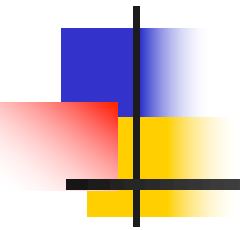
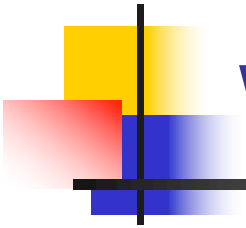
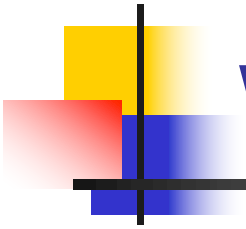


Review and Wrap-up



Why study theory of computation?

- *Theory is relevant to practice*
- *Theory is relevant to you*
- *Theory is good for you*



Why study theory of computation?

- *1) Theory is relevant to practice*
  - Designing a new programming language for a specialized application
    - Grammars
  - String searching and pattern matching
    - Automata and regular expressions
  - Intractable problems
    - NP-completeness

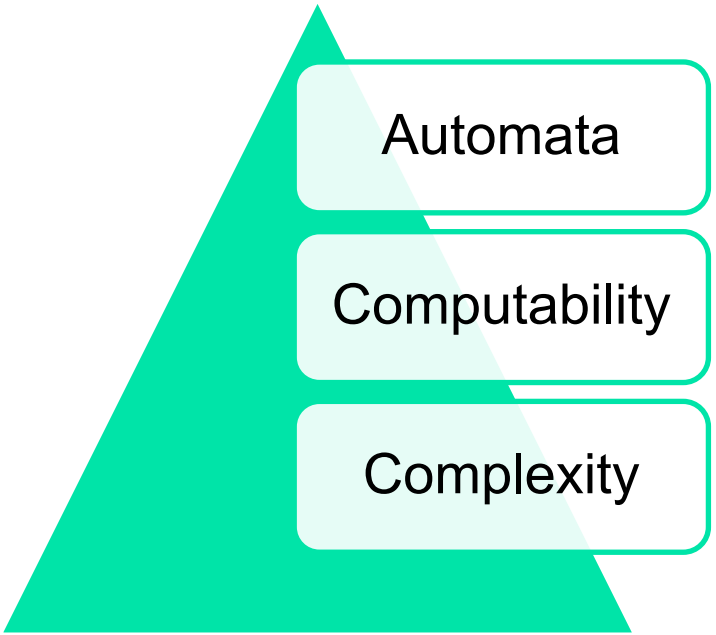
Why study theory of computation?

- *2) Theory is relevant to you*
  - It shows you a new, simpler, and more elegant side of computers
  - A theoretical course can heighten your aesthetic sense and help you build more beautiful systems
- *3) Theory is good for you*
  - Studying it expands your mind

- Side note: interesting recent article

https://www.quantamagazine.org/landmark-computer-science-proof-cascades-through-physics-and-math-20200304/

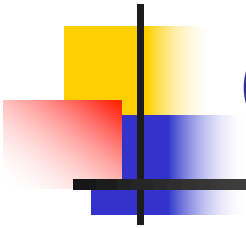
Three central areas of the theory of computation



- Linked by this question:

What are the fundamental capabilities and limitations of computers?

- In each of the three areas the question is interpreted differently, and the answers vary according to the interpretation.

