CS 456/656 Answers to Study Guide for Examination March 6, 2024

A *binary function* is defined to be a function F on binary strings such that, for each binary string w, $F(w)$ is a binary string. (Of course, the strings could be numerals.)

A real number x is defined to be *recursive* if there is a machine which runs forever writing the decimal expansion of x. There are other equivalent definitions of recursive real number.

Complementation Theorem: A class of languages decided by a class of machines is closed under complementation. If a machine decides a language L, it can be made to decide its complement by swapping the two outputs.

1. True or False. T = true, F = false, and O = open, meaning that the answer is not known science at this time.

   (i) **F** Every subset of a regular language is regular.

   Every language is the subset of some regular language.

   (ii) **F** The complement of a CFL is always a CFL.

   $L = \{a^n b^n c^n \,:\, n \geq 0\}$ is not CF, but it's complement is.

   (iii) **T** The class of context-free languages is closed under union.

   If G_1, G_2 are CF grammers for L_1 and L_2, add a subscript 1 to variables of G_1, and a subscript 2 to variables of G_2. Let G consist of all the productions of both grammars, together with $S \to S_1 \,|\, S_2$. Then $L(G) = L_1 \cup L_2$.

   (iv) **F** The class of context-free languages is closed under intersection. Let $L_1 = \{a^n b^n c^m \,|\, n, m \geq 0\}$ and $L_2 = \{a^n b^m c^m \,|\, n, m \geq 0\}$, both CF. $L_1 \cap L_2 = \{a^n b^n c^n \,|\, n \geq 0\}$, which is not CF.

   (v) **T** The language of binary numerals for multiples of 23 is regular.

   If leading zeros are allowed, you need 23 states in a DFA which decides the language. Otherwise, you need 24. The set of numerals of any base, not just 2, for the members of any arithmetic sequence is a regular language.

   (vi) **T** The set of binary numerals for prime numbers is in $\mathcal{P}$–TIME.

   The base doesn't matter, as long as its at least 2. (This excludes unary (caveman) numerals.) This is a fact that was proven only recently, by Maninda Agrawal, N. Kayal, and N. Saxena, and published in 2004, but I believe the result leaked out ealier. Before then, the correct answer to this question would have been **O**. On the other hand, it is very simple to write a polynomial time algorithm for primality of unary numerals.

   (vii) Duplicate question.

   (viii) **T** Every language is countable.

   There are only countably many strings over any given alphabet.

(ix) **F** The set of languages over the binary alphabet is countable.

Let Σ be any alphabet. Then Σ^* is the set of all strings over Σ, which is infinite and countable. But Cantor proved that, for any set S, the set 2^S has more elements than S. The set of all languages over any alphabet Σ is 2^{Σ^*}, which is not countable, by Cantor's diagonalization argument. This statement is also false for any alphabet.

(x) **O** $\mathcal{P} = \mathcal{NP}$.

Solve this and you will be **really** famous.

(xi) **T** The complement of any $\mathcal{P}$–TIME language is $\mathcal{P}$–TIME.

By the complementation theorem.

(xii) **O** The complement of any $\mathcal{NP}$ language is $\mathcal{NP}$.

This is an important open problem. If $\mathcal{P} = \mathcal{NP}$, then the answer is true.

(xiii) **T** The complement of any decidable language is decidable.

By the complementation theorem.

(xiv) **T** The complement of any undecidable language is undecidable. let L' be the complement of L. If L is undecidable and L' is decidable, this violates the answer to the previous question.

2. Give an unambiguous CFG which generates a language not accepted by any DPDA.

The folowing CFG generates all palindromes over $\{a, b\}$. No DPDA can accept this language, because it would not be able to detect the middle of its input string.

$S \to aSa$
$S \to bSb$
$S \to a$
$S \to b$
$S \to \lambda$

3. Suppose L is a problem such that you can check any suggested solution in polynomial time. Which one of these statements is certainly true?

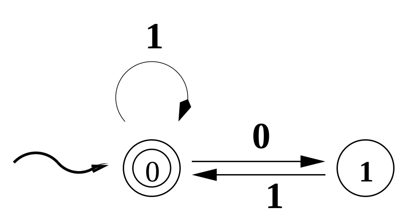
(a) L is $\mathcal{P}$.

