Lecture 1
Computation and Languages

CS311

Fall 2012

Computation

- Computation uses a well defined series of actions to compute a new result from some input.
- We perform computation all the time

```
  348
+ 213
-------
```

```
    1
  348
+ 213
-------
    1
```

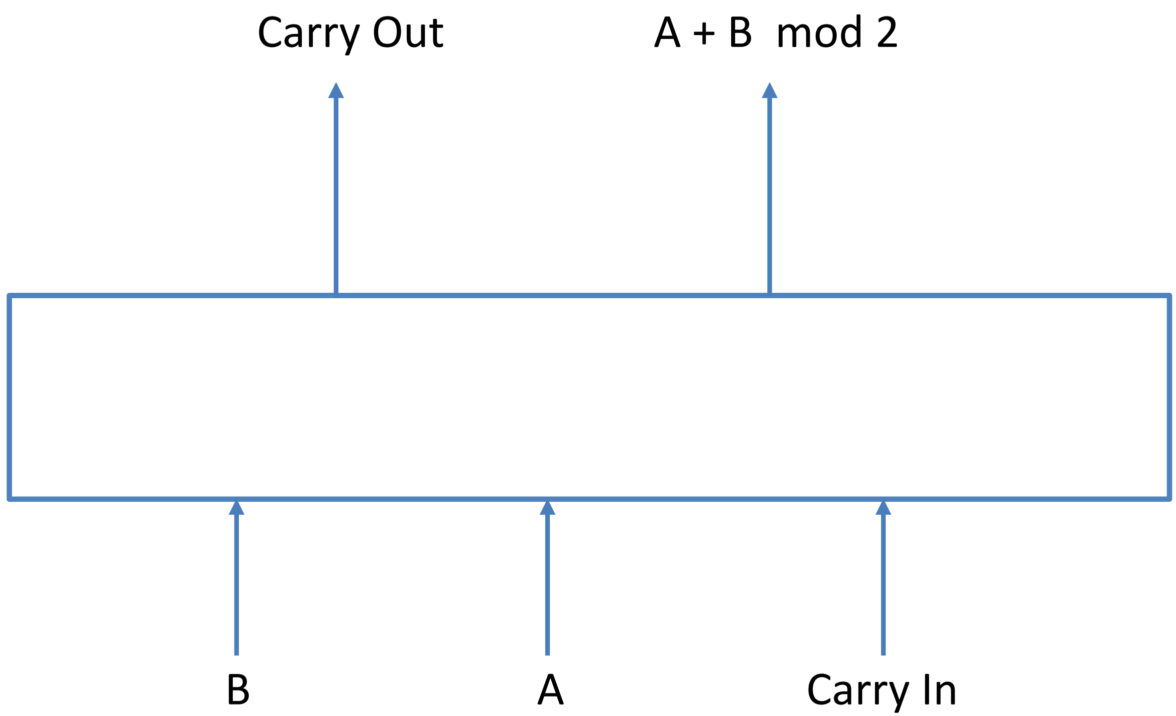
```
  0 1
  348
+ 213
-------
   61
```

```
  0 1
  348
+ 213
-------
  561
```

