## A (Mostly) Informal Introduction to Computation Theory

- Computation theory is a different, more structural and less statistical approach to complexity, emergence, organization.
- Computation theory can be very elegant, rigorous, and mathematical.
- But I'll present little of the formalism. I think the math can obscures some of the basic ideas, which are really quite simple.

We'll begin with some examples in the form of a game:

- I'll give you the speci£cation for a set
- I'll then show you an object, and you need to tell me if it's in the set or not

## Example 1

The set $\mathcal{L}$ consists of all sequences of $0$'s and $1$'s of any length, except for those that have two $00$'s in a row.

Accept all sequences of $1$'s and $0$'s except for those which have two or more $0$'s in a row.

$$1110101101$$

$$1101101001$$

$$110110101011$$

## Example 2:

The set $\mathcal{L}$ consists of all sequences of correctly balanced parentheses.

( ( ) ( ) )

( ( ) ( ) ( ) ) ( ( ) )

( ( ) ( ( ) ( ) ( ) ) )

This example is harder.

**Example 3:**

The set $\mathcal{L}$ consists of all sequences of $0$'s and $1$'s, except for those that contain a **prime** number of consecutive $0$'s!

$$1100011001$$

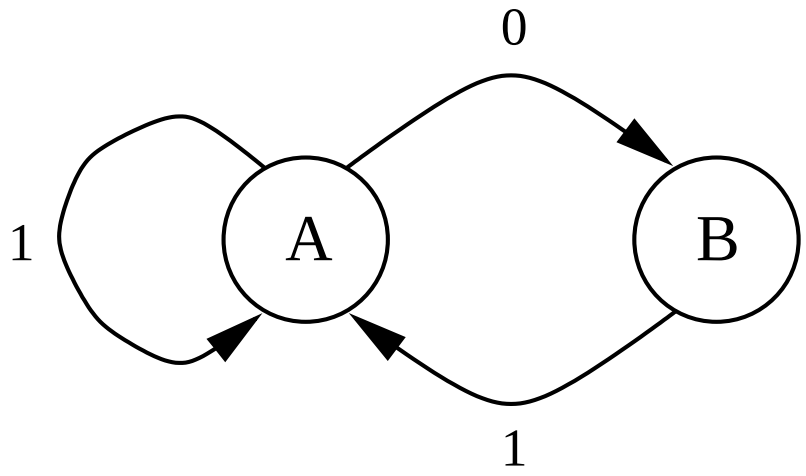
$$11000011$$

$$1110000000000001$$

$$11\overbrace{00\cdots00}^{1031\text{ elements}}11$$

## What to learn from the examples

- There are qualitative differences between the procedures you just used to identify the strings on the previous slides.
- These distinctions lie at the heart of computation theory.
- We'll start by focusing on example 1.
- Your task was to accept all sequences of $1$'s and $0$'s except for those which have two or more $0$'s in a row.




- Sequence is OK if there exists a path through this machine as one scans left to right.
- Example: $1011001$ is not in the set.

## Finite State Machines

- The mathematical object on the previous page is known as a **Finite State Machine** or a **Finite Automaton**.
- Note that this two state machines can correctly identify arbitrarily long sequences.
- The machine is a £nite representation of the in£nite set $\mathcal{L}$.

### Some terminology and de£nitions

- A **Language** $\mathcal{L}$ is a set of words (symbol strings) formed from an **Alphabet** $\mathcal{A}$.
- We'll always assume a binary alphabet, $\mathcal{A} = \{1, \infty\}$.

**Big Idea:** There is a correspondence between the rules needed to generate or describe a language, and the type of machine needed to recognize it.

