University of Nevada, Las Vegas Computer Science 456/656 Spring 2021

Answers to Practice Problems for the Examination on April 12, 2023

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

   (i) **T** The language of all binary strings which are the binary numerals for multiples of 23 is regular.

   (ii) **F** If L is an $\mathcal{RE}$ (recursively enumerable) language and $w \notin L$, there must be proof that $w \notin L$.

   (iii) **T** If L is a co-$\mathcal{RE}$ language and $w \notin L$, there must be proof that $w \notin L$.

   (iv) **O** If L is a $\mathcal{P}$–SPACE language and $w \in L$, there must be a proof of polynomial length that $w \in L$.

   If $\mathcal{P}$–SPACE $= \mathcal{NP}$, then it's true.

   (v) **T** If L is any $\mathcal{P}$–TIME language, there is a reduction of L to the Boolean circuit problem, and this reduction can be calculated in polylogarithmic time with polynomially many processors.

   (vi) **T** Let $L = \{\langle G_1\rangle\langle G_2\rangle : G_1$ is not equivalent to $G_2\}$ Then L is recursively enumerable.

   Context-free grammar equivalence is co-$\mathcal{RE}$.

   (vii) **T** The complement of any $\mathcal{P}$-SPACE language is $\mathcal{P}$-SPACE.

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

   (ix) **T** The complement of every recursive language is recursive.

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

   (xi) **T** Every language which is generated by a general grammar is recursively enumerable.

   (xii) **F** The context-free language membership problem is undecidable.

   The CYK algorithm decides that problem.

   (xiii) **T** The factoring problem, where inputs are written in binary notation, is co-$\mathcal{NP}$.

   I don't expect you to know the proof.

   (xiv) **T** The factoring problem, where inputs are written in unary (caveman) notation, is $\mathcal{P}$–TIME.

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

   That does back to the first part of the course.

   (xvi) **F** The question of whether two regular expressions are equivalent is known to be $\mathcal{NP}$-complete.

   (xvii) **T** The halting problem is recursively enumerable.

   (xviii) **F** The intersection of any two context-free languages is context-free.

   (xix) **F** The question of whether a given Turing Machine halts with empty input is decidable.

(xx) **T** The class of languages accepted by NTM's (non-deterministic Turing machines) is the same as the class of languages accepted by Turing machines.

(xxi) **F** The class of languages accepted by non-deterministic push-down automata is the same as the class of languages accepted by deterministic push-down automata.

(xxii) **T** Let π be the ratio of the circumference of a circle to its diameter. The problem of whether the n^{th} digit of the decimal expansion of π for a given n is equal to a given digit d is decidable.

(xxiii) **F** An undecidable language is necessarily $\mathcal{NP}$-complete.

That was a trick question. All $\mathcal{NP}$ languages are decidable

(xxiv) ________ Every context-free language is in the class $\mathcal{P}$-TIME.

By the CYK algorithm.

(xxv) **T** Every regular language is in the class $\mathcal{NC}$

(xxvi) **T** Let $L = \{a^i b^j c^k : i = j \text{or} j = k\}$. Then L is not generated by any unambiguous context-free grammar.

(xxvii) **F** Every context-free grammar can be parsed by some deterministic top-down parser.

(xxviii) **T** Every context-free grammar can be parsed by some non-deterministic top-down parser.

(xxix) **F** Commercially available parsers do not use the LALR technique, since most modern programming languages are not context-free.

(xxx) **F** The boolean satisfiability problem is undecidable.

(xxxi) **F** If anyone ever proves that the binary integer factorization problem is in $\mathcal{P}$–TIME, then all public key/private key encryption systems will be known to be insecure.

RSA encryption will be known to be insecure, but there could be other encryption systems that would be unaffected.

(xxxii) **T** If anyone ever proves that $\mathcal{P} = \mathcal{NP}$, then all public key/private key encryption systems will be known to be insecure.

(xxxiii) **T** If a string w is generated by a context-free grammer G, then w has a unique leftmost derivation if and only if it has a unique rightmost derivation.

If and only if G is unambiguous.

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

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

Did anyone fall for this, after I've mentioned it many time?

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

I never proved this in class, but the proof is not terribly difficult.

(xxxvii) **T** Every language which is generated by a general grammar is recursively enumerable.

(xxxviii) **T** There exists some mathematical statement which is true but which has no proof.

(xxxix) **F** The set of all binary numerals for prime numbers is in the class $\mathcal{P}$.

(xl) **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.

The usual method for finding new $\mathcal{NP}$-complete probliems

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

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

The other way around!

(xliii) ________ No language which has an ambiguous context-free grammar can be accepted by a DPDA.

We've done example of ambiguous grammars parsed by the LALR technique.

(xliv) **F** The class of languages accepted by non-deterministic push-down automata is the same as the class of languages accepted by deterministic push-down automata.

Oops! Repeated question.

(xlv) **T** Let $F(0) = 1$, and let $F(n) = 2^{F(n-1)}$ for $n > 0$. Then F is recursive.

(xlvi) **F** The "Busy beaver" function is recursive.

It's on the internet.