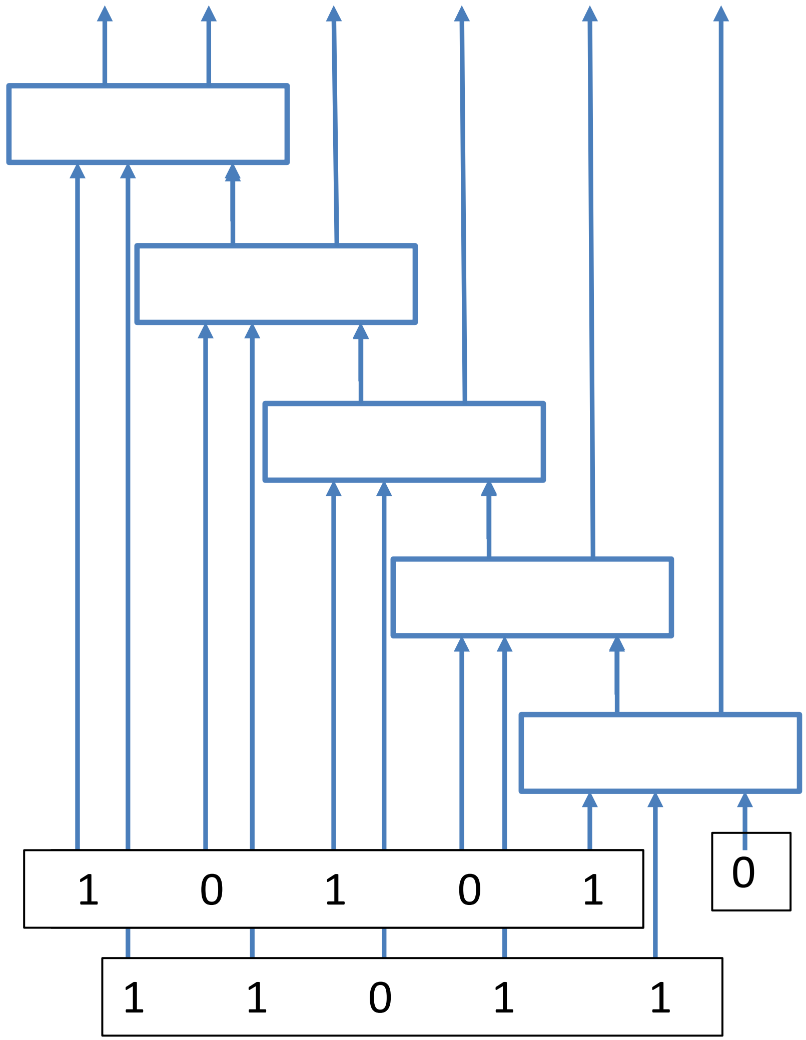
Properties

- As computer scientists we know Computation
  - Can be carried out by machines
  - Can be broken into sub-pieces
  - Can be paused
  - Can be resumed
  - Can be expressed using many equivalent systems
- The study of computation includes computability
  - what can be computed by different kinds of systems

Binary adders



Ripple Carry Adder



Languages and Computation

- There are many ways to compute the sum of two binary numbers.
- One historically interesting way is to use the notion of a language as a view of computation.

Language = A set of strings

- A *language* over an alphabet Σ is any subset of Σ^*. That is, any set of strings over Σ.

- A language can be finite or infinite.

- Some languages over {0,1}:
  - {Λ,01,0011,000111, … }

  - The set of all binary representations of prime numbers: {10,11,101,111,1011, … }

- Some languages over ASCII:
  - The set of all English words
  - The set of all C programs

Language Representation

- Languages can be described in many ways
  - For a finite language we can write down all elements in the set of strings {“1”, “5” , “8”}
  - We can describe a property that is true of all the elements in the set of strings $\{ x \mid |x|=1 \}$
  - Design a machine that answers yes or no for every possible string.
  - We can write a generator that enumerates all the strings (it might run forever)

