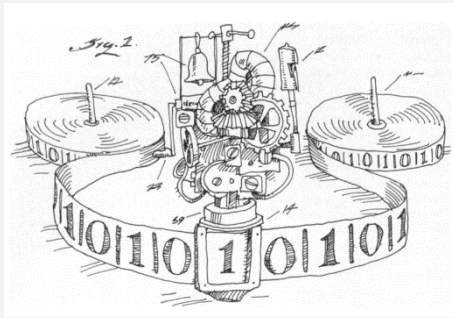
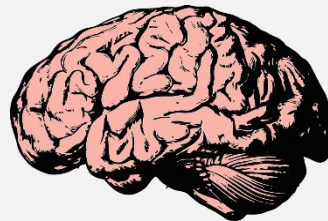
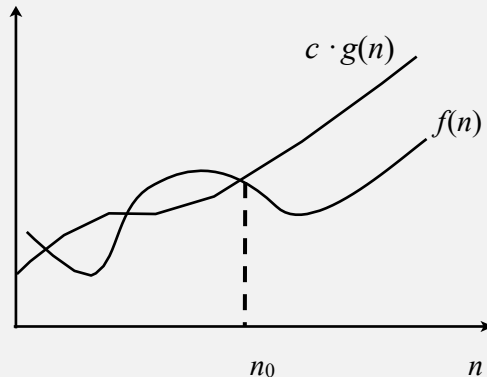


COMP 3602

Intro to Theory of Computation

Lecture 8



Logistics

- Test 1 day/time: Mon Feb 2
 - Written test on Turing machines, automata/regex, PDA/CFGs, pumping lemmas
 - Practice problems are up
 - One **letter-size double-sided handwritten** cheatsheet allowed
- Theory reading group
 - Stays Mondays 1pm for now, sorry!
- Drills are up, some due Feb 2nd.
 - Do the sample drills first!
 - You can do the drills multiple times, but the grade is an average of all attempts
 - So if you open it and let it time out, it will add a 0 to your average!

Practice problems questions?

1. Give regular expressions and draw automata (DFAs or NFAs) for the following languages:
 - a. $\{w \in \{0,1\}^* \mid w \text{ consists only of the string } 010\}$
 - b. $\{w \in \{0,1\}^* \mid w \text{ contains a substring } 010\}$
 - c. $\{w \in \{0,1\}^* \mid w \text{ contains a substring } 010, \text{ starts with } 0 \text{ and ends with } 1\}$
 - d. $\{w \in \{0,1\}^* \mid w \text{ does not contain a substring } 010\}$
 - e. $\{w \in \{a,b,c\}^* \mid w \text{ contains exactly 2 "c" and at least one "a" }\}$
 - f. $\{w \in \{a,b,c\}^* \mid \text{length of } w \text{ is divisible by 5 and it ends at "a" }\}$
2. Give NFAs with the specified number of states for the following languages, then use the subset construction to get DFAs out of them.
 - a. $L = 1^*$, one state.
 - b. Strings with all 1s (if any) in the same substring (That is, strings without a substring 100^*1). Three states.
 - c. $0^*(1000^*)^*$ with three states (here, "1000*" means 100 followed by 0^*)
3. Give a pushdown automaton and a context-free grammar for the following languages (for extra practice, try writing grammars for languages above)
 - a. $\{w \in \{0,1\}^* \mid \text{the length of } w \text{ is odd}\}$
 - b. $\{w \in \{a,b,c\}^* \mid w \text{ contains more c's than a's}\}$
 - c. $\{w \in \{0,1\}^* \mid w \text{ is a palindrome or } w=0^n u 0^n \text{ for some } n, \text{ where } u \in \{0,1\}^* \}$
4. Show using the pumping lemmas that
 - a. $\{a^n b \{a,b,c,d\}^* c d^{2n}\}$ is not regular.
 - b. $\{w \in \{0,1\}^* \mid \text{contains a substring } 10 \text{ or } w=0^n 1^m \text{ where } n \neq m \}$ is not regular. Hint: use closure properties of regular languages.
 - c. $\{ww^r w \mid w = w_1 \dots w_n \in \{0,1\}^* \text{ where } w^r = w_n \dots w_1 \text{ is the reverse of } w \}$ is neither regular nor context-free.
 - d. $\{1^{n^2}\}$ is neither regular nor context-free.

5. Describe Turing machines for the following languages. Use the definition of Turing machines with a single 2-way infinite tape, and movement directions $\{L,R\}$. Make sure to include both the informal outline of the algorithm, and the full description, that is, Σ , Γ , Q , q_0 , q_a , q_r and δ . Do not draw diagrams for Turing machines! Hint: first look at a few examples of strings that are in the language and strings that are not in the language, and don't forget the empty string.
 - $L_1 = \{w \in \{b,c,d\}^* \mid \text{non-empty strings of length at least 3 with the last three symbols all different.}\}$
 - $L_2 = \{w \in \{0,1\}^* \mid w=0^n1^n0^{2n} \text{ for } n \geq 0 \}$. That is, L_2 consists of all strings of length divisible by 4 (including length 0), where the first quarter of the string is all 0s, second quarter is all 1s, and the second half all 0s again.
6. Describe a Turing machine that decrements by 2 a natural number ≥ 2 written on its tape in binary. It should end in q_a pointing to the first symbol of the output if the tape contained a string starting with 1 encoding a natural number $x \geq 2$, and in q_r otherwise. Give both an informal algorithm and a full description of the TM.
7. Consider a TM where instead of "move left one cell" and "move right one cell" the only available moving operations are "jump left 11 steps", "jump right 5 steps" and "stay at the same place"; let's call this model 11L5RS Turing machines. Just as in the model of TM we defined in class, 11L5RS TMs have one two-way-infinite tape, the only difference is the movement operations.
 - Show that 11L5RS TM model is equivalent to the model we defined in class. Do both directions, with constructions (from M construct M'), and an outline of a proof why your construction works.
 - Show that if instead it were "jump left 10 steps" "jump right 5 steps", then this 10L5RS TM model would not be able to simulate the standard TM.
8. Show that single-tape Turing machines that only move right recognize exactly the regular languages. Hint: do both directions of the equivalence, and remember that the Turing machine is not guaranteed to stop.
9. Show that finite automata with two stacks can simulate Turing machines.

