Deterministic Finite Automata

Based on slides of Aaron Deever

Overview

- Goal: universal model of computation
  - What can and can not be computed?

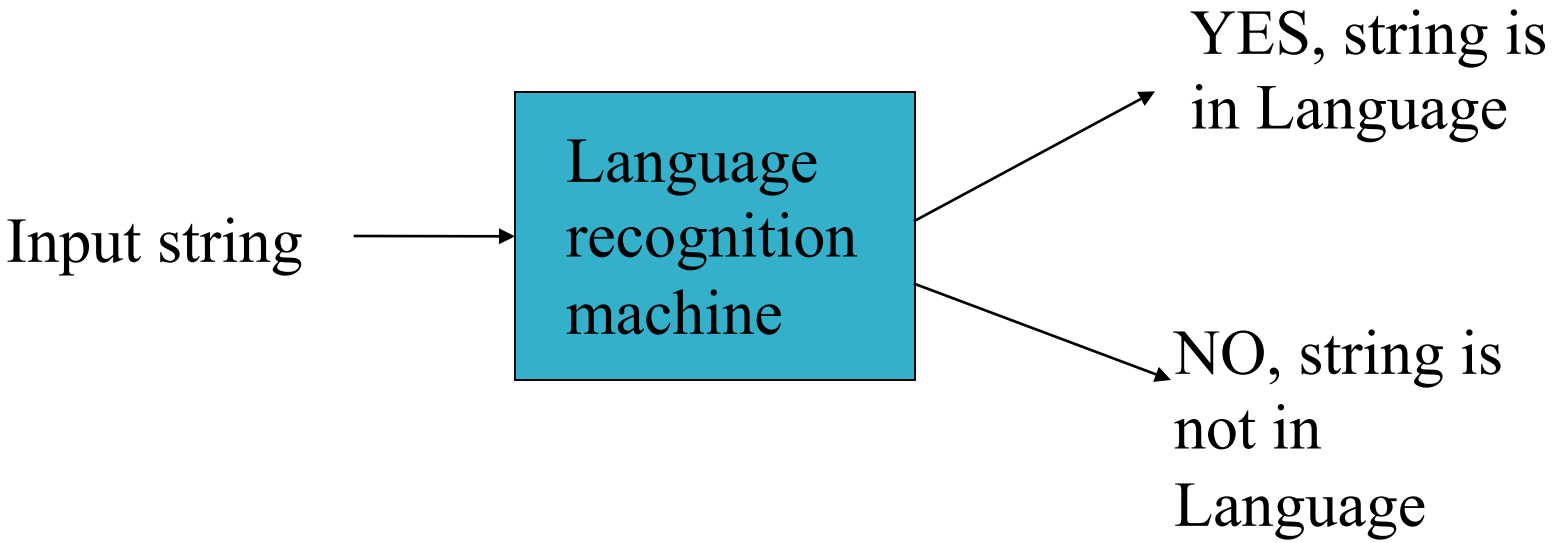
- To start: restricted model of computation
  - Finite automata
    - Model for a computer with limited memory
    - Fixed, finite number of states it can be in
    - Can only retain its current state

Example

- Suppose we want to design a machine that can keep track of whether we get more "heads" than "tails" when we flip a coin
  - Input is a string over the alphabet $\Sigma = \{0,1\}$
- Can we build a finite automaton that can keep track of this?
- What if we are only interested in whether the total number of heads is an even number – can a finite automaton keep track of this?

Language Recognition Machine

- Given a string, and a definition of a language (set of strings), is the string a member of the language?

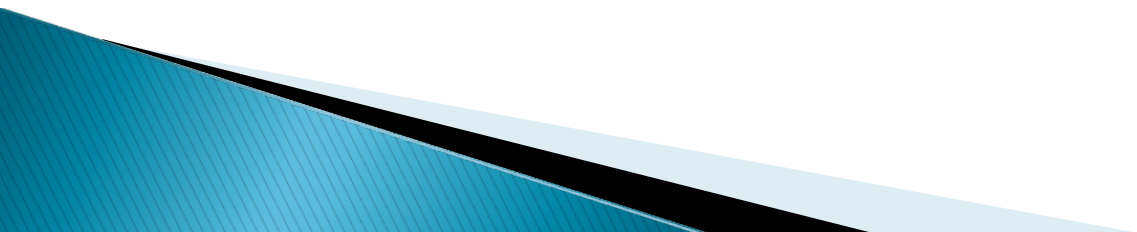


Language Classes

- Over the course of the semester, we will be looking at classes of languages:
  - Each class will have its own means for describing a language
  - Each class will have its own machine model for string recognition
  - Languages and machines get more complex as we move forward in the course.

