Formal Languages

Definitions

- **Alphabet** - non-empty finite set of symbols, denoted Σ.
- **String** (or **word**) - finite sequence of symbols from Σ.
- **Language** - set of strings.

String

- Denote the length of a string w by $|w|$.
- Empty string is denoted by ϵ, $|\epsilon| = 0$, $|\epsilon\epsilon\epsilon| = 0$.

  Note: ϵ is not a symbol in Σ.

Be mindful of notation - understand the difference between a symbol, a string and a language (set).

For example:

ϵ is the empty string.

Given $\Sigma = \{a\}$, is $a = \epsilon a \epsilon$?

Be mindful of notation - understand the difference between a symbol, a string and a language (set).

For example:

ϵ is the empty string.

Given $\Sigma = \{a\}$, is $a = \epsilon a \epsilon$? **Yes**

$\{\}$ or $\emptyset$ is the empty language.

What about $\{\epsilon\}$?

Be mindful of notation - understand the difference between a symbol, a string and a language (set).

For example:

ϵ is the empty string.

Given $\Sigma = \{a\}$, is $a = \epsilon a \epsilon$? **Yes**

$\{\}$ or $\emptyset$ is the empty language.

What about $\{\epsilon\}$?

A singleton set. Language with 1 string - only contains ϵ.

The alphabet will depend on the domain you are working in.

- $\Sigma = \{a, b, c, \ldots, z\}$, $L = \{$English words$\}$
- $\Sigma = \{$ASCII characters$\}$, $L = \{$Valid Assembly programs$\}$
- $\Sigma = \{$ASCII characters$\}$, $L = \{$Valid C programs$\}$
- $\Sigma = \{0, 1\}$, $L = \{$Valid ML programs$\}$
- $\Sigma = \{a\}$, $L = \{$Set of strings where # of a's is divisible by 3$\}$
- $\Sigma = \{a, b, c\}$, $L = \{$Palindromes over $a, b, c\}$

Problem: Given a string, how to recognize if it belongs to the specified language.

Recognition

The difficulty of determining if a string belongs to a specified language depends on the complexity of the language.

- $\{$English words$\}$ - Easy
- $\{$Valid Assembly programs$\}$ - Still Easy
- $\{$Valid C programs$\}$ - Harder
- $\{$Valid C programs that always halt$\}$ - Impossible

Classes of Languages

The difficulty of recognition defines a hierarchy of classes of languages.

- Finite (easy)
- Regular
- Context-free (harder)
- Context-sensitive
- Recursive (hard)
- Recursively Enumerable
- etc (impossible)

Methods of Recognition

How can we determine if a given string belongs to a language?