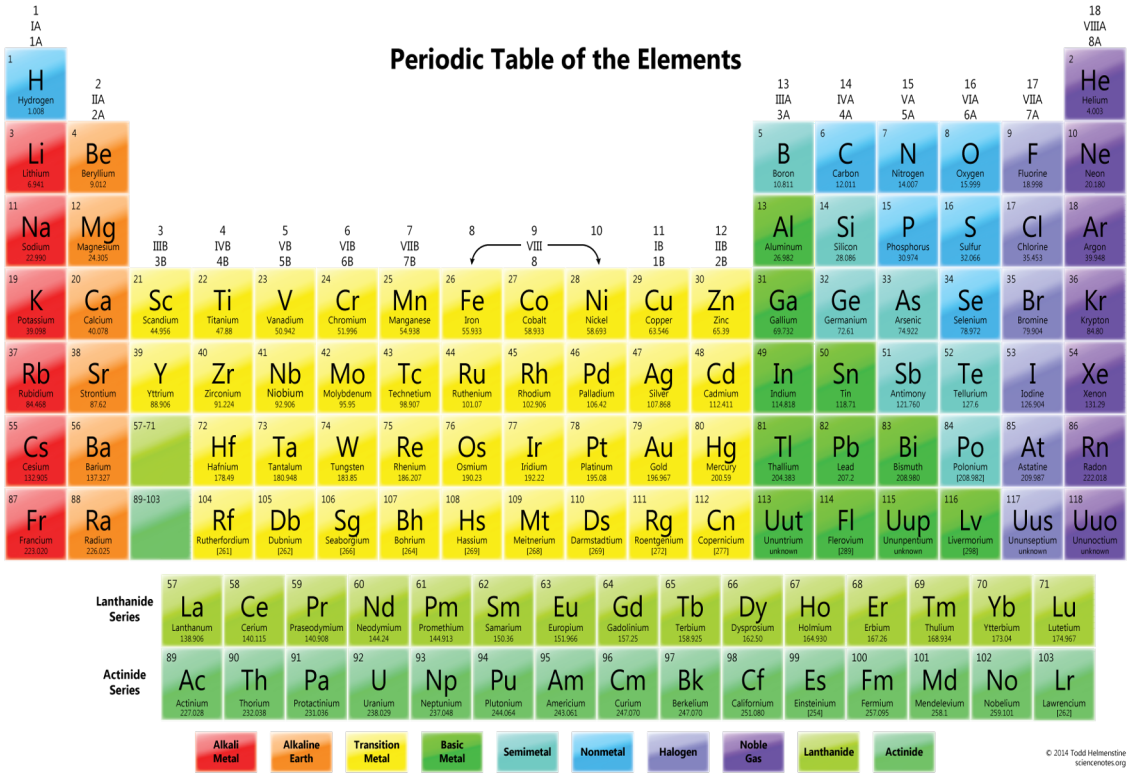


Complexity theory

- Computer problems are not created equal; some are easy, others are hard
- Example:
  - Sorting (easy)
  - Scheduling (much harder)
- Central question of complexity theory:
  - *What makes some computational problems hard and others easy?*
- *Bad news: we don't know the answer to this question yet (despite intensive research since the 1970s)*
- *Good news: researchers have come up with an elegant scheme for classifying problems according to their computational difficulty*

Complexity theory: Classification of problems

Much like the periodic table is used to used to classify elements according to their chemical properties, complexity theory classifies problems according to their computational difficulty.



Complexity theory: practical utility

Informing us how to deal with a problem that is computationally hard

- Identify aspect of the problem at the "root" of the difficulty, and *alter* it to make the problem easier to solve
- Settle for *less than perfect solution*
  - Approximation/heuristics
- Hardness may only be a *worst-case phenomenon*
- Consider *alternative* types of computation
  - Randomization

Use in cryptography

Computability theory

- Mathematicians such as Gödel, Turing, and Church discovered that certain basic problems cannot be solved by computers.
- One example is the problem of determining whether a mathematical statement is true or false.
- Among the consequences of this profound result was the development of ideas concerning theoretical models of computers.

Kurt Gödel
(1906--1978)

Alan Turing
(1912--1954)

Alonzo Church
(1903--1995)

Computability vs Complexity Theory

The theories of computability and complexity are closely related

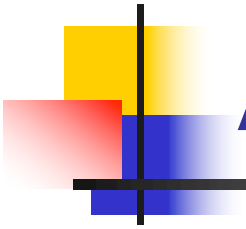
Complexity theory

Objective is to classify problems as easy ones and hard ones

Computability theory

Objective is to classify problems as solvable or not solvable

Computability theory introduces several of the concepts used in complexity theory



Automata theory

- Deals with the definitions and properties of mathematical models of computation
- These models play a role in several applied areas of computer science
- One model, called finite automaton, is used in
  - Text processing
  - Compilers
  - Hardware design
- Another model, called context-free grammar, is used in
  - Programming languages
  - Artificial intelligence