(b) L is $\mathcal{NP}$.

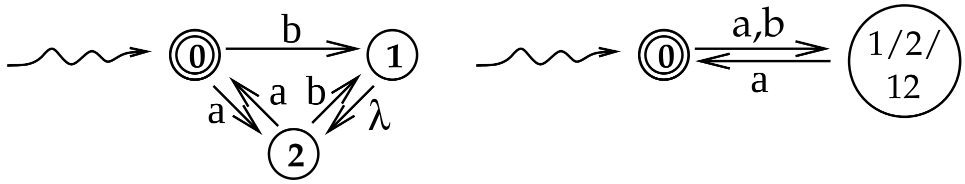
(c) L is $\mathcal{NP}$-complete.

Only the second one. But if $\mathcal{P} = \mathcal{P}$ and L is any infinite language, all three statements are true.

4. Let L be the language of all binary strings where each 0 is followed by 1. Draw a DFA which accepts L.

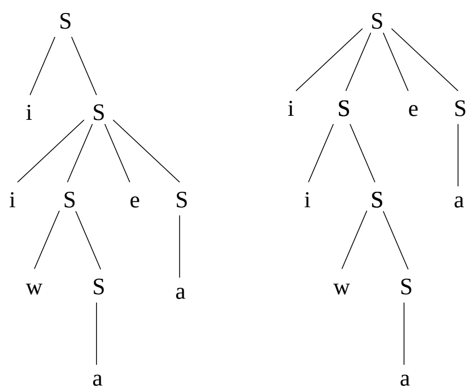


5. Consider the NFA M pictured below. Construct a minimal DFA equivalent to M.



6. Let G_1 be the CF grammar given below. Prove that G_1 is ambiguous by giving two different parse trees for the string *iiwaea*.

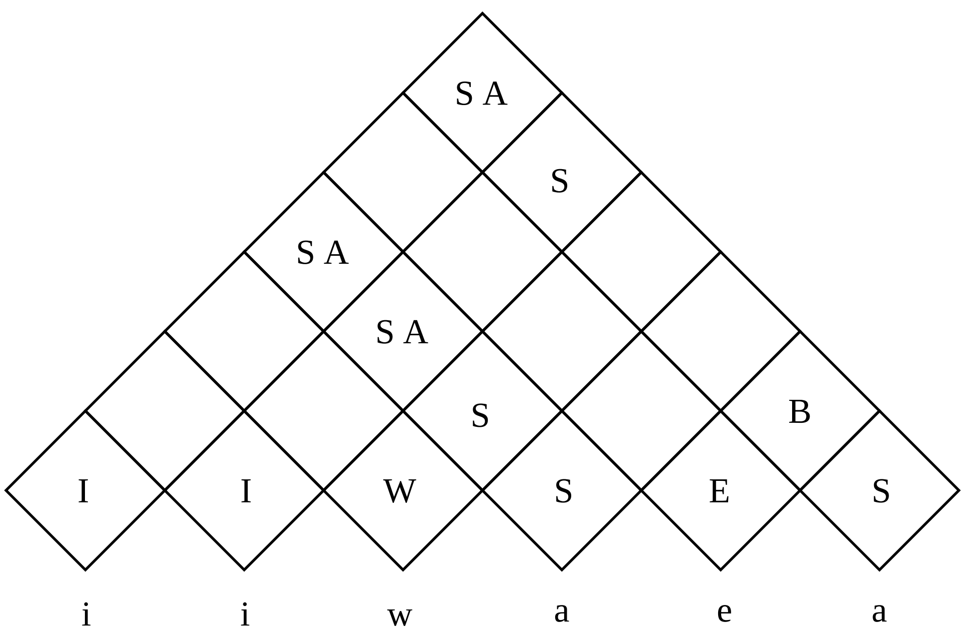
1. $S \to a$
2. $S \to wS$
3. $S \to iS$
4. $S \to iSeS$



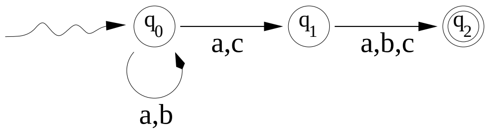
7. The CNF grammar G_2, given below, is equivalent to the grammar G_1 given in Problem 6.

