Warm Up Problem

- How did we solve the issue of our code needing to print out one byte at a time?

CS 241 Lecture 6

Deterministic Finite Automata

With thanks to Brad Lushman, Troy Vasiga and Kevin Lanctot

Reminder Formal Languages Definitions

We begin with a few definitions

Definition

An **alphabet** is a non-empty finite set of symbols often denoted by Σ.

Definition

An **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 $|w|$.

Membership in Languages

In order of relative difficulty, to recognize that an element is a member of a language is easier for:

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

Finite Languages

Why are these easy to determine membership?

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 theoretically possible.
- However, is there a more efficient way?

A Leading Example:

Suppose we have the language

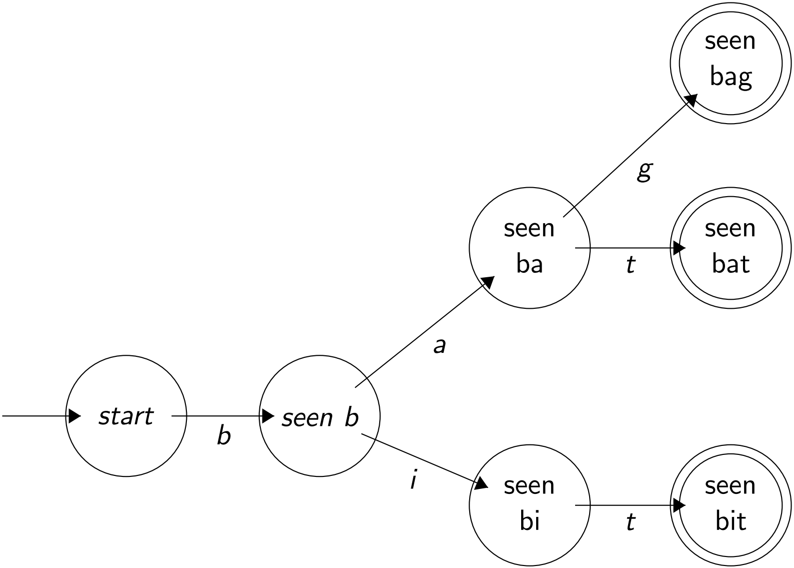
$$L = \{\text{bat}, \text{bag}, \text{bit}\}$$

Write a program that determines whether or not $w \in L$ given that each character of w is scanned exactly once without the ability to store previously seen characters.

Algorithm 1 Algorithm to recognize L

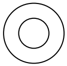
```
if first char is a b then
    if next char is a then
        if next char is g then
            if no next char then
                Accept
            else
                Reject
            end if
        else if next char is t then
            if no next char then
                Accept
            else
                Reject
            end if
        else
            Reject
        end if
    else if next char is i then
        if next char is t then
            if no next char then
                Accept
            else
                Reject
            end if
        else
            Reject
        end if
    else
        Reject
    end if
else
    Reject
end if
```

Pictorially

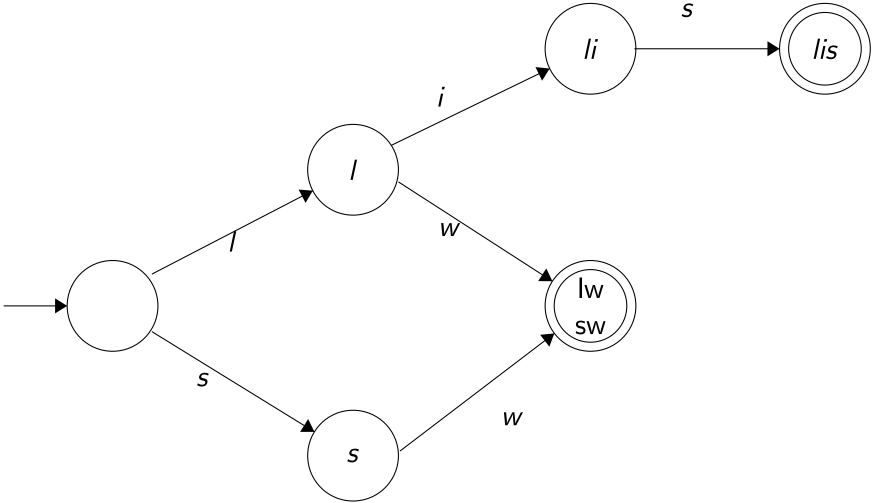


Extremely Important Features of Diagram

- An arrow into the initial start state.
- Accepting states are two circles.
- Arrows from state to state are labelled.
- Error state(s) are implicit (CS 241 Special).