(xlvii) ________ Let π be the ratio of the circumference of a circle to its diameter. (That's the usual meaning of π you learned in school.) The problem of whether the n^{th} digit of π, for a given n, is equal to a given digit d is decidable.

You can write a program that prints the n^{th} digit of π for any n.

(xlviii) **T** There is a machine that parses Pascal. (A parser for a computer language is a machine that constructs a correct parse tree for every valid program written in that language.)

(xlix) **F** There is a machine that parses C++. Hint: look this up on the internet.

This is recently discovered flaw in C++.

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

The busy beaver function.

(li) **T** Every context-free language is in the class $\mathcal{P}$-TIME.

Repeat question! Use CYK.

(lii) **T** The Post correspondence problem is undecidable.

You had to look that up. I do not expect you to understand the Post correspondence problem.

For the next three problems, recall that a fraction is a string consisting of a numeral, follwed by a slash, followed by another numeral. Thus, any set of fractions is a language. If x is any real number, let $L_L(x)$ be the set of all fractions whose values are less than x, and let $L_R(x)$ be the set of all fractions whose values are greater than x.

(liii) **F** If a sequence of fractions converges to a real number x, then x must be a recursive real number.

If so, every real number would be recursive.

(liv) **T** If $L_L(x)$ is recursive, then then x must be a recursive real number.

$L_R(x)$ is the complement of $L_L(x)$, and is hence recursive. We can then construct an increasing sequence converging to x and also a decreasing sequence converging to x. This gives us decimal approximations to x of arbitrary accuracy, hence we can always find the n^{th} digit.

(lv) **F** If $L_L(x)$ is recursively enumerable, then then x must be a recursive real number.

This is really subtle. I'll try to explain it later.

(lvi) **T** If $L_L(x)$ and $L_R(x)$ are both recursively enumerable, then x must be a recursive real number.

Since $L_R(x)$ is the complement of $L_L(x)$, it is co-$\mathcal{RE}$. But it is also $\mathcal{RE}$, which means it is recursive, hence its complement $L_L(x)$ is also recursive. We have reduced the problem to (liv), hence x is recursive.

2. State a problem, or language, that is known to be in the class $\mathcal{NP}$, is not known to be $\mathcal{P}$–TIME, and is not known to be $\mathcal{NP}$–complete.

The binary numeral factorization problem.

3. Determine whether the following 2CNF Boolean expression is satisfiable. If so, give a satisfying assignment.

$(!d+g)*(!h+!d)*(f+e)*(e+!e)*(b+!j)*(!e+j)*(!i+c)*(a+d)*(g+!j)*(e+!c)*(!j+f)$
$(b+i)*(d+!j)*(!h+!c)*(f+g)*(h+!c)*(!b+!j)*(!g+!j)*(a+c)*(!i+g)$

$(!d+g)*(!h+!d)*(f+e)*(b+!j)*(!e+j)*(!i+c)*(a+d)*(g+!j)*(e+!c)*(!j+f)$
$(b+i)*(d+!j)*(!h+!c)*(f+g)*(h+!c)*(!b+!j)*(!g+!j)*(a+c)*(!i+g)$

$a=1$ $f=1$

$(!d+g)*(!h+!d)*(b+!j)*(!e+j)*(!i+c)*(g+!j)*(e+!c)*$
$(b+i)*(d+!j)*(!h+!c)*(h+!c)*(!b+!j)*(!g+!j)*(!i+g)$

Cycle in G: $b \to !j \to !e \to !c \to !i \to b$. Thus $b = !j = !e = !c = !i$

$(!d+g)*(!h+!d)*(b+b)*(b+!b)*(b+!b)*(g+b)*(!b+b)*$
$(b+i)*(d+b)*(!h+b)*(h+b)*(!b+b)*(!g+b)*(b+g)$

$(!d+g)*(!h+!d)*(b)*(g+b)*(d+b)*(!h+b)*(h+b)*(!g+b)*(b+g)$

$(!d+g)*(!h+!d)*(1)*(g+1)*(d+1)*(!h+1)*(h+1)*(!g+1)*(1+g)$

$b=1$, hence $j=e=c=i=0$

$(!d+g)*(!h+!d)$

$d=0$

$(1+g)*(!h+1)$

λ: satisfiable.

Satisfying assignment: $a=1, b=1, c=0, d=0, e=0, f=1, i=0, j=0,$ g, h any.

4. Prove that the halting problem is undecidable. Do it the way you should, not by quoting a theorem in a handout.

Assume the halting problem is decidable. Let D be a machine which executes the following program.

Read a machine description $\langle M \rangle$.
If (M halts with input $\langle M \rangle$)
    Loop forever
Else halt.

Since the halting problem is decidable, the program will not get stuck evaluating the condition of the "if" statement.

Now run D with input $\langle D \rangle$.

If D accepts $\langle D \rangle$, then D will not accept $\langle D \rangle$ since it will loop forever. If D does not accept $\langle D \rangle$, then D halts, and hence accepts $\langle D \rangle$. We now have a contradiction, hence the machine D cannot exist, hence the halting problem is undecidable.