A language for even numbers written in base 3

Base 10
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

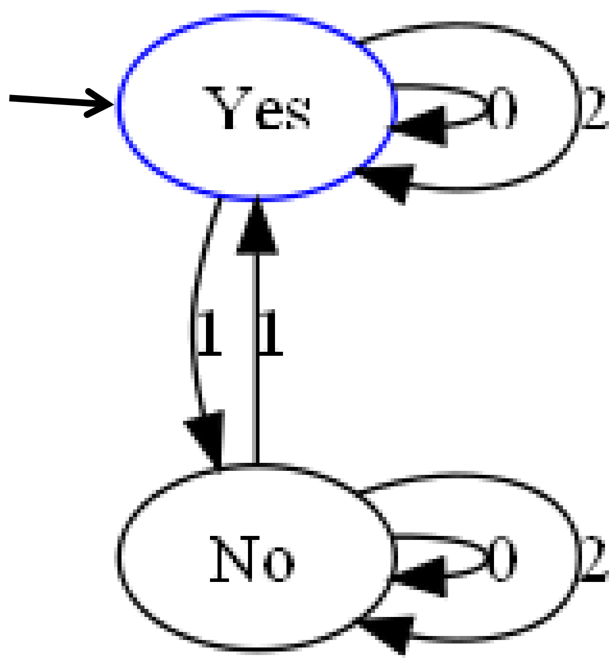
Base 3
- 0
- 1
- 2
- 10
- 11
- 12
- 20
- 21
- 22

The language

{ 0, 2, 11, 20, 22, …}

There is an infinite number of them, we can write them all down. We'll need to use another mechanism

A machine that answers yes or no for every even number written in base 3.



{ 0, 2, 11, 20, 22, …}

DFA Formal Definition

- **A D**FA is a quintuple **A**=(**Q**, Σ, **s**, **F**, δ), where
  - **Q** is a set of *states*
  - Σ is the alphabet of *input symbols (A in Hein)*
  - **s** is an element of **Q** --- the *initial state*
  - **F** is a subset of **Q** ---the set of *final states*
  - δ: **Q** $\times$ $\Sigma \longrightarrow$ **Q** is the *transition function*

Example

- Q = {Yes,No}
- Σ = {0,1,2}
- S = Yes (the initial state)
- F = {Yes} (final states are labeled in blue)
- δ: $\mathbf{Q} \times \Sigma \longrightarrow \mathbf{Q}$

