Week 14 - Wednesday

COMP 2230

Last time

- Simplifying finite-state automata
- Started context-free languages

Questions?

Assignment 6

Logical warmup

- Imagine there's a band of steel stretched tightly around the equator of the Earth
  - For the purposes of this problem, imagine that the equator of the Earth is a perfect circle, with no irregularities in height due to mountains, oceans, or anything else
- Now, let's add one yard of steel to the band, making it slightly longer, and imagine that it raises the steel off the equator by the same amount all around
- Is the band now high enough to …
  - Slip a playing card under it?
  - Put your hand under it?
  - Roll a baseball under it?

Context Free Languages

Context free grammars

- Instead of using regular expressions, a context free language is often described with a **grammar**
- A grammar is a formal system of rewriting rules consisting of
  - Terminals: symbols of the alphabet
  - Non-terminals: symbols that produce other sequences of terminals and non-terminals
- Grammars often start with the non-terminal starting symbol S
- Any string that can be derived from S through some sequence of rule rewrites is a string in the language

CFG examples

- Write a grammar that describes legal nesting of parentheses and braces in Java
  - Don't worry about the stuff that goes inside
- Write a grammar for legal mathematical expressions in Java using variables and integers, **+**, **–**, *******, **/**, and parentheses
- Write a grammar that corresponds to the same language defined by the regular expression $ab^*\ (a \mid bb)\ (ba)^*$

Pushdown automata

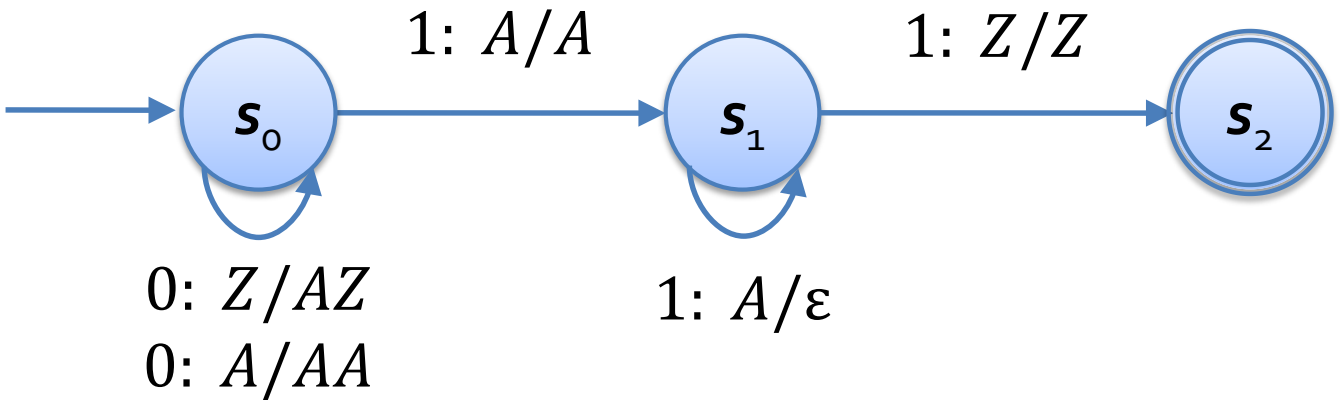
- As you know, regular languages can be expressed by regular expressions and finite state automata
  - Thus, regular expressions are equal to FSAs in power
- Context free languages can be expressed by context free grammars
- There is also a class of automata called **pushdown automata** that correspond to context free languages

Pushdown automata

- A pushdown automaton is an idealized machine with seven objects:
  - Q a finite set of states
  - Σ finite input alphabet
  - Γ the finite stack alphabet
  - δ is a finite subset of $Q \times (\Sigma \cup \varepsilon) \times \Gamma \times Q \times \Gamma^*$ which gives a set of transition rules
  - q_0 is the start state
  - $Z \in \Gamma$ is the initial stack symbol
  - $F \subseteq Q$ is the set of accepting states
- All of this looks totally insane, but it's really just adding a stack to finite state automata

Pushdown automaton example

- To make the PDA for the language $0^k 1^k$, $k \geq 1$, we have the following:



- In this case, the notation X/Y means that if X is on the top of the stack, replace it with Y
- The top of the stack is Z
- Notation is not well standardized for PDAs
  - It's awkward keeping track of the stack

Chomsky's Hierarchy of Grammars

Chomsky hierarchy

- Noam Chomsky is a brilliant linguist and advocate for anarchism
  - He was also a friend of Jeffrey Epstein
  - He's also 97 and had a stroke that keeps him at home
- Remember that a grammar consists of terminals (alphabet symbols), non-terminals, production rules, and a start symbol
- He noted that grammars can be divided into four levels in terms of expressiveness:
  - Type-0 (Unrestricted grammars)
  - Type-1 (Context sensitive grammars)
  - Type-2 (Context free grammars)
  - Type-3 (Regular grammars)

Rules for grammars

- Each grammar has rules for what is a legal production rule
- Let α, β, and γ be any combinations of terminals and non-terminals, where γ is non-empty

1. Unrestricted grammars
   - $\alpha \rightarrow \beta$ (anything to anything)
2. Context-sensitive grammars
   - $\alpha A \beta \rightarrow \alpha \gamma \beta$ (non-terminal in a particular context to anything)
3. Context-free grammars
   - $A \rightarrow \gamma$ (a single non-terminal to anything)
4. Regular grammars
   - $A \rightarrow a$ and $A \rightarrow aB$ (a single non-terminal to a single terminal or a terminal and a single non-terminal on the right side)

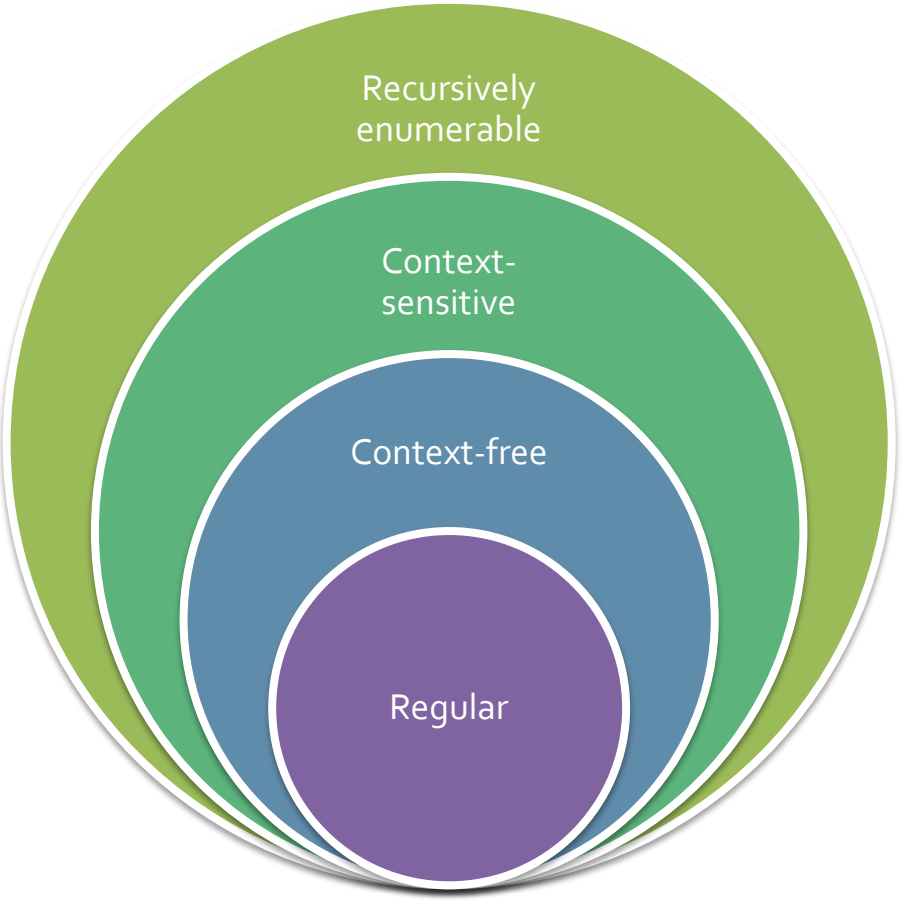
Languages broken down

- Every kind of language has a particular kind of machine associated with it

Grammar	Languages	Automaton	Production Rules	Examples
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$	All languages, any computable functions
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$	Most natural human languages
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$	Most programming languages
Type-3	Regular	Finite state automaton	$A \rightarrow a$ and $A \rightarrow aB$	Regular expressions