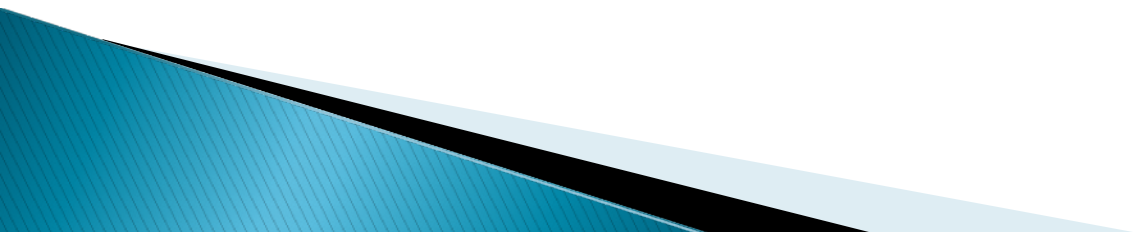
Languages

- A language is a set of strings.
- A class of languages is nothing more than a set of languages



Regular Languages

- Today we start looking at our first class of languages: Regular languages
  - Machine for accepting: Finite Automata
  - Means of defining: Regular Expressions

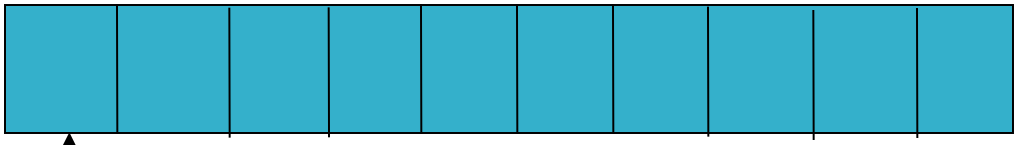


Deterministic Finite Automata

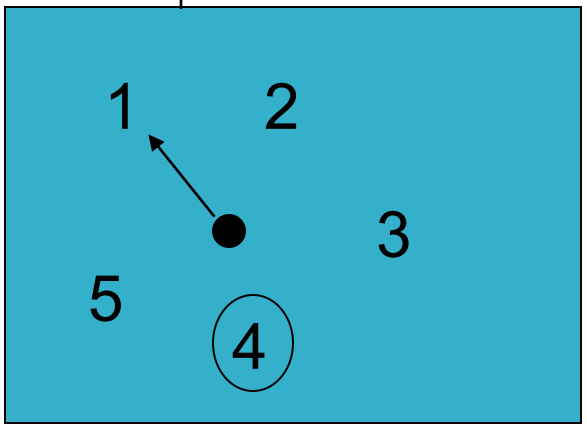
- A deterministic finite automaton (DFA) consists of:
  - A read tape (with symbols on it)
  - A machine with a fixed, finite number of states.
  - At any point, the machine is in one of these states.
    - Start state – The state the machine is in at the beginning of execution
    - Accepting states – The state(s) the machine has to be in after execution in order for a string to be "accepted"
      - There may be 0 or more of these
    - Non-accepting states
      - There may be 0 or more of these

Deterministic Finite Automata

This holds the input string



Input tape



State machine

In this example (we'll define formal notation later):

- Current state (arrow)
- Starting state (1)
- Accepting states (circles)

Deterministic Finite Automata

- How the automaton works
  - Reads a character (symbol from alphabet) on the tape
  - Based on the character read and the current state of the machine, puts machine into another state
  - Moves the read head to the right
  - Repeats the above until all characters have been read.

Deterministic Finite Automata

- Testing a string for membership
  - Place the string to be tested on the read tape
  - Place the machine in the start state
  - Let the machine run to completion
  - If, upon completion, the machine is in an accepting state, the string is accepted, otherwise it is not.