  delta Yes 0 = Yes

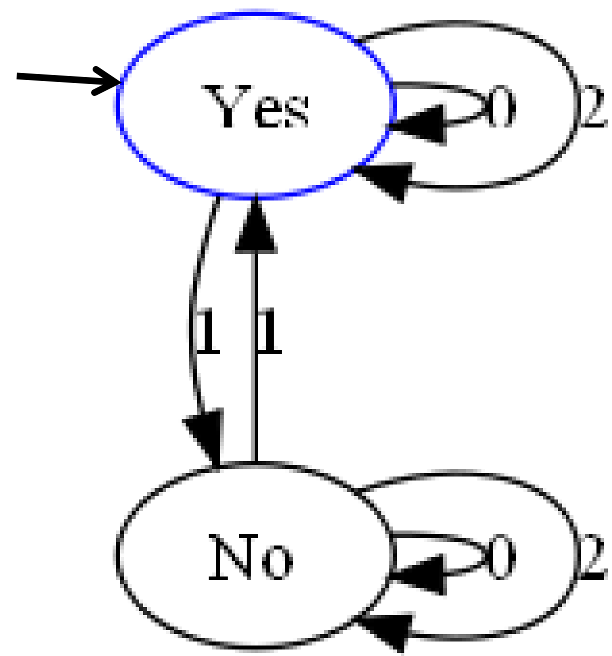
  delta Yes 2 = Yes

  delta Yes 1 = No

  delta No 0 = No

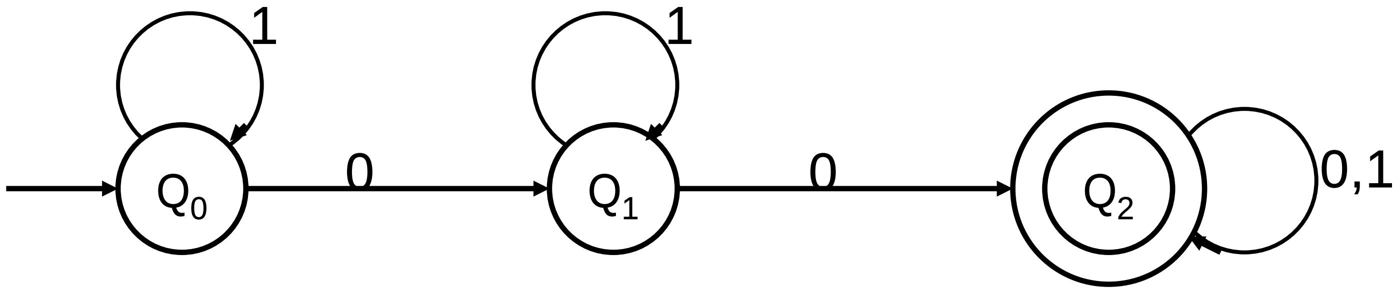
  delta No 1 = Yes

  delta No 2 = No



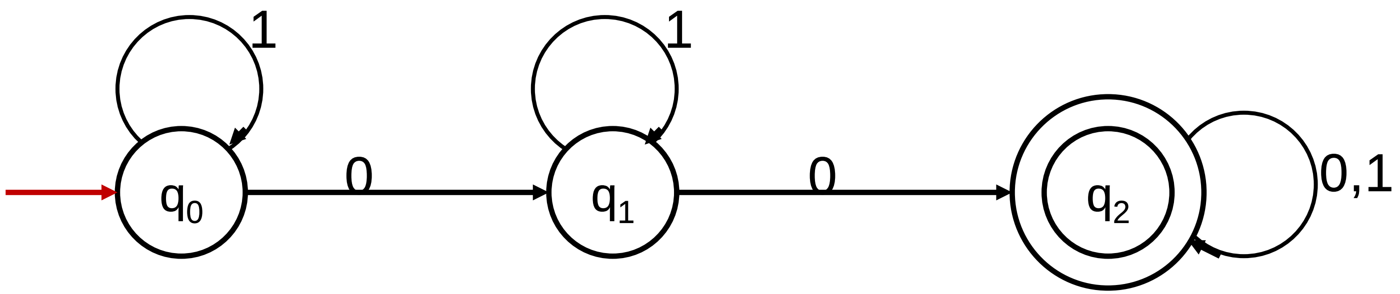
Properties

- DFAs are easy to present pictorially:



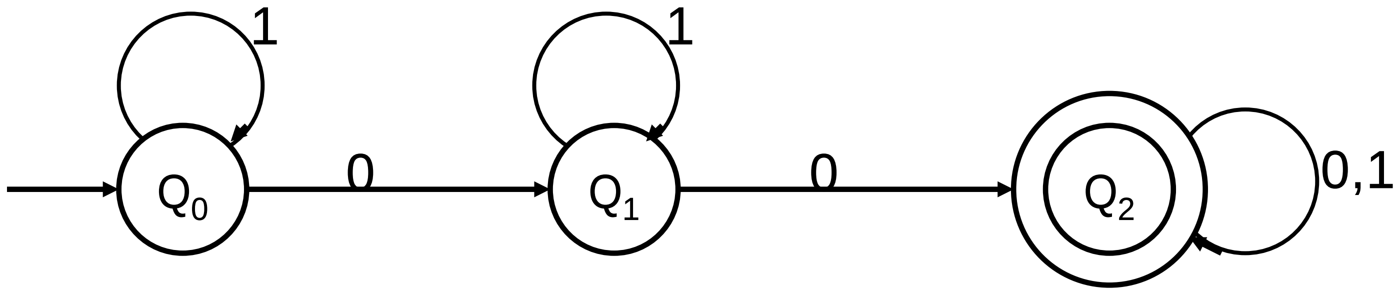
They are directed graphs whose nodes are *states* and whose arcs are labeled by one or more symbols from some alphabet Σ.

Here Σ is {0,1}.

