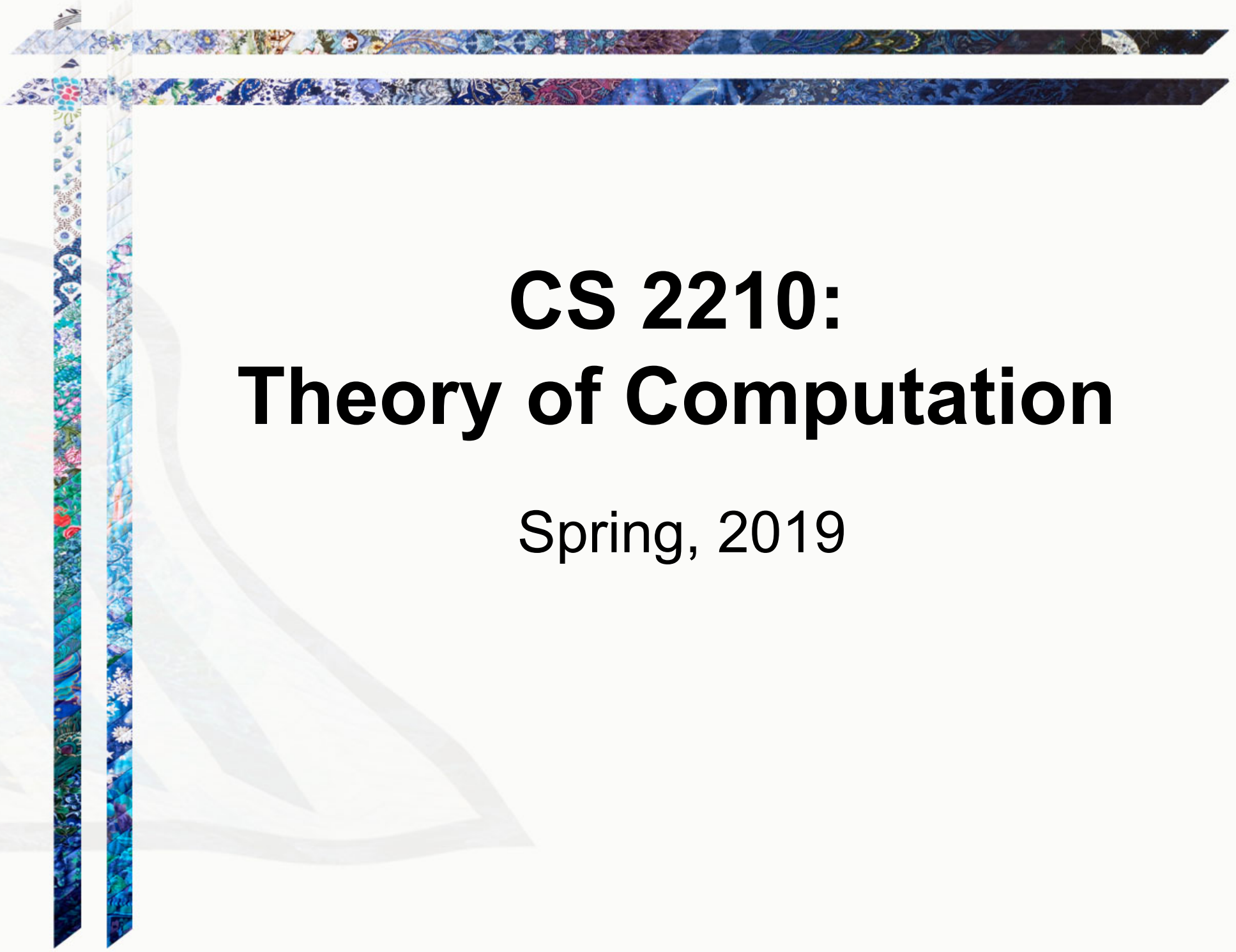


CS 2210: Theory of Computation

Spring, 2019

Administrative Information

- Background survey
- Textbook: E. Rich, *Automata, Computability, and Complexity: Theory and Applications,* Prentice-Hall, 2008.
- Book website: http://www.cs.utexas.edu/~ear/cs341/automatabook/
- My Office Hours:
  - Monday, 6:00-8:00pm, Searles 224
  - Tuesday, 1:00-2:30pm, Searles 222
- TAs
  - Anjulee Bhalla: Hours TBA, Searles 224
  - Ryan St. Pierre, Hours TBA, Searles 224