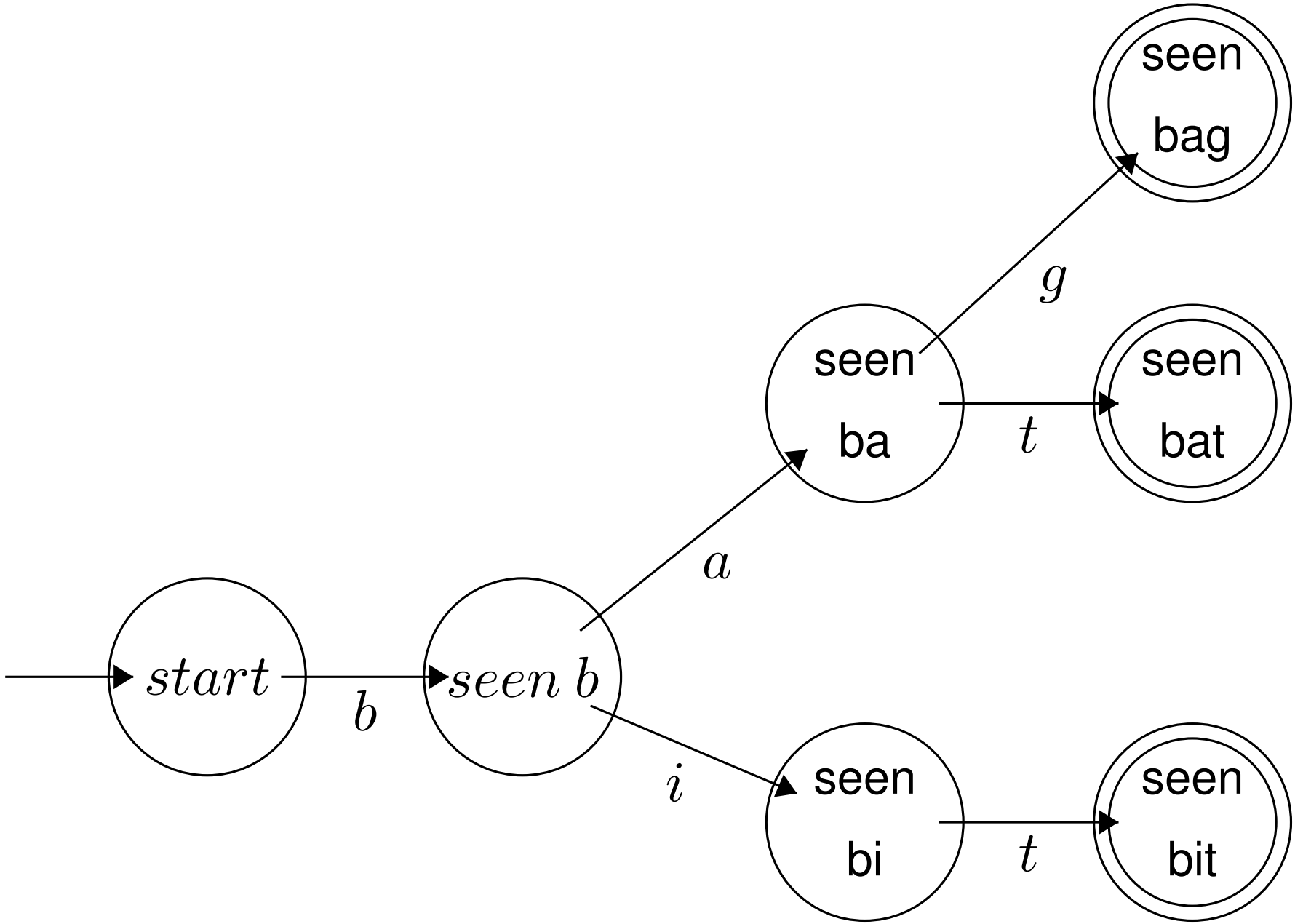
- Write a program (based on the language) to check.
- Other - finite state machines, Turing machines, etc

For example: Given language $L = \{\text{bat, bag, bit}\}$.

We could write a program to check if a given input string matches one of the words within the language.

```
if 1st char is 'b' then
   if next char is 'a' then
      if next char is 't' then
         if no more input then
            Accept
         else
            Reject
      else if next char is 'g' then
         if no more input then
            Accept
         else
            Reject
   if next char is 'i' then
...
```

Visualization



Extremely Important Features of Diagram

- Circles represent *states* - these represent the state of the program; i.e. the progress we have made in recognizing a valid pattern. recognized so far.
- An arrow into the start state denotes where to begin.
- Transition arrows from state to state have a single symbol.
- States that *accept* are double circled.

Note: we could extend this diagram for any finite language.

Remember: Our problem is to recognize a valid programs - most programming languges have an infinite number of valid programs.

Regular Languages -Building Blocks

- Finite 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.

Regular Expressions

Regular Expression	Set Notation	Description
$\emptyset$	$\{\}$	Empty Language
ϵ	$\{\epsilon\}$	Language containing only ϵ
a	$\{a\}$	Singleton language of 1 string
$E_1 \mid E_2$	$L_1 \cup L_2$	Alternation
$E_1 \cdot E_2$	$L_1 L_2$	Concatenation
E^*	L^*	Repetition

Precedence:

* before $\cdot$, e.g. $aa^* \equiv a(a^*)$

$\cdot$ before $\mid$, e.g. $aa \mid b \equiv (aa) \mid b$

Deterministic Finite Automata (DFA)

Formally a DFA M is a 5-tupple $M = (\Sigma, Q, q_0, A, \delta)$

- Σ - non-empty, finite alphabet
- Q - non-empty, finite set of states
- q_0 - start state
- $A \subseteq Q$ - set of accepting states
- $\delta : (Q \times \Sigma) \rightarrow Q$ transition function

  From a given state, on a given alphabet symbol transition to next state - consumes a single symbol.

Extended Transition Function

Can extend δ to consume a word (instead of a single symbol) by the following recursive definition:

Base Case: $\delta^*(q, \epsilon) = q$

Recursive Case: $\delta^*(q, cw) = \delta^*(\delta(q, c), w)$ where $c \in \Sigma, w \in \Sigma^*$

Acceptance: DFA M accepts a word w if $\delta^*(q_0, w) \in A$; i.e. accept if starting at q_0 and following δ for each symbol of w, in turn, ends in a state in A.

The language (set) of M is all words accepted by M, denoted $L(M)$.