- One state is *initial* (denoted by a short incoming arrow), and several are *final/accepting* (denoted by a double circle in the text, but by being labeled blue in some of my notes). For every symbol $a \in \Sigma$ there is an arc labeled *a* emanating from every state.
- 

- Automata are string processing devices. The arc from q_1 to q_2 labeled 0 shows that when the automaton is in the state q_1 and receives the input symbol 0, its next state will be q_2.

- Every path in the graph spells out a string over *S*. Moreover, for every string $w \in \Sigma^*$ there is a unique path in the graph labelled *w*. (Every string can be processed.) The set of all strings whose corresponding paths end in a final state is the *language of the automaton*.



- In this example, the language of the automaton consists of strings over {*0,1*} containing at least two occurrences of *0*. In the base 3 example, the language is the even base three numbers

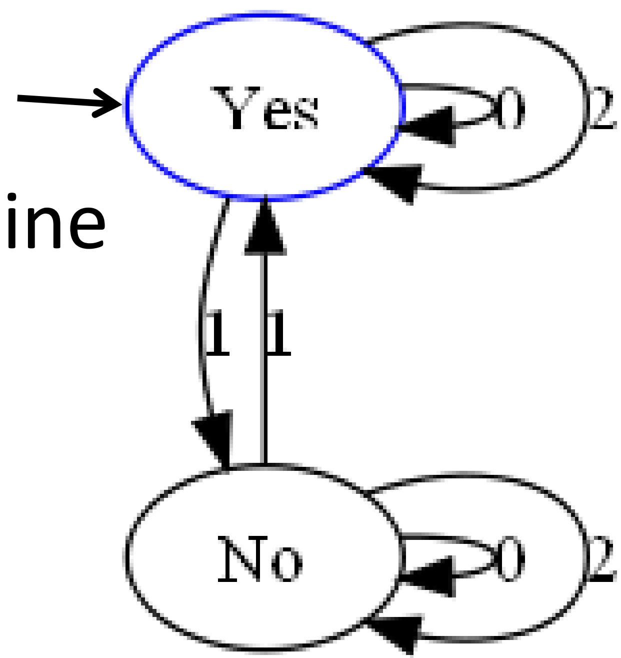
What can DFA's compute

- DFAs can express a wide variety of computations
  1. Parity properties (even, odd, mod n) for languages expressed in base m
  2. Addition (we'll see this in a few slides)
  3. Many pattern matching problems (grep)
- But, not everything.
  - E.g. Can't compute $\{ x \mid x \text{ is a palindrome} \}$

Are they good for things other than computation?

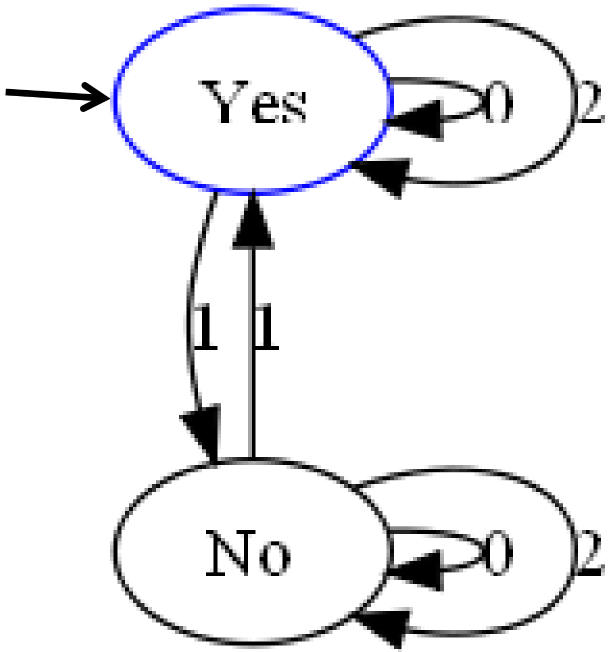
- We can use DFAs to compute if a string is a member of some languages.
- But a DFA is mathematical structure ($\mathbf{A} = (\mathbf{Q}, \Sigma, \mathbf{s}, \mathbf{F}, \delta)$)
- It is itself an object of study
- We can analyze it and determine