What you can expect from the course

- How to do proofs
- Models of computation
- What's the difference between computability and complexity?
- What's the Halting Problem?
- What are P and NP?
- Why do we care whether P = NP?
- What are NP-complete problems?
- Where does this make a difference outside of this class?
- How to work the answers to these questions into the conversation at a cocktail party…

What I will expect from you

- Problem Sets (25%):
  - Goal: Problems given on Mondays and Wednesdays
  - Due the next Monday
  - Graded by following Monday
  - A learning tool, not a testing tool
  - Collaboration encouraged; more on this in next slide
- Quizzes (15%)
- Exams (2 non-cumulative, 30% each):
  - Closed book, closed notes, but…
  - Can bring in 8.5 x 11 page with notes on both sides
- Class participation: Tiebreaker

Other Important Things

- Go to the TA hours
- Study and work on problem sets in groups
- Collaboration Issues:
  - Level 0 (In-Class Problems)
    - No restrictions
  - Level 1 (Homework Problems)
    - Verbal collaboration
    - But, individual write-ups
  - Level 2 (not used in this course)
    - Discussion with TAs only
  - Level 3 (Exams)
    - Professor clarifications only

Right now…

- What does it mean to study the "theory" of something?
- Experience with theory in other disciplines?
- Relationship to practice?
  - "In theory, theory and practice are the same. In practice, they're not."
- What would/should/could a theory of computation look like?
  - Would it be something like a "theory of refrigeration"?
  - What would make it useful?
  - In what way?
- Why should we care?

The Meaning of Life

"I went to the woods because I wished to live deliberately, to front only the essential facts of life, and see if I could not learn what it had to teach, and not, when I came to die, discover that I had not lived. I did not wish to live what was not life, living is so dear; nor did I wish to practice resignation, unless it was quite necessary. I wanted to live deep and suck out all the marrow of life, to live so sturdily and Spartan-like as to put to rout all that was not life, to cut a broad swath and shave close, to drive life into a corner, and reduce it to its lowest terms, and, if it proved to be mean, why then to get the whole and genuine meanness of it, and publish its meanness to the world; or if it were sublime, to know it by experience, and to be able to give a true account of it in my next excursion."

Walden, Henry David Thoreau

The Meaning of CS

"I went to CS 2210 because I wished to compute deliberately, to front only the essential facts of computation, and see if I could not learn what it had to teach, and not, when I graduated, discover that I had no idea what a Turing machine -- or NP-completeness -- was. I did not wish to try to compute what was not computable; nor did I wish to give up trying to find an efficient algorithm for a particular problem, unless it was quite necessary. I wanted to compute deeply and suck all of the theorems out of my axioms, to compute so abstractly and provably as to put to rout all that was not computation, to drive computation into a corner, and reduce it to its lowest terms, and, if some things proved to be not computable, why then to get the whole and genuine undecidability of it, and publish it to the world; or, if it were sublime, to know it by experience."

CS 289, Stephen Majercik

(with apologies to Thoreau)

Computer Science is…

IBM 7090 Programming in the 1950's

```
ENTRY       SXA     4,RETURN
            LDQ     X
            FMP     A
            FAD     B
            XCA
            FMP     X
            FAD     C
            STO     RESULT
RETURN      TRA     0
A           BSS     1
B           BSS     1
C           BSS     1
X           BSS     1
TEMP        BSS     1
STORE       BSS     1
            END
```