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. (See next page)
4. Nothing else.

Union, Concatenation, Kleene Star

Let L, L_1 and L_2 be two 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 $L^* = \{\epsilon\} \cup \{xy : x \in L^*, y \in L\} = \bigcup_{n=0}^{\infty} L^n$ where

$$L^n = \begin{cases} \{\epsilon\} & \text{if } n = 0 \\ LL^{n-1} & \text{otherwise} \end{cases}$$

Equivalently, L^* is the set of all strings consisting of 0 or more occurrences of strings from L concatenated together.

Examples

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

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

Sample Question

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

Sample Question

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

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

Regular Expressions

- In tools like *grep*, regular expressions are often used to help find patterns of text.
- 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 $L_1 \mid L_2$ or $L_1 + L_2$ for alternation
  - Concatenation is still $\cdot$
  - The empty language maintains the same notation of $\emptyset$.

  Order of operations: $*$, $\cdot$ then $\mid$ (or $+$). (Kleene star, concatenation then alternation). The previous example as a regular expression would be ab^*a.

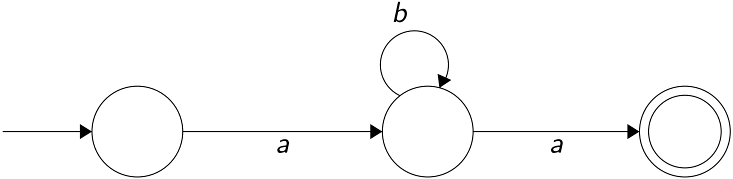
Extending the Finite Languages Diagram

Can we use out pictorial representation to represent regular languages?

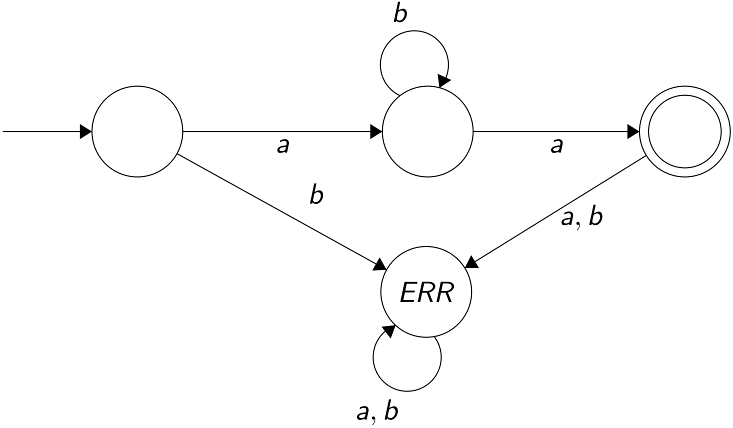
Extending the Finite Languages Diagram

Can we use out pictorial representation to represent regular languages?

Yes! As long as we allow our picture to have loops!



Picture With Error State (for CS 360)



Error state 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).

Deterministic Finite Automata

These machines are called Deterministic Finite Automata.

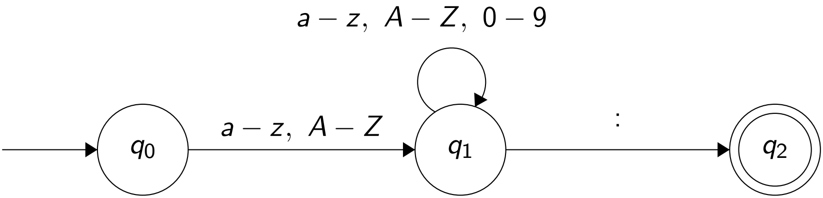
Definition

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):



- $\Sigma = \{$ 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 would be how we refer to the states in Q.
- For each character you see, follow the transition. If there is none, go to the 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