   Use the CYK algorithm to prove that *iiwaea* is generated by G_2.

1. $S \to a$
2. $S \to WS$
3. $W \to w$
4. $S \to IS$
5. $S \to AB$
6. $A \to IS$
7. $B \to ES$
8. $E \to e$
9. $I \to i$



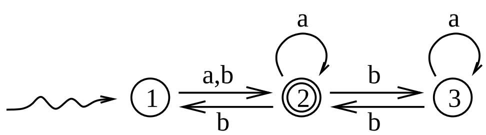
8. Give a grammar, with at most 3 variables, for the language accepted by the follwing NFA.



$S \to aS|bS|aA|cA$
$A \to aB|bB|cB$
$B \to \lambda$

You actually need only one variable. Do you see how?
$S \to aS\,|\,bS\,|\,aa\,|\,ab\,|\,ac\,|\,ca\,|\,cb\,|\,cc$

9. Give a regular expression for the language accepted by the following NFA

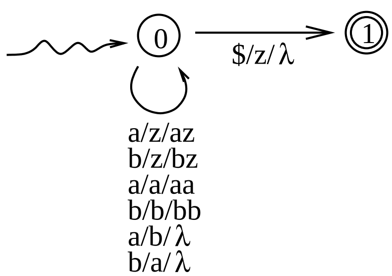


$(a+b)(b(a+b)+a+ba^*b)^*$

10. Let L be the language consisting of all strings over $\{a, b\}$ which have equal numbers of each symbol. Give a CFG for L.

11. The grammar below is ambiguous, but there is an equivalent unambigous CFG.
$S \to aSbS$
$S \to bSaS$
$S \to \lambda$

12. Design a DPDA which accepts the language described in Problem 10.



13. True or False. If the question is currently open, write "O" or "Open."

(i) **F** Every subset of a regular language is regular.

Every language is a subset of some regular language.

(ii) **O** $\mathcal{P} = \mathcal{NP}$.

(iii) **T** The complement of any $\mathcal{P}$–TIME language is $\mathcal{P}$-TIME.
By the complementation theorem.

(iv) **O** The complement of any $\mathcal{NP}$ language is $\mathcal{NP}$.
An important open problem.

(v) **T** The complement of any $\mathcal{P}$–SPACE language is $\mathcal{P}$–SPACE.
By the complementation theroem.

(vi) **T** The complement of every recursive language is recursive.
By the complementation theorem.

(vii) **F** The complement of every recursively enumerable language is recursively enumerable.

The complement of any undecidable $\mathcal{RE}$ language, such as HALT, is co-$\mathcal{RE}$, but not $\mathcal{RE}$.

(viii) **T** If a language L is accepted by an NFA M with p states, then L has pumping length p.

If $w \in L$, of length at least p, any computation of M of length at least p must have a cycle of length between 1 and p. A substring of w starting and ending with visits to the same state is a pumpable substring.

(ix) **T** Given any unambiguous context-free grammar G and any string $w \in L(G)$, there is always a unique leftmost derivation of w using G.

One of the definitions of ambiguity is that some string has more than one leftmost derivation.

(x) **F** For any deterministic finite automaton, there is always a unique minimal non-deterministic finite automaton equivalent to it.

Trick question! It's the converse which is true.

(xi) Duplicate question.

(xii) Duplicate question.

(xiii) **T** Let π be the ratio of the circumference of a circle to its diameter. Then π is recursive.

There are known infinite series that converge to π. Given such a serious and a number n, there is a program which can compute the n^{th} decimal digit of π, and there is a program that runs forever, printing the decimal expansion. However, that program must have unbounded memory.

(xiv) **T** The Kleene closure of any recursive language is recursive.

Let $L \subseteq \Sigma^*$ be recursive, let M be a machine which decides L. and let $w \in \Sigma^*$ of length n. We can use M, to decide which of the $O(n^2)$ substrings of w are members of L, and then we can quickly determine whether w is the concatenation of some of those substrings.

(xv) **T** If $\mathcal{P} = \mathcal{NP}$, then all one-way encoding systems are breakable in polynomial time.