Topics covered:
Part One: Automata and Languages (about 60% of the lectures)

- Intro
  - Automata, Computability and Complexity
  - Mathematical notions
  - Definitions, Theorems, and Proofs
- Regular Languages
  - Finite Automata
  - Nondeterminism
  - Regular Expressions
  - Nonregular Languages
- Context-Free Languages
  - Context-Free Grammars
  - Pushdown Automata
  - Deterministic Context-Free Languages

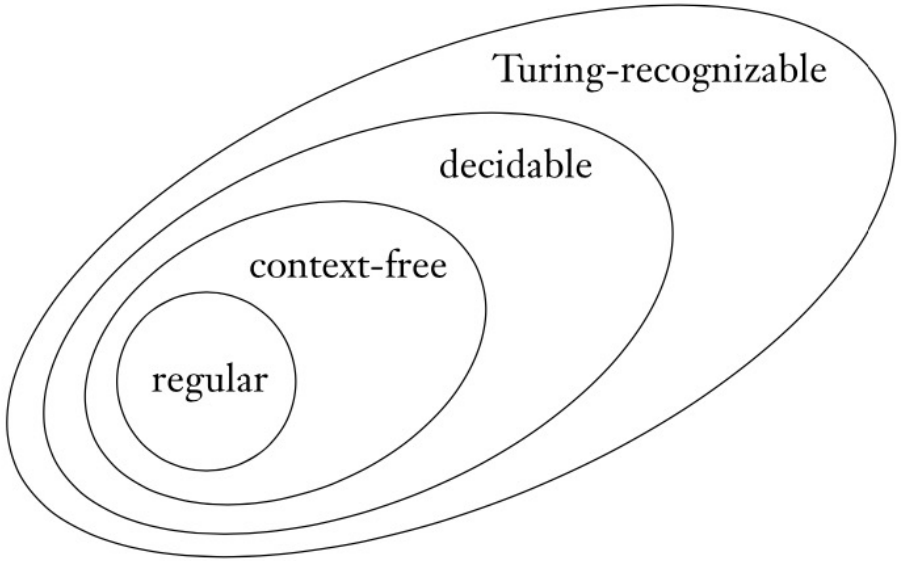
Topics covered:
Part Two: Computability Theory
(about 35% of the lectures)

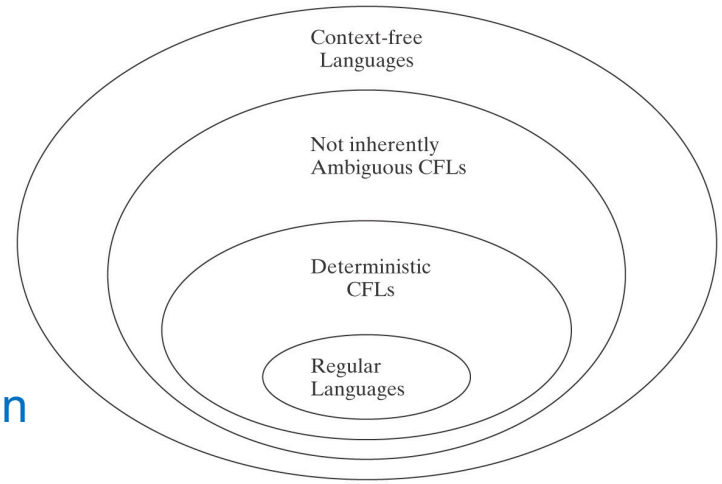
- The Church-Turing Thesis
  - Turing Machines
  - Variants of Turing Machines
  - The Definition of Algorithm
- Decidability
  - Decidable Languages
  - Undecidability
- Reducibility
  - Undecidable Problems from Language Theory
  - Mapping Reducibility

Topics covered:
Part Three: Complexity Theory
(two lectures)

- Time Complexity
  - Measuring Complexity
    - Big-O and small-o notation
    - Analyzing algorithms
    - Complexity relationships among models
  - The Class P
  - The Class NP