Programming in the 1970's (IBM 360)

```
//MYJOB    JOB (COMPRESS),
          'VOLKER BANDKE',CLASS=P,COND=(0,NE)
//BACKUP  EXEC PGM=IEBCOPY
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=MY.IMPORTNT.PDS
//SYSUT2   DD  DISP=(,CATLG),
           DSN=MY.IMPORTNT.PDS.BACKUP,
//         UNIT=3350,VOL=SER=DISK01,
//         DCB=MY.IMPORTNT.PDS,
           SPACE=(CYL,(10,10,20))
//COMPRESS EXEC PGM=IEBCOPY
//SYSPRINT DD  SYSOUT=*
//MYPDS    DD  DISP=OLD,DSN=*.BACKUP.SYSUT1
//SYSIN    DD  *
COPY INDD=MYPDS,OUTDD=MYPDS
//DELETE2 EXEC PGM=IEFBR14
//BACKPDS  DD  DISP=(OLD,DELETE,DELETE),
          DSN=MY.IMPORTNT.PDS.BACKUP
```

APL

```
(⌈/V)>(+/V)-⌈/V
```

Algorithms

- How many have had Algorithms?
- A step removed from programming
- How do we design algorithms to solve problems?
  - Divide and conquer
  - Greedy
  - Dynamic Programming
- How do we measure the “goodness” of an algorithm?
- But we’re still talking about specific problems

Theory

- Talking about ***classes*** of problems
- What can we say about ***all possible computations*** on ***all possible computers***?
- What makes some classes of problems ***computationally hard*** and others ***computationally easy***?
- What exactly do we get from ***resources like time and space***?
- What is ***possible/impossible***? (don't want to waste time trying to do impossible things)
- And we want to be able to ***prove our assertions***

How?

- Devise a simplified/abstract model of computation and say that computation is what this model can do!
- Seems like kind of a leap, but all these models are equivalent:
  - Lambda calculus
  - Turing machines
  - Markov algorithms
  - 2-stack automata
  - Post formal systems
  - Unrestricted grammars
  - Partial recursive functions
  - Recursively enumerable languages
- So it feels like we're not missing anything

Start With Simple Models

- Finite state automata
- Pushdown automata
- Many ideas carry over to Turing machines
- Easy to build/program, but have practical uses:
  - Pattern matching in text editors
  - Lexical analysis and language parsing in compilers
  - HTML parsing in web browsers
  - Graphical user interfaces in operating systems
  - Controlling NPCs in computer games
  - Control units in elevators, traffic signals, net protocol stacks
- Introduces notion of:
  - COMPUTATION = LANGUAGE RECOGNITION/ACCEPTANCE

Computation = Language Acceptance

- A machine accepts a language if, given a string composed of the symbols that language is based on:
  - The machine outputs yes if the string is in the language
  - The machine outputs no if the string is not in the language
- Can make comparisons of abstract machines based on language acceptance, e.g.
  - Machine A is **at least as powerful as** machine B if machine A can recognize all of the languages that B can.
  - Machine A is **more powerful than** B, if in addition, it can be programmed to recognize at least one additional language.
  - Two machines **are equivalent** if they can be programmed to recognize precisely the same set of languages.

Artificial? Restrictive?

- Complex problems can be reduced to language recognition problems, e.g. TSP
- TSP: given weighted graph **G = (V,E)**, what is the minimum cost circuit that visits every node exactly once?
- TSP = given weighted graph **G = (V,E)** and cost **C**, is there a circuit that visits every node exactly once that costs <= **C**?
  - Suppose we have a way of encoding **G** and **C** using characters from some alphabet
  - Suppose that we have a machine that "recognizes" those problem encodings for which the answer is yes
- Now we have a way of solving the TSP optimization problem

