alphabets:

- $\Sigma = \{a, b, c, \dots, z\}$, the Latin (English) alphabet.
- $\Sigma = \{0, 1\}$, the alphabet of binary digits.
- $\Sigma = \{0, 1, 2, \dots, 9\}$, the alphabet of base 10 digits.

Strings:

- ϵ (epsilon) is the empty string. It is in Σ^* for any Σ . $|\epsilon| = 0$
- For $\Sigma = \{0, 1\}$, strings include $w = 011101$ or $x = 1111$.
Note $|w| = 6$ and $|x| = 4$.

Note: For our course, assume Σ will never contain the symbol ϵ . ϵ is just a notational convention; the actual string is empty.

Examples

Languages:

- $L = \emptyset$ or $\{\}$, the empty language
- $L = \{\epsilon\}$, the language consisting of (only) the empty string
- $L = \{ab^n a : n \in \mathbb{N}\}$, the set of strings over the alphabet $\Sigma = \{a, b\}$ consisting of an a followed by 0 or more b characters followed by an a

Objective

Given a language, determine if a string belongs to it.

How hard is this question? This depends on the language!

- $L = \text{any dictionary}$: Trivial
- $L = \{ab^na : n \in \mathbb{N}\}$: Very easy
- $L = \{\text{valid MIPS assembly programs}\}$: Easy
- $L = \{\text{valid Java/C/C++ Programs}\}$: Harder
- $L = \{\text{set of programs that halt}\}$: Impossible!

Which languages a priori are easier than others?

Membership in Languages

In order of relative difficulty:

- finite
- regular
- context-free
- context-sensitive
- recursive
- impossible languages

Finite Languages

Why are these easy to determine membership?

- To determine membership in a language, just check for equality with all words in the language!
- Even if the language is of size $10^{10^{10}}$, this is still $O(1)$, albeit with a very high k .
- Is there a more efficient way?

A Leading Example:

Suppose we have the language

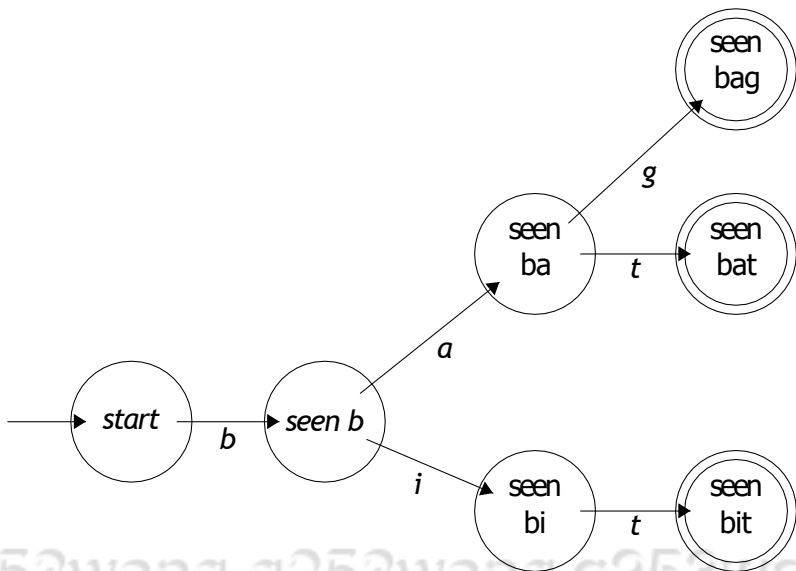
- $L = \{\text{bat, bag, bit}\}$

Write a program that determines whether $w \in L$ given that each character of w is scanned exactly once without the ability to store previously seen characters.

Algorithm 1 Algorithm to recognize L

```
1: if first char is a then
2:   if next char is a then
3:     if next char is g then
4:       if no next char then
5:         Accept
6:       else
7:         Reject
8:       end if
9:     else if next char is t then
10:      if no next char then
11:        Accept
12:      else
13:        Reject
14:      end if
15:    else
16:      Reject
17:    end if
18:  else if next char is i then
19:    if next char is t then
20:      if no next char then
21:        Accept
22:      else
23:        Reject
24:      end if
25:    else
26:      Reject
27:    end if
28:  else
29:    Reject
30:  end if
31: else
32:   Reject
33: end if
```

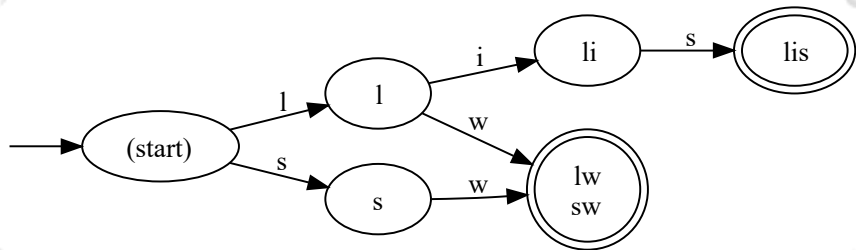
Pictorially



Extremely Important Features of Diagram

- An arrow into the initial start state.
- Accepting states are two circles.
- Arrows from state to state are labelled.
- Error state(s) are implicit (a common "hack").

Second Example



Beyond the finite

Despite the simplicity of the finite examples, these diagrams can easily generalize to recognize a larger class of languages known as *regular languages*.

Definition

A **regular language** over an alphabet Σ consists of one of the following:

1. The empty language and the language consisting of the empty word are regular
2. All languages $\{a\}$ for all $a \in \Sigma$ are regular.
3. The union, concatenation or Kleene star (pronounced klay-nee) of any two regular languages are regular.
4. Nothing else.