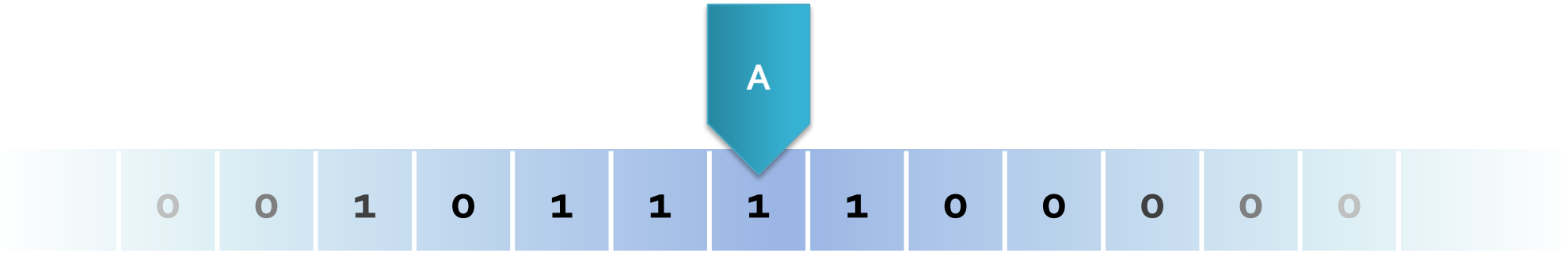
Fun facts about formal languages

- Each set of languages in the hierarchy strictly contains the sets beneath it
- Regular languages
  - Can be accepted by nondeterministic or deterministic finite automata (or a read-only Turing machine)
  - Are closed under union, intersection, complement, concatenation, and Kleene star
- Context-free languages
  - Are defined by those languages accepted by nondeterministic pushdown automata
  - Are closed under union, concatenation, and Kleene star (but not under intersection or complement)
  - Deciding whether a string is in a context-free language can be determined in polynomial time
  - Deciding whether a language is empty is decidable
- Context-sensitive languages
  - Are very rarely used
  - Are closed under union, intersection, complement, and Kleene star
  - Deciding whether a string is in a context-sensitive language is a PSPACE-complete problem



Turing machine

- A Turing machine is a mathematical model for computation
- It consists of a head, an infinitely long tape, a set of possible states, and an alphabet of characters that can be written on the tape
- A list of rules saying what it should write and should it move left or right given the current symbol and state



More formally

- A Turing machine is an idealized machine with seven objects:
  - Q a finite set of states
  - Σ the finite input alphabet
  - Γ the finite tape alphabet
  - δ is a finite subset of $Q \times (\Sigma \cup \varepsilon) \times \Gamma \times Q \times \Gamma^*$ which gives a set of transition rules
  - q_0 is the start state
  - $\square \in \Gamma$ is a special symbol called the blank
  - $F \subseteq Q$ is the set of accepting states
- Again, this is really just adding an infinite tape to a finite state automaton