## Regular Expressions

- A **Regular Expression** is a way of writing down rules that generate a language.
- To generate a regexp, start with the symbols in $\mathcal{A}$.
- You can make new expressions via the following operations: grouping, concatenating, logical OR (denoted $+$), and closure $*$.
- Closure means $0$ or more concatenations.
- Examples:
  1. $(0+1) = \{0, 1\}$
  2. $(0+1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$
  3. $(01)^* = \{\epsilon, 01, 0101, 010101, \ldots\}$
- ($\epsilon$ is the empty symbol. )

## Regular Languages and FSM

- A language $\mathcal{L}$ is a **Regular Language** if and only if it can be generated by a regular expression.
- A puzzle: what is the regular expression that generates the language of example 1?

Two important results:

1. For any regular language, there is an FSM that recognizes it.
2. Any language generated by an FSM is regular.

Notes on terminology:

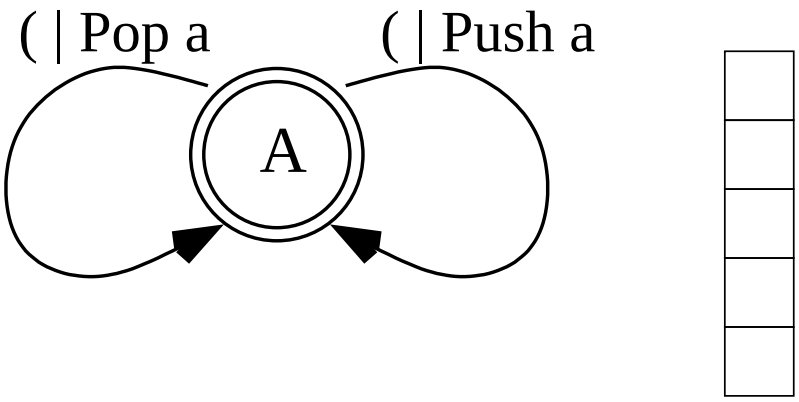
- A regular expression is a **rule**.
- A regular language is a **set**.
- A FSM is a **machine**.

Regular languages $\leftrightarrow$ FSM's is the £rst example of the correspondence between sets and the procedures or machines needed to recognize them.

## Revisiting Parentheses

- This example is different than the last—you can't scan left to right unless you remember stuff.
- There is no FSM that can recognize this language. The problem is that as the strong grows in length, the number of states necessary also grows.
- This task requires in£nite memory. However, the memory only needs to be organized in a simple way.
- The parentheses language can be recognized by a device knowns as a **Pushdown Automata**.
- Put an object on the stack if you see a left paren $($ and take it off if you see a right paren $)$.
- If the stack is empty after scanning the sequence, then it is ok.

## Pushdown Automata




- This is the PDA for the parentheses example
- If you see a “(“, write (push) a symbol to the stack.
- If you see a “)“, erase (pop) a symbol from the stack.
- The machine can only write to the top of the stack.

## Context-Free Languages

- The languages recognized by PDA are **context-free languages.**
- Regular languages are generated sequentiality—one symbol after the next.
- CFL's are generated by writing rules applied in parallel.
- For example, to generate the parentheses language, apply the following:

$$\begin{aligned} W &\rightarrow (V \\ V &\rightarrow (VV \text{ or } ) \end{aligned}$$

- Start with $W$. The set of all possible applications of the above rules give you the set of all possible balanced parentheses.
- For example:

$$W,\ (V,\ ((VV,\ (()(VV,\ (()()($$

## CFL Terminology

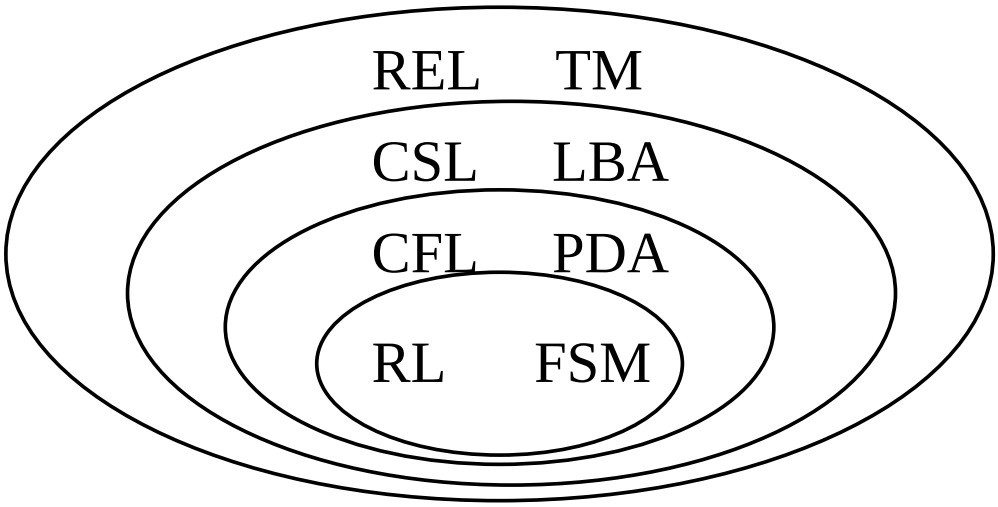
- $(, )$ are **terminals**, symbols in the alphabet $\mathcal{A}$.
- $W, V$ are **variables**, symbols not in $\mathcal{A}$, to be eventually replaced by terminals.
- CFL's are context free in the sense that the production rule depends only on the variable, not on where the variable is in the string.

## CFL Summary

- Every CFL can be recognized by a PDA, and every PDA produces a CFL.
- Also, FSM's are a proper subset of PDAs, and
- CFL's are a proper subset of Regular Languages
- We can thus divide languages into two classes, one of which is strictly more complex than the other.
- Are there even more complex languages? Yes ...

## Chomsky Hierarchy

- The hierarchy continues:




- This hierarchy of languages/machines is known as the **Chomsky Hierarchy**.

- Each level in the hierarchy contains something new, and also contains all the languages at lower levels of the hierarchy.

## Chomsky Hierarchy, terminology

- CSL = **Context Sensitive Language**. These are like CFS's, but allow transitions that depend on the position of the variable in the strings.
- LBA = **Linear Bounded Automata**. These are like PDA's, except:
  1. Controller can write anywhere on work tape.
  2. Work tape restricted to be a linear function of input.
- **Recursively Enumerable Languages** are those languages produced by an unrestricted grammar.
- An **Unrestricted Grammar** is like a CSL, but allows substitutions that shrink the length of the string.
- TM = **Turing Machines**. These are LBA's with linear tape restriction removed. These are the most powerful model of computation. (Example 3 requires a TM.) Much more on these later!

## Chomsky Hierarchy, Conclusions

- Order languages (sets) by the type of machine needed to recognize elements of the language.
- There are qualitative difference between machines at different levels of the hierarchy.
- At lower levels of the hierarchy, there are algorithms for minimizing machines. (I.e., remove duplicate nodes.)
- The minimum machine can be viewed as a representation of the pattern contained in the language. The machine is a description of all the regularities.
- The size of the machine may be viewed as a measure of complexity.
- The machine itself reveals the “architecture” of the information processing.

## Other computation theory notes

- It is possible to re£ne the Chomsky hierarchy with different sorts of machines. The result is a rich partial ordering of languages.
- To use computation theory as a basis for measuring complexity or structure, I think it’s important to start at the bottom of the hierarchy and work your way up.

## Computation Theory References

The basic material presented is quite standard and there are many references on it. Here are a few:

- Hopcroft and Ullman. Introduction to Automata Theory, Languages and Computation. Addison-Wesley. 1979. *A standard reference. Not my favorite, though. It's thorough and clear, but rather dense.*

- Brookshear. Theory of Computation: Formal Languages, Automata, and Complexity. Benjamin/Cummings. 1989. *I like this book. I £nd it much clearer than Hopcroft and Ullman.*

Computation theory applied to physical sequences

- Badii and Politi. Complexity: Hierarchical Structures and Scaling in Physics. Cambridge. 1997. *Excellent book, geared toward physics grad students. Closest thing to a textbook that covers topics similar to those I've covered throughout these lectures.*

- Bioinformatics textbooks?