Union, Concatenation, Kleene Star

Let L , L_1 and L_2 be three regular languages. Then the following are regular languages

- Union: $L_1 \cup L_2 = \{x : x \in L_1 \text{ or } x \in L_2\}$
- Concatenation: $L_1 \cdot L_2 = L_1 L_2 = \{xy : x \in L_1, y \in L_2\}$
- Kleene star: $L^* = \{\epsilon\} \cup \{xy : x \in L^*, y \in L\}$

$$= \bigcup_{n=0}^{\infty} L^n$$

Examples

Suppose that $L_1 = \{\text{up, down}\}$, $L_2 = \{\text{hill, load}\}$ and $L = \{a, b\}$ over appropriate alphabets. Then

- $L_1 \cup L_2 = \{\text{up, down, hill, load}\}$
- $L_1 L_2 = \{\text{uphill, upload, downhill, download}\}$
- $L^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, \dots\}$

Sample Question

Let $\Sigma = \{a, b\}$. Explain why the language $L = \{ab^n a : n \in \mathbb{N}\}$ is regular.

Solution: $\{a\}$ and $\{b\}$ are finite, and so regular. $\{b\}^*$ is also regular, regular languages are closed under Kleene star. Then, the concatenation $\{a\} \cdot \{b\}^* \cdot \{a\}$ must also be regular.

Regular Expressions

In tools like `egrep`, regular *expressions* are often used to help find patterns of text. Regular expressions are just a way of *expressing* regular languages.

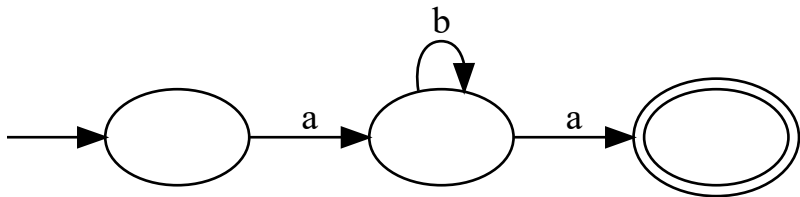
The notation is very similar, except we drop the set notation. As examples:

- $\{\epsilon\}$ becomes ϵ (and similarly for other singletons).
- $L_1 \cup L_2$ becomes $L_1 \mid L_2$
- Concatenation is never written with the explicit \cdot
- Order of operations: $*$, \cdot then \mid . (Kleene star, concatenation, then union). The previous example as a regular expression would be ab^*a .

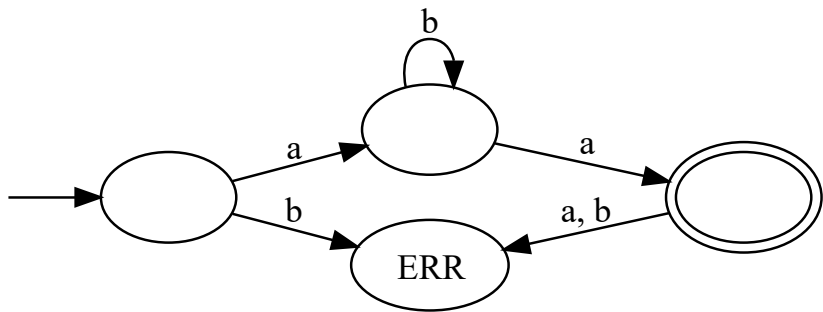
Extending the Finite Languages Diagram

Can we use our pictorial representation to represent regular languages?

Yes, if we allow our picture to have loops!



Version with Error State (for CS360)



Error States in CS 241

- If a bubble does not have a valid arrow leaving it, we assume this will transition to an error state.
- In CS 360 and CS 365, you will be required to show explicitly the error state (you can choose to do so in this class as well if you want).
- Error states are part of the formal system, but are usually left out in informal discussion.

Deterministic Finite Automata

These machines are called Deterministic Finite Automata

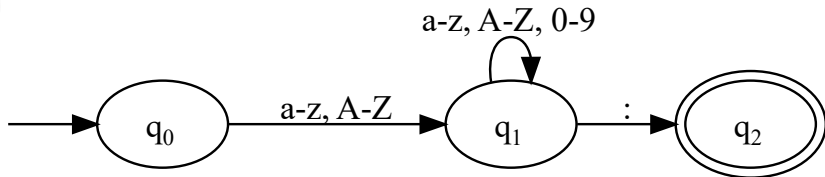
Definition

A **DFA** is a 5-tuple $(\Sigma, Q, q_0, A, \delta)$:

- Σ is a finite non-empty set (alphabet).
- Q is a finite non-empty set of states.
- $q_0 \in Q$ is a start state
- $A \subseteq Q$ is a set of accepting states
- $\delta : (Q \times \Sigma) \rightarrow Q$ is our [total] transition function (given a state and a symbol of our alphabet, what state should we go to?).

Example

MIPS labels for our DFA (described below):



- $\Sigma = \{\text{ASCII characters}\}$
- $Q = \{q_0, q_1, q_2\}$
- q_0 is our start state
- $A = \{q_2\}$ (note: this is a set!)
- δ is defined by
 - $\delta(q_0, \text{letter}) = q_1$
 - $\delta(q_1, \text{letter or number}) = q_1$
 - $\delta(q_1, :) = q_2$
 - All other transitions go to an error state.

Rules for DFAs

- States can have labels inside the bubble. This is how we refer to the states in Q.
- For each character you see, follow the transition. If there is none, go to the (implicit) error state.
- *Once the input is exhausted*, check if the final state is accepting. If so, accept. Otherwise reject.

Samples In Class

Write a DFA over $\Sigma = \{a, b\}$ that...

- Accepts only words with an even number of *as*
- Accepts only words with an odd number of *as* and an even number of *bs*
- Accepts only words where the parity of the number of *as* is equal to the parity of the number of *bs*