If $\mathcal{P} = \mathcal{NP}$, there is no one-way function.

(xvi) **T** A language L is in $\mathcal{NP}$ if and only if there is a polynomial time reduction of L to SAT.

Yes, by the definition of $\mathcal{NP}$, since SAT is $\mathcal{NP}$–complete.

(xvii) **T** The intersection of any context-free language with any regular language is context-free.

I never explained this, but it's not hard to understand how to attach a DFA which accepts a regular language to the finite control of a PDA which accepts a CFL.

(xviii) **F** Let L be the set of all strings of the form $\langle G_1 \rangle \langle G_2 \rangle$ where G_1 and G_2 are equivalent context-free grammars. Then L is recursively enumerable.

But it is co-$\mathcal{RE}$.

(xix) **T** If L_1 reduces to L_2 in polynomial time, and if L_2 is $\mathcal{NP}$, and if L_1 is $\mathcal{NP}$-complete, then L_2 must be $\mathcal{NP}$-complete.

This is the most common way that new $\mathcal{NP}$–complete languages are found.

(xx) **O** The question of whether two regular expressions are equivalent is $\mathcal{NP}$-complete. (Do not guess. Look it up.)

Equivalence of regular expressions is $\mathcal{P}$–SPACE complete. It is unknown whether $\mathcal{P}$–SPACE $= \mathcal{NP}$.

(xxi) Duplicate question.

(xxii) **T** If L_1 reduces to L_2 in polynomial time, and if L_2 is $\mathcal{NP}$, then L_1 must be $\mathcal{NP}$.

(xxiii) **T** Every language which is accepted by some non-deterministic machine is accepted by some deterministic machine.

But it might take exponentially longer to accept a string.

(xxiv) **F** The language of all regular expressions over $\{a, b\}$ is a regular language.

Trick question. No language which has nested parentheses can be regular. The language is context-free, however.

(xxv) **F** The equivalence problem for C++ programs is recursive.

Machine equivalence is undecidable.

(xxvi) **F** Every function that can be mathematically defined is recursive.

This is somewhat hard to explain. An example is the busy beaver function.

(xxvii) **T** A language is L is $\mathcal{NP}$ if and only if there is a polynomial time reduction of L to Boolean satisfiability.

A basic property of $\mathcal{NP}$–complete languages, in fact, one of the definitions, and SAT is $\mathcal{NP}$–complete.

(xxviii) **T** If there is a recursive reduction of the halting problem to a language L, then L is undecidable.

If L_1 can be reduced to L_2 by an "easy" function then L_2 cannot be "harder" than L_1

(xxix) **F** If there is a recursive reduction of a language L to the halting problem, then L is undecidable.

The converse of the previous problem, but an easy language can be easily reduced to a hard language.

(xxx) **T** The set of rational numbers is countable.

Every fraction can be thought of as the ordered pair (p,q) where p and q are integers. Since there are countably many integers, there are countably many such pairs.

(xxxi) **F** The set of real numbers is countable.

HERE Cantor's diagonalization proof shows that there are uncountably many real numbers.

(xxxii) **T** The set of recursive real numbers is countable.

The decimal expansion of every recursive real number is written by some Pascal program, and there are only countably many Pascal programs.

(xxxiii) **F** There are countably many binary functions.

Let Σ^* be the set of all binary strings, which is countably infinite. The set of binary functions is then $(\Sigma^*)^{\Sigma^*}$, which is uncountable, by Cantor's diagonalization proof.

(xxxiv) **T** There are countably many recursive binary functions.

Each has to be computed by a Pascal program, and there are countably many Pascal programs.

14. Give a context-free language whose complement is not context-free.

Let $L =$ the complement of the language $\{a^n b^n c^n : n \geq 0\}$. L is context free, but its complement is not.

15. State the pumping lemma for regular languages.

For any regular langage L
There exists an integer p
Such that for any string $w \in L$ of length at least p
There exist string x, y, and z
Such that the following statements hold
1. $w = xyz$
2. $|xy| \leq p$ 3. $y \neq \lambda$ (or $|y| \geq 1$)
4. For any integer $i \geq 0$, $xy^i z \in L$.