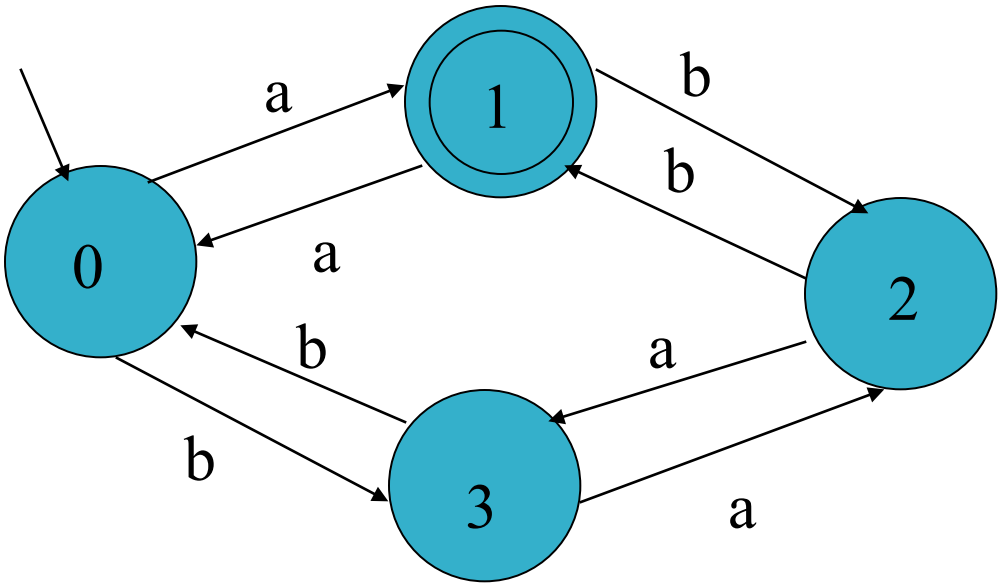
Deterministic Finite Automata

- Transition function
  - Defines what state the machine will move into given:
    - The current machine state
    - The character read off the tape
- A finite automaton is sometimes illustrated as a directed graph where nodes represent states and labelled edges represent transitions.
  - State diagram

Deterministic Finite Automata

- Example:
  - L = {w ∈ {a,b}* | w has an odd number of a's and an even number of b's}

	# a's	# b's
0	even	even
1	odd	even
2	odd	odd
3	even	odd



Deterministic Finite Automata

- The DFA can also be specified by a table

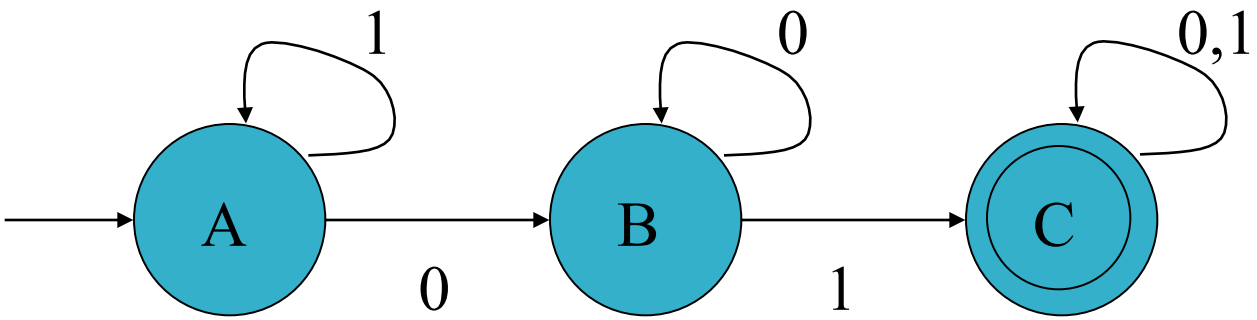
SYMBOL

STATE

	a	b
→ 0	1	3
*1	0	2
2	3	1
3	2	0

Deterministic Finite Automata

- Another example
  - L = {w ∈ {0,1}* | w contains 01 as a substring}



Deterministic Finite Automata

- More Examples
  - $L = \{w \in \{0,1\}^* \mid w$ ***does not*** contain 01 as a substring$\}$
  - $L = \{w \in \{0,1\}^* \mid w$ begins with a 1 and ends with a 0$\}$
  - $L = \{w \in \{0,1\}^* \mid w$ contains at least 3 1's$\}$
  - $L = \{w \in \{0,1\}^* \mid$ every odd position of w is a 1$\}$
    - $w = w_1 w_2 \ldots w_k$
    - $\epsilon \in L$

See Exercise 1.6 in Sipser for more.