Relationship among classes of languages





zooming in on CFLs

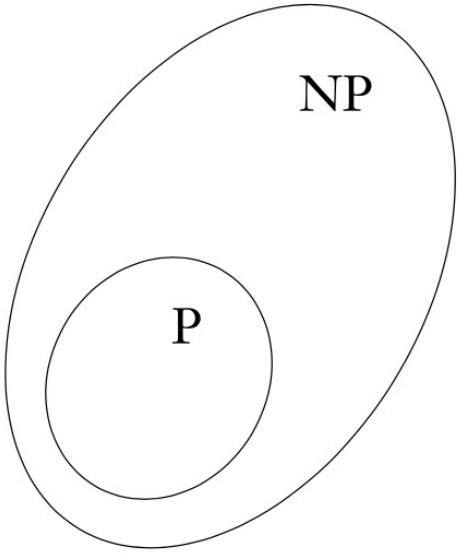
The P vs NP question

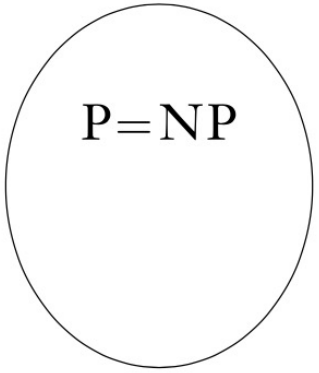
P = the class of languages for which membership can be *decided* quickly.

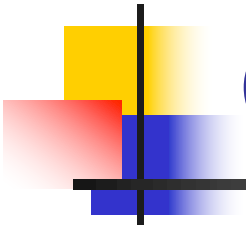
NP = the class of languages for which membership can be *verified* quickly.

Whether P = NP is one of the greatest unsolved problems in theoretical computer science.

One of the two possibilities depicted in this figure is possible.

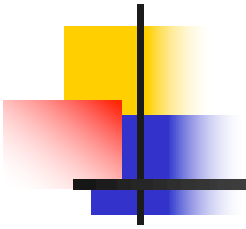






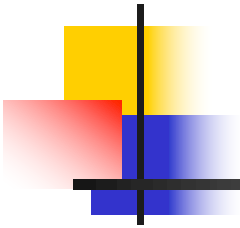
Coursework and grading

- 7 homeworks (58%)
  - Best 6 out of 7 will be used toward final grade
- 2 midterm exams (20%)
  - Mid term 1 (10%)
  - Mid term 2 (10%)
- 1 final exam (20%)
- Class participation (2%)



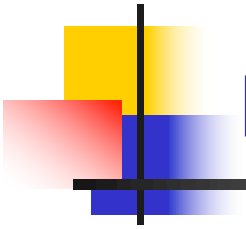
Final exam

- Closed-book, written exam
  - **Date**: *Monday May 2, 10:30am – 12:30pm*
  - **Place**: *SPRK G0010* (same room we meet for class)
- Will have 8 questions
  - 4 of them have no parts
  - 4 of them have multiple parts



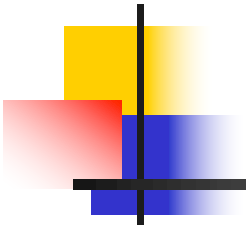
Final exam: questions nature

- **Automata and Regular Expressions**
  - Convert *this* NFA into an equivalent DFA
  - Provide an English language description of the language accepted by this NFA
  - Give a regular expression generating this language
- **Context free grammars and languages**
  - Write a CFG generating *this* language
  - Convert this CFG into an equivalent Push Down Automata
- Show that *this operation* is closed for *this language*
  - *language:* regular, CFG
- Given this DFA, construct a minimum state equivalent DFA using the **Table Filling Algorithm**



Final exam: questions nature (Cont'd)

- **Turing Machines**
  - Give an implementation level description of a TM that accepts *this language*
- **Decidability, Complexity theory**
  - Is this statement on decidability True or False?
  - Is this statement on big-o/small-o notation True or False?
  - Is this (basic) statement on *time complexity* True or False?



Final words

- Prep for final
  - Review lectures (notes/slides + videos)
  - Review all 7 HWs (prompts, solution hints)
- You have done a great job so far, and you got this!

And one last word…

- Remember to complete the course evaluation on myWSU
- Thanks!