Do not draw diagrams for Turing machines!

Questions/comments?

Topics we covered so far

- Computational hardness
 - Representative problems: Is k even? prime? Its factors? Description?
 - Review of $O, \Omega, \Theta, o, \omega, \tilde{O}$
 - Upper and lower bounds for comparison-based sorting
- Models of computation
 - Turing machines (and their variants)
 - Church-Turing thesis
 - Non-deterministic and multi-tape TMs: cannot do more, but more efficient
 - Example of a proof of equivalence of two models: one-way vs. two-way infinite tape TMs
 - DFAs and NFAs, their equivalence
 - Pushdown automata (PDAs), deterministic $\neq$ non-deterministic
- Regular languages (exactly the class computed by DFAs)
- Context-free languages (exactly the class computed by PDAs)
- Proving hardness for regular and context-free languages: pumping lemmas

Regular languages vs. CFLs

Regular languages

- Class recognizable by NFAs
 - Non-determinism **is not** necessary
- Described by regular expressions
- Closed under regular operations, complement, intersection
- Pumping lemma:
 - $\exists p \forall s |s| \geq p \exists x, y, z \forall i \geq 0$
 - $s = xyz \wedge |xy| \leq p \wedge |y| > 0 \wedge xy^iz \in L$
- Some CFLs are not regular

Context-free languages

- Class recognizable by PDAs
 - Non-determinism **is** necessary
- Generated by context-free grammars
- Closed under regular operations but **not** complement and intersection
- Pumping lemma:
 - $\exists p \forall s |s| \geq p \exists u, v, x, y, z \forall i \geq 0$
 - $s = uvxyz \wedge vxy \leq p \wedge |vy| > 0 \wedge uv^ixy^iz \in L$
- All regular languages are context-free
 - Some non-CFLs are computable on a TM

Algorithms for regular languages and CFLs

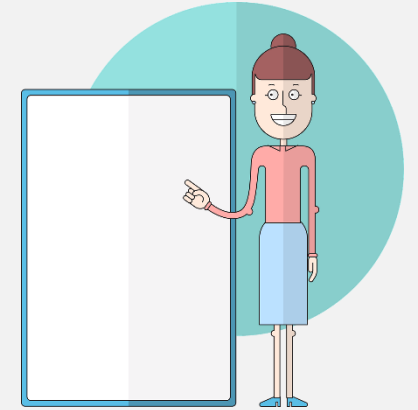
- Membership problem: given a string w and the description of the language as a DFA/NFA/regex/CFG/PDA, check if $w \in L$:
 - For DFAs: linear time algorithm
 - Efficient even for NFAs
 - So efficient on regular expressions, too.
 - Efficient for CFGs (and so PDAs):
 - Cocke-Younger-Kasami (CYK): dynamic programming algorithm

Algorithms for regular languages and CFLs

- Emptiness problem: given a DFA/NFA, does it accept anything?
 - Efficient for both DFAs and NFAs: reachability (use BFS/DFS)
- Learning and minimization
 - Minimization efficient for DFAs
 - Solvable, but not known to be efficient for NFAs and regular expressions
 - Learning algorithm for regular languages by Dana Angluin
 - Need to define a learning model carefully: the Student-Teacher model



Student-teacher game / counterexample computation



Regular languages can be learned *with equivalence queries* [Angluin'87].

Minimizing DFAs

- **Theorem:** for every regular language, there exists a unique minimal DFA
 - Unique up to state names.
 - Not true for NFAs! There may be multiple minimal NFAs.

Theorem: for every regular language, there exists a unique minimal DFA

- Proof idea:
 - Unreachable states are useless; can get rid of them.
 - A string w *distinguishes* states p and q in a DFA D if the computation on w starting at one of them (say, p) accepts, and starting at q rejects
 - If p, q are indistinguishable, then only need one of them!
 - Each set of indistinguishable states of D forms an equivalence class
 - From each such class, only need one state.
 - Suppose there are two minimal DFAs. Can construct a bijection from states of one to states of another.
- **Corollary:** there is an efficient algorithm for minimizing DFAs
 - Use BFS/DFS to find unreachable states and get rid of them
 - Use dynamic programming to figure out which states are indistinguishable

Corollary: there is an efficient algorithm for minimizing DFAs

- Use BFS/DFS to find unreachable states and get rid of them
- Use dynamic programming to figure out which states are indistinguishable