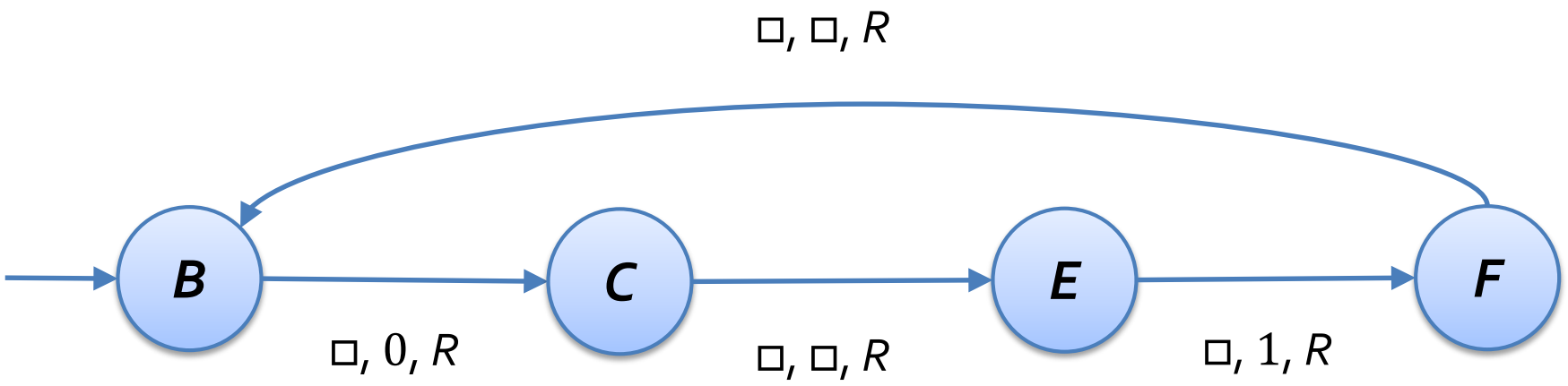
Turing machine example

- You can specify a Turing machine with a table giving its behavior for a specific configuration
- Turing's first example machine printed an infinite sequence of alternating 1s and 0s, separated by spaces:

Configuration		Behavior	
State	**Symbol**	**Operation**	**Result State**
B	Blank	Write 0, Move Right	C
C	Blank	Write Blank, Move Right	E
E	Blank	Write 1, Move Right	F
F	Blank	Write Blank, Move Right	B

A Turing machine as a transition diagram

- The transition table from the previous slide can be drawn as a transition diagram too:



Church-Turing thesis

- If an algorithm exists, a Turing machine can perform that algorithm
- In essence, a Turing machine is the most powerful model we have of computation
- Power, in this sense, means the ***ability*** to compute some function, **not** the ***speed*** associated with its computation
- Do you own a Turing machine?

Halting problem

- Given a Turing machine and input x, does it reach the halt state?
- First, recognize that we can encode a Turing machine as input for another Turing machine
  - We just have to design a system to describe the rules, the states, etc.
- We want to design a Turing machine that can read another

Turntables

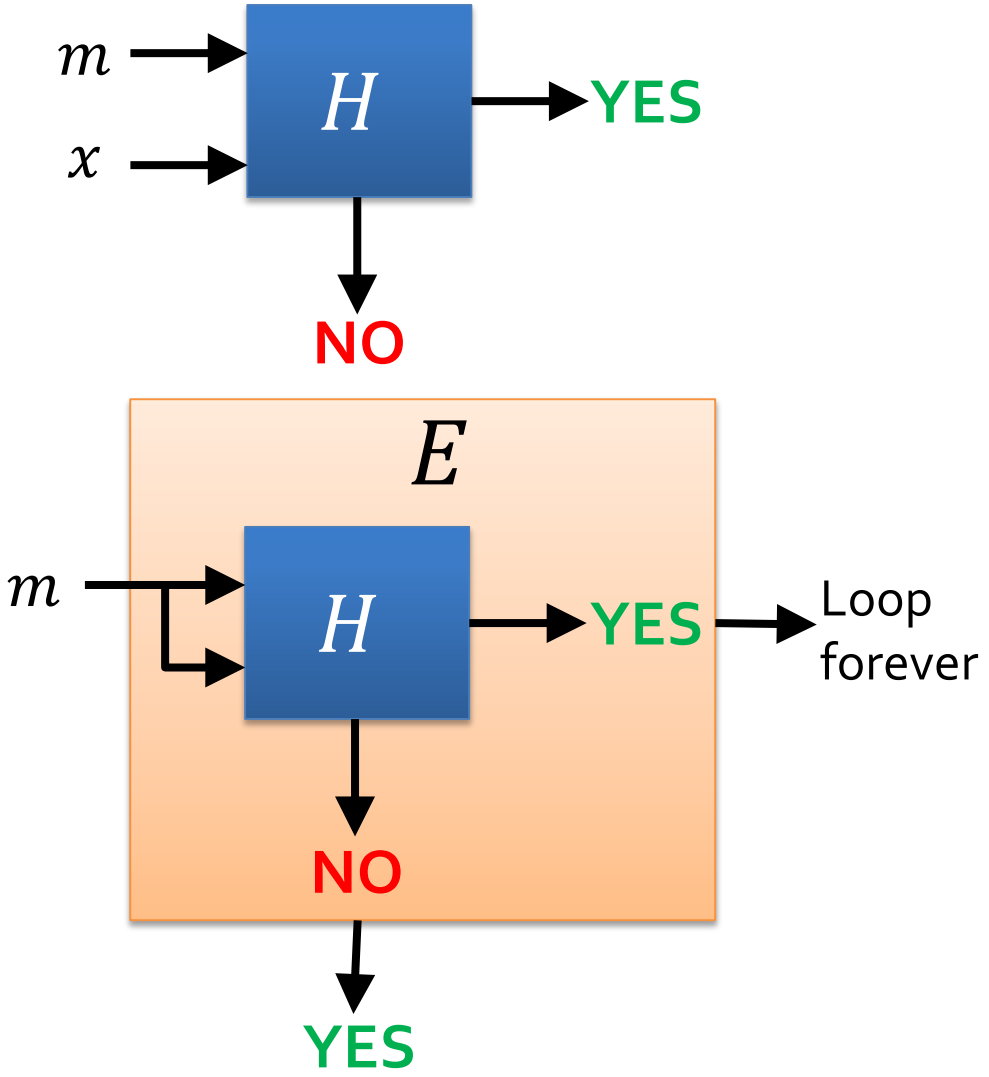
- Douglas Hofstadter uses the metaphor of turntables
- Imagine that evil people design records that will shake turntables apart when they're played
- Maybe turntable A can play record A and turntable B can play record B
- However, if turntable A plays record B, it will shatter