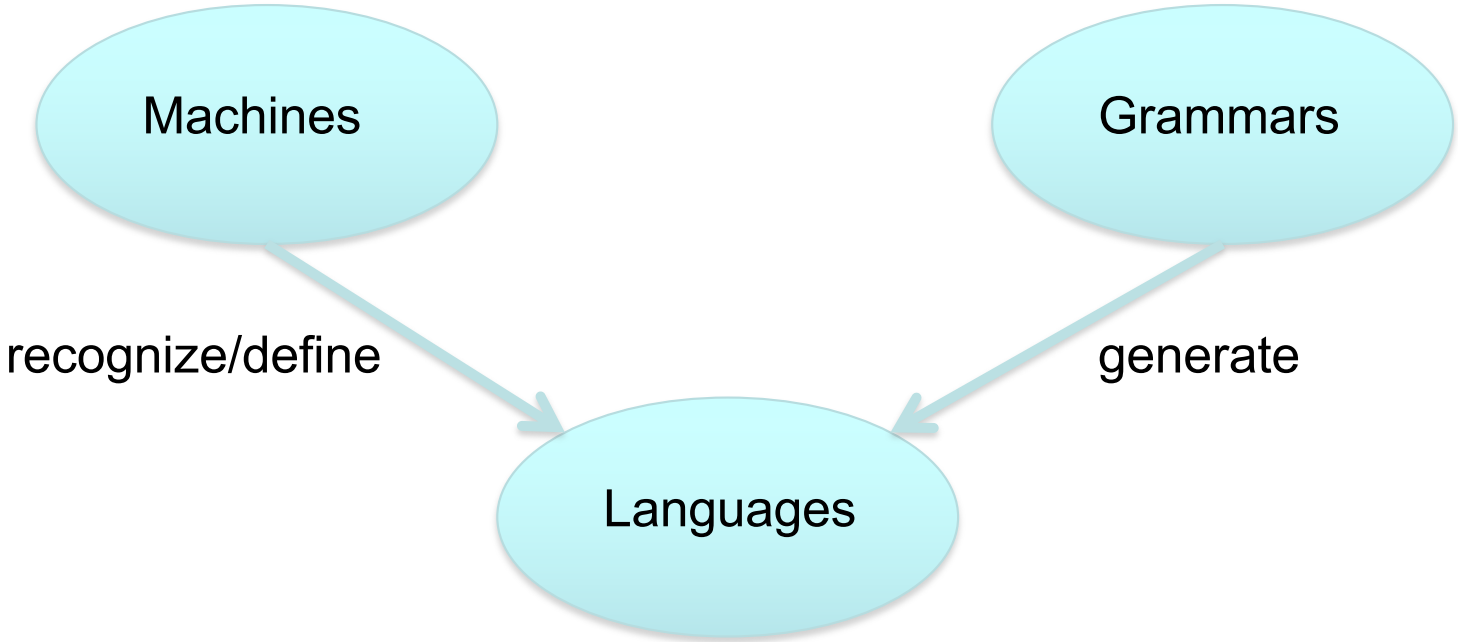
Traveling Salesperson

- Use binary search to find **C***, the minimum cost circuit
- Fine the actual circuit by removing one edge $\mathbf{e_i}$ at a time and using the machine to find out whether the graph without that edge **G = (V, E – {e_i})** still has a circuit of cost <= **C***
  - No? Put $\mathbf{e_i}$ back in
  - Yes? Leave $\mathbf{e_i}$ in
- End up with only those edges in the minimum cost circuit