16. State the Church-Turing thesis. Why is it important?

Every computation by any machine can be emulated by some Turing machine. This is important because Turing machines are simple, making it easier to prove that a given computation cannot be done by a Turing machine, hence not by any machine.

17. For a given instance of an $\mathcal{NP}$ problem, a *witness*, a *certificate*, and a *guide string* all have the same purpose. Give the verification definition of an $\mathcal{NP}$ language.

Let L be any $\mathcal{NP}$ language. There exists a machine V (the verifier) and there exists an integer k such that

(a) For any $w \in L$ of length n there is a string c of length $O(n^k)$, such that V accepts the string w, c in $O(n^k$ time

(b) If $w \notin L$ and c is any string, V does not accept w, c.

The string c could be called either a "certificate" (certifying that $w \in L$) or a "witness."

Alternatively, if L is accepted by a non-deterministic machine M is polynomial time, then a *guide string* for $w \in L$ is a string of instructions which tells M, with input w, what to do at each step where there is a choice, in order to reach a final state.

In all three cases, we mean a string of polynomial length which allows a machine to verify that $w \in L$. For example, for some Boolean expression, a witness would be an assignment of the variables which satisfies the expression.

18. Give a polynomial time reduction of the subset sum problem to the partition problem.

    If $X = (x_1, x_2, \ldots x_n, K)$ is an instance of the subset sum problem, let $S = \sum_{i=1}^{n}$. Then $Y = (x_1, x_2, \ldots x_n, S+1, = (x_1, x_2, \ldots x_n, S+1, K-S+1)$ is an instance of the partition problem which has a solution if and only if X has a solution.

19. Prove that every decidable language is can be enumerated by some machine in canonical order.

    Let $w_1, w_2, \ldots$ be the canonical enumeration of Σ^*. The following program enumerates a deciable language $L \subseteq \Sigma^*$ in canonical order:

    For each i from 1 to ∞
      If($w_i \in L$)
    write w_i

    It is important that L be decidable, else the program might never get past the if statement, being unable to decide the condition.

20. Prove that every language which can be enumerated in canonical order by some machine is recursive.

    If L is finite, we are done, since a finite language is recursive.

    Otherwise, suppose some machine enumerates L in canonical order: $w_1, w_2, \ldots$. The following program decides L.

    Read w.
    For i = 1 to ∞
      If($w = w_i$)
        Write "Yes" and halt
      else if ($w < w_i$) (in canonical order)
        Write "No" and halt

21. Prove that every language which can be enumerated by any machine is recognized by some machine.

    Suppose some machine enumerates L as $w_1, w_2, \ldots$, not necessarily in canonical order. The following program accepts L.

    Read w
    For $i = 1$ to ∞
      If($w = w_i$)
        Write "Yes" and halt.

22. True or False. T = true, F = false, and O = open, meaning that the answer is not known science at this time.

(i) **T** The context-free grammar equivalence problem is co-$\mathcal{RE}$.

This follows immediately from the answer to the next question.

(ii) **T** Let $L = \{(G_1, G_2)\}$: G_1 and G_2 are not equivalent. Then L is recursively enumerable.

This is actually quite easy. Let $L_1 = L(G_1)$ and $L_2 = L(G_2)$. Let Σ_1 and Σ_2 be the terminals of G_1 and G_2, respectively. Let $w_1, w_2, \ldots$ be an enumeration of $(\Sigma_1 \cap (\Sigma_2)^*$.
The following program recognizes L.
For($i = 1 \ldots \infty$)
  If $w_i \in L_1$ and $w_i \notin L_2$ or $w_i \in L_2$ and $w_i \notin L_1$
    Accept and Halt.

(iii) **T** The factoring problem for unary numerals is $\mathcal{P}$–TIME

Yes, because the numeral $\langle n \rangle$ has n bits, and $O(n)$ smaller numerals to check for being divisors of n.

(iv) Duplicate question.

(v) **F** If L is a recursively enumerable language, there must be a machine which enumerates L in canonical order.

That is only true for recursive (decidable) languages.

(vi) **F** The set of all positive real numbers is countable.

(vii) **T** Duplicate question.

(viii) **T** If L is a context-free language over the unary alphabet, then L must be regular.

I have not given you a proof of this.

(ix) **F** The union of any two undecidable languages is undecidable.