some of its properties



Prove

- Q = {Yes,No}
- Σ = {0,1,2}
- S = Yes (the initial state)
- F = {Yes} (final states are labeled in blue)
- δ: **Q** × Σ ⟶ **Q**
  - delta Yes 0 = Yes
  - delta Yes 2 = Yes
  - delta Yes 1 = No
  - delta No 0 = No
  - delta No 1 = Yes
  - delta No 2 = No



parity(Yes) = 0
parity(No) = 1

Let s∈*Q*, *d*∈Σ
Delta(s,d) = parity^{-1} ((3 * (parity s) + d) `mod` 2)

Six cases

1. delta Yes 0 = Yes $\quad parity^{-1}$ ((3 * (parity Yes) + 0) `mod` 2)
2. delta Yes 2 = Yes $\quad parity^{-1}$ ((3 * (parity Yes) + 2) `mod` 2)
3. delta Yes 1 = No $\quad parity^{-1}$ ((3 * (parity Yes) + 1) `mod` 2)
4. delta No 0 = No $\quad parity^{-1}$ ((3 * (parity No) + 0) `mod` 2)
5. delta No 1 = Yes $\quad parity^{-1}$ ((3 * (parity No) + 1) `mod` 2)
6. delta No 2 = No $\quad parity^{-1}$ ((3 * (parity No) + 2) `mod` 2)