Grammars

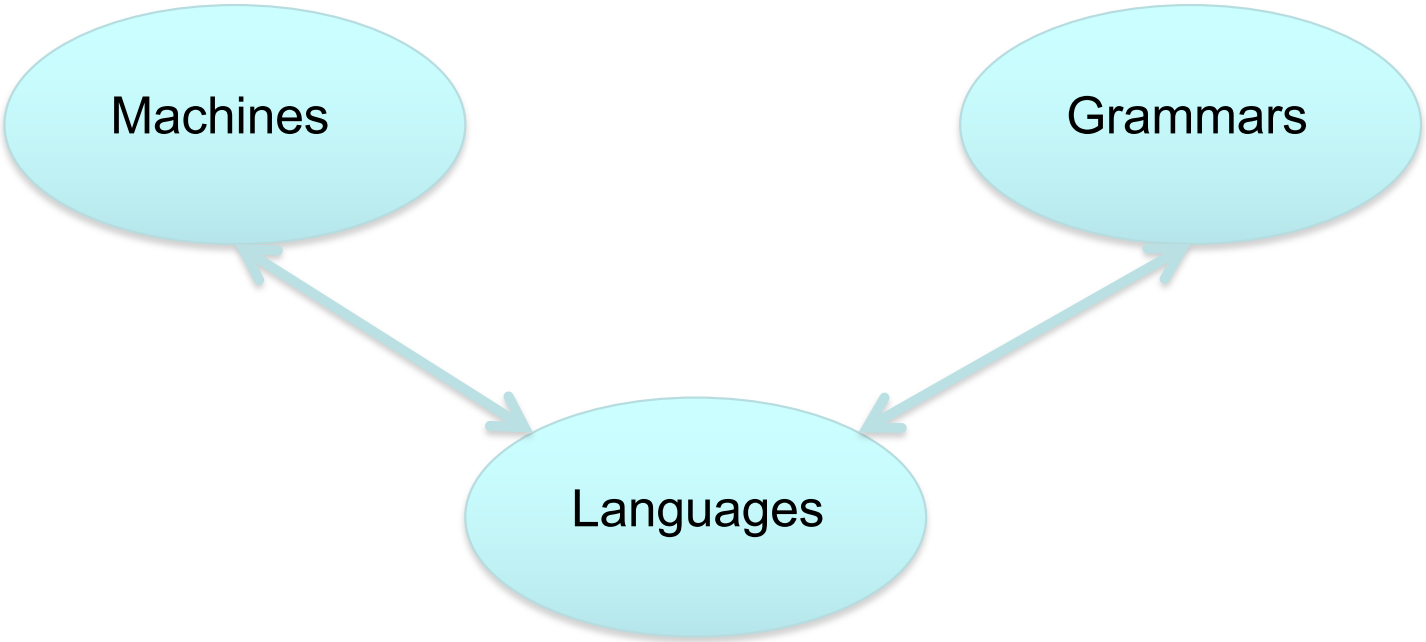
- Grammars can generate languages
- Like English, only a *lot* simpler

Three Kinds of Entities



- Machines recognize Languages
- Grammars generate Languages

Can Go in Both Directions



- Machines of a particular type recognize languages in a particular class and languages in that class have machines that recognize them
- Grammars of a particular type generate languages in a particular class and languages in that class have grammars that generate them

Classes of Languages

- Regular Languages =
  Finite State Automata =
  Regular Grammars
- Context-Free Languages =
  Pushdown Automata =
  Context-Free Grammars

- NOTE: Notions of finite automata and regular expressions were developed with models of neuron nets (biologists) and switching circuits (electrical engineers) in mind.

Hierarchy

Machines	Languages	Grammars
• FSA	• Regular	• Regular
• N-PDA	• Context-Free	• Context-Free
• Linear Bounded Automata	• Context-Sensitive	• Context-Sensitive
• Turing machines	• Recursively Enumerable (Semidecidable)	• Unrestricted

Hierarchy

Machines	Languages	Grammars
• **FSA**	• **Regular**	• **Regular**
• **N-PDA**	• **Context-Free**	• **Context-Free**
• Linear Bounded Automata	• Context-Sensitive	• Context-Sensitive
• **Turing machines**	• **Recursively Enumerable (Semidecidable)**	• **Unrestricted**

Value of Theory

- Slick program in CS 1101 with great documentation
- A
- C program in CS 2101 with pointers that don't crash the program
- A-
- Neural network in CS 3445 that learns how to classify all 34,874 items in the test set with 90% accuracy
- A+
- Knowledge of the theoretical underpinnings of computer science
- Priceless

More concretely….

- Useful when it's 3 in the morning and your program just doesn't work?
- Unlikely
- Useful when you're part of a team working on a robotics project?
- Possibly
- Developing algorithms for data mining on web documents using graph representations?
- Definitely

Applications of the Theory

- FSMs for parity checkers, vending machines, communication protocols, and building security devices.
- Using FSMs to control NPCs in video games.
- Interactive games as nondeterministic FSMs.
- Programming languages, compilers, and context-free grammars.
- Natural languages are mostly context-free. Some speech understanding systems use probabilistic FSMs.
- Computational biology: DNA and proteins are strings.
- Artificial intelligence: the undecidability of first-order logic.

Summary

- Computation can be viewed as language recognition or acceptance
  - Machines are “equivalent” to Languages
  - Grammars are “equivalent” to Languages
  - Machines are “equivalent” to Grammars
- Start with severely restricted model of computation that cannot compute everything
- Loosen restrictions until we get a model that can compute everything that’s computable
- Go beyond:
  - Not everything is computable
  - Complexity theory