Let $L_1 \subseteq \Sigma^*$ be undeciable. and let L_2 be the complement of L_1. Then L_2 is undecidable, but $L_1 \cup L_2 = \Sigma^*$, which is decidable.

(x) **T** co-$\mathcal{P}$–TIME = $\mathcal{P}$–TIME

By the complementation theorem.

23. Give a definition of a recursive real number. (There is more than one correct definition.)

Here are some of the definitions.

(a) $x \in \mathbb{R}$ is recursive means that there is a machine that writes the decimal expansion of x.
(b) $x \in \mathbb{R}$ is recursive means that the function D, where $D(n)$ is the n^{th} digit of the decimal expansion of x, is recursive.
(c) $x \in \mathbb{R}$ is recursive means that, for any fracton y, the question of whether $x < y$ is decidable.

24. Which of these languages (problems) are **known** to be $\mathcal{NP}$-complete? If a language, or problem, is known to be $\mathcal{NP}$-complete, fill in the first circle. If it is either known not be be $\mathcal{NP}$-complete, or if whether it is $\mathcal{NP}$-complete is not known at this time, fill in the second circle.

⊗	○	Boolean satisfiability.
○	⊗	2–SAT.
⊗	○	3–SAT.
⊗	○	Subset sum problem.
⊗	○	Traveling salesman problem.
⊗	○	Dominating set problem.
○	⊗	C++ program equivalence.
⊗	○	Partition.
○	⊗	Regular language membership problem.
⊗	○	Block sorting.

25. State the pumping lemma for context-free languages.

For any context-free language L,
there exists an integer p,‘
such that for any $w \in L$ of length at least p,
there exist strings u, v, x, y, z such that the following statements hold:

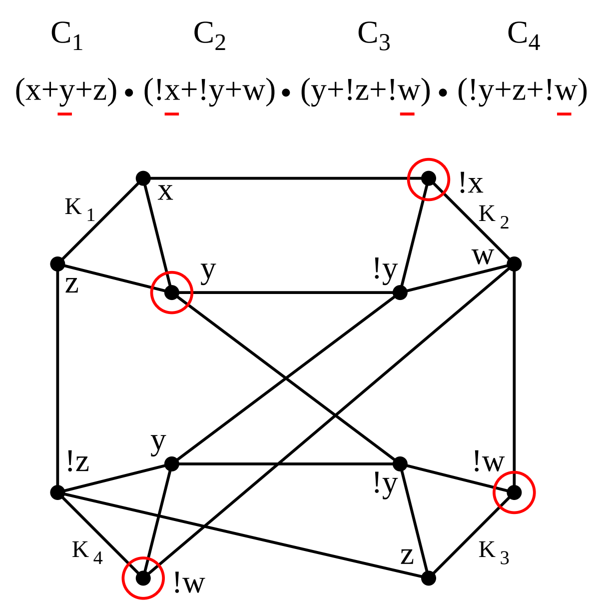
1. $uvxyz = w$
2. $|vxy| \leq p$
3. $|v| + |y| > 0$
4. for any integer $i \geq 0$, $uv^ixy^iz \in L$.

26. Give a polynomial time reduction of 3SAT to the independent set problem.

Let $E = C_1 * C_2 * \cdots C_k$ be Boolean expression in 3–CNF form For any i, let $C_i = t_{i,1} + t_{i,2} + t_{i,3}$ where each $t_{i,p}$ is either a variable or the negation of a variagle. Let G be the graph with $3k$ vertices $\{v_{i,j}\}$ each labeled with one term of E. Let there be an edge from $v_{i,p}$ to $v_{j,q}$ if either $i = j$ or $t_{i,p} * t_{j,q}$ is a contradiction. Then E is satisfiable if and only if G has an independent set of order k.

We illustrate an example, where $E = (x + y + z) * (!x+!y + w) * (y+!z+!w) * (!y + z+!w)$.

The graph G, with $3k = 12$ vertices, is shown, where vertices of a 4–independent set are circled in red. The corresponding term in each clause is underlined in red.



The corresponding assignment of the variables of E is

(a) $x =$ false

(b) $y =$ true

(c) $w =$ false

The remaining variable z can be assigned either true or false. We default to false.