Stuff you have to buy for this proof

- Turing machines can perform all possible computations
- It's possible to encode the way a Turing machine works such that another Turing machine can read it
- It's easy to make a slight change to a Turing machine so that it gives back the opposite answer (or goes into an infinite loop)

Proof by contradiction

- You've got a Turing machine M with encoding m
- You want to see if M will halt on input x
- Assume there is a machine H that can take encoding m and input x
  - $H(m, x)$ is **YES** if it halts
  - $H(m, x)$ is **NO** if it loops forever
- We create (evil) machine E that takes description m and runs $H(m, m)$
  - If $H(m, m)$ is **YES**, E loops forever
  - If $H(m, m)$ is **NO**, E returns **YES**



A mind-bending proof

- Let's say that e is the description of E
- What happens if you feed description e into E?
  - $E(e)$ says what E will do with itself as input
- If it returns **YES**, that means that E on input e loops forever
  - But it can't, because it just returned **YES**
- If it loops forever, then E on input e would return **YES**
  - But it can't, because it's looping forever!
- Our assumption that machine H exists must be wrong

Halting problem conclusion

- Clearly, a Turing machine that solves the halting problem doesn't exist
- Essentially, the problem of deciding if a problem is computable is itself uncomputable
- Therefore, there are some problems (called **undecidable)** for which there is no algorithm
- Not an algorithm that will take a long time, but **no algorithm**
- If we find such a problem, we are stuck
- … unless someone can invent a more powerful model of computation

Post correspondence problem

- Given two finite lists of words A and B, can you pick k words (repetitions allowed) from A and k words (repetitions allowed) from B so that the words from A concatenated are exactly the same string as the words from B concatenated
- Example:

A	B
aa	$bbbb$
aba	a
bb	$abba$

- One solution:
  - $aa + bb + bb = aabbbb = a + a + bbbb$
- The Post correspondence problem (PCP) is **undecidable** (there is no algorithm that can solve all instances of it)

Other undecidable problems

- Are two context-free languages the same?
- Is the intersection of two context-free languages empty?
- Is a context-free language equal to Σ^*?
- Is a context-free language a subset of another context-free language?
- Is a given statement of first-order logic provable from a starting set of axioms?
- Given a set of matrices, is there some sequence that they can be multiplied in (perhaps with repetitions) that will yield the zero matrix?

More (useful) undecidable problems

- Does a given Java program have an infinite loop in it?
- Will a given Python program terminate regardless of what inputs it's given?
- Will computation with input x use all states of a Turing machine?
- Given input x, will the output of a C++ program be y?

Ticket Out the Door

Upcoming

Next time…

- Review of first third of the course

Reminders

- Keep working on Assignment 6