parity(Yes) = 0

parity(No) = 1

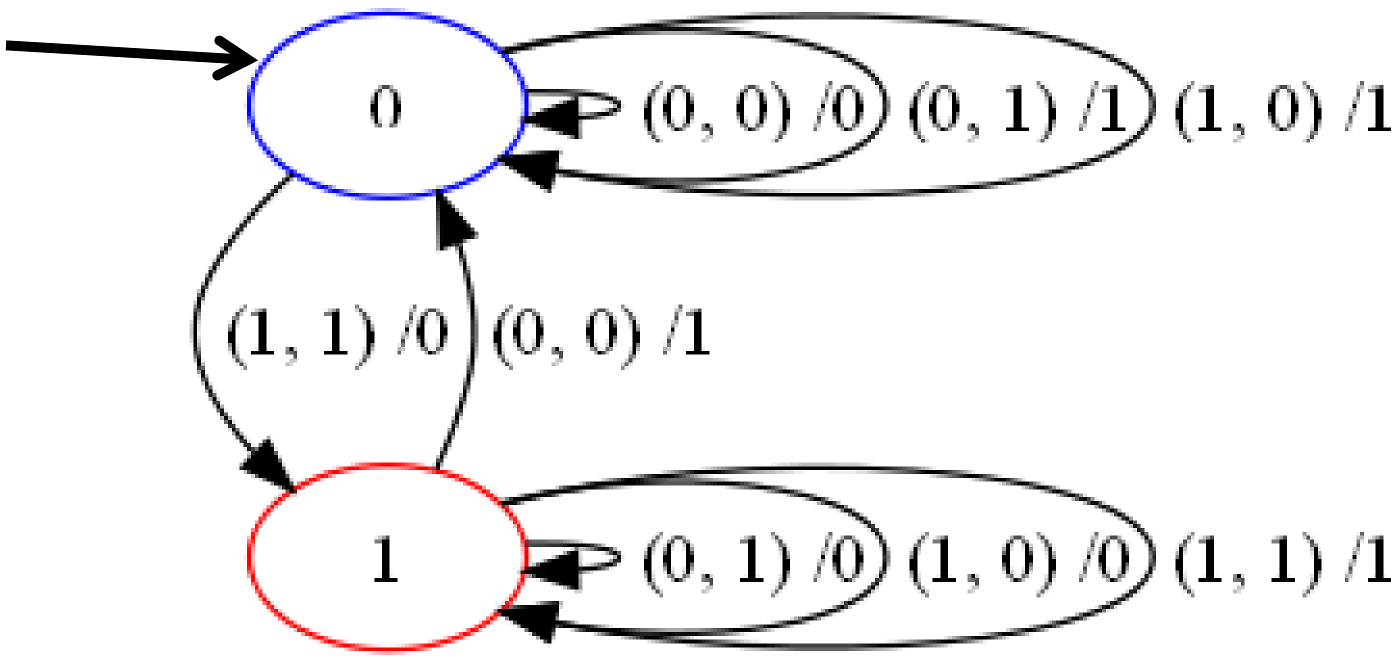
Addition as a language

- Let A,B,C be elements of $\{0,1\}^n$ I.e. binary numbers of some fixed length n
- Consider the language L = { ABC | A+B=C }
- E.g. Let n=4 bits wide
  - 0000 0000 0000 is in L
  - 0010 0001 0011 is in L
  - 1111 0001 0000 is not in L

How can we encode this as a DFA?

- Change of representation
- Let a string of 3 binary numbers, such as "0010 0001 0011" be encoded as a string of 3-tuples such as "(0,0,0) (0,0,0) (1,01) (0,1,1)"
- Why can we do this? Nothing says the alphabet can't be a set of triples!
- Now lets reverse the order of the triples in the string "(0,1,1) (1,01) (0,0,0) (0,0,0)"
  - Least significant bit first.

Encode as follows



Mealy Machine

- **A Mealy** is a 6-tuple **A**=(**Q**, Σ, **O**, **s**, δ, emit), where
  - **Q** is a set of *states*
  - Σ is the alphabet of *input symbols (A in Hein)*
  - **O** is the alphabet of the output
  - **s** is an element of **Q** --- the *initial state*
  - δ: **Q** × Σ ⟶ **Q** is the *transition function*
  - emit : **Q** × Σ ⟶ **O** is the emission *function*

The Big Picture

- Computer Science is about computation
- A computational system describes a certain kind of computation in a precise and formal way (DFA, Mealy machines).
  - What can it compute?
  - How much does it cost?
  - How is it related to other systems?
  - Can more than one system describe exactly the same computations?

History

- The first computational systems were all based on languages.
- This led to a view of computation that was language related.
  - E.g. which strings are in the language.
  - Is one language a subset (or superset) of another.
  - Can we decide?
  - If we can decide, what is the worst case cost?
  - Are there languages for which the membership predicate cannot be computed?

A Tour of this class

- Languages as computation
  - A hierarchy of languages
    - Regular languages
    - Context Free languages
    - Turing machines
  - A Plethora of systems
    - Regular expressions, DFAs, NFAs, context free grammars, push down automata, Mealy machines, Turing machines, Post systems, and more.
- Computability
  - What can be computed
  - Self applicability (Lisp self interpretor)
  - The Halting Problem

Take aways

- A computational system is like a programming language.
  - A program describes a computation.
  - Different languages have different properties.
  - A language can be analyzed
    - A formal computational system is just data (DFA is a 5-tuple)
    - The structure can be used to prove things about the system
      - What properties hold of all programs?
      - What can never happen?
  - A program can be analyzed
    - A program is just data
    - What does this program do?
    - Does it do the same as another?
    - What is its cost?
    - Is it hard understand?

Why is this important?

- Languages are every where
- Many technologies are based upon languages
  - Parsing, grep, transition systems.
- The historical record has a beauty that is worth studying in its own right.
- Reasoning about computation is the basis for modern computing.
  - What do programs do? What can we say about what they don’t do? What do they cost? What systems makes writing certain class of programs easier?
- Computational Systems and Programs are just data.
- Knowing what is possible, and what isn’t.