COT 6410 Spring 2020 Final Exam Name:__________________

- The Chomsky hierarchy classifies languages (sets over finite alphabets) into four types (Regular, Context-Free, Context-Sensitive and Phrase-Structured). Associated with each is a grammar type that generates languages of this type and a machine class that accepts languages of this type. The grammars have names that correspond to the language type, e.g., Context-Free Grammars. The machine hierarchy, corresponding to the language types, is Finite State, Pushdown, Linear Bounded and Turing Machine. All but Pushdown Automata have the same capability whether deterministic or non-deterministic. Only Finite State Automata have the property that they can be algorithmically reduced to a minimal form (in this case, minimal state). Undecidability problems abound for all but Regular Languages. However, all but Phrase-Structured languages have associated algorithms to determine membership.
- The notation $\mathbf{z = \langle x,y \rangle}$ denotes the pairing function with inverses $\mathbf{x = \langle z \rangle_1}$ and $\mathbf{y = \langle z \rangle_2}$.
- The minimization notation $\boldsymbol{\mu}$ **y [P(…,y)]** means the least **y** (starting at **0**) such that **P(…,y)** is true. The bounded minimization (acceptable in primitive recursive functions) notation $\boldsymbol{\mu}$ **y (u≤y≤v) [P(…,y)]** means the least **y** (starting at **u** and ending at **v**) such that **P(…,y)** is true. I define $\boldsymbol{\mu}$ **y (u≤y≤v) [P(…,y)]** to be **v+1**, when no **y** satisfies this bounded minimization.
- The tilde symbol, **~**, means the complement. Thus, set **~S** is the set complement of set **S**, and the predicate **~P(x)** is the logical complement of predicate **P(x).**
- A function **P** is a predicate if it is a logical function that returns either **1** (**true**) or **0** (**false**). Thus, **P(x)** means **P** evaluates to **true** on **x**, but we can also take advantage of the fact that **true** is **1** and **false** is **0** in formulas like **y × P(x)**, which would evaluate to either **y** (if **P(x)**) or **0** (if **~P(x)**).
- A set **S** is recursive if **S** has a total recursive characteristic function χ_S, such that $\mathbf{x \in S \Leftrightarrow \chi_S(x)}$. Note χ_S is a total predicate. Thus, it evaluates to **0** (**false**), if $\mathbf{x \notin S}$.
- When I say a set **S** is **Recursively Enumerable** (**RE**), unless I explicitly say otherwise, you may assume any of the following equivalent characterizations:
  1. **S** is either empty or the range of a total recursive function $\mathbf{f_S}$.
  2. **S** is the domain of a partial recursive function $\mathbf{g_S}$.
- If I say a function **g** is partially computable, then there is an index **g** (we tend to overload the index as the function name), such that $\mathbf{\Phi_g(x) = \Phi(x, g) = g(x)}$. Here $\mathbf{\Phi}$ is a universal partially recursive function.
  Moreover, there is a primitive recursive function **STP**, such that
  STP(g, x, t) is **1** (true), just in case **g**, started on **x**, halts in **t** or fewer steps.
  STP(g, x, t) is **0** (false), otherwise.
  Finally, there is another primitive recursive function **VALUE**, such that
  VALUE(g, x, t) is **g(x)**, whenever **STP(g, x, t)**.
  VALUE(g, x, t) is defined but meaningless if **~STP(g, x, t)**.
- The notation $\mathbf{f(x)\downarrow}$ means that **f** converges when computing with input **x** ($\mathbf{x \in Dom(f)}$). The notation $\mathbf{f(x)\uparrow}$ means **f** diverges when computing with input **x** ($\mathbf{x \notin Dom(f)}$).
- When I ask you to show one set of indices, **A**, is many-one reducible to another, **B**, denoted **A ≤m B,** you must demonstrate a total computable function **f**, such that $\mathbf{x \in A \Leftrightarrow f(x) \in B}$. The stronger relationship is that **A** and **B** are many-one equivalent, **A ≡m B**, requires that you show **A ≤m B** and **B ≤m A**. The related notion of one-one reducibility and equivalence require that the reducing function, **f** above, be 1-1. The notation just replaces the **m** with a **1**, as in **A ≤1 B**. We can also replace the **m** or **1** with a **t**, as in **A ≤t B**, to indicate the notion of Turing reducibility. When we say **A ≤t B**, we mean there is some computable algorithm that uses an Oracle for **B** to solve **A**.

- The **Halting Problem** for any effective computational system is the problem to determine of an arbitrary effective procedure **f** and input **x**, whether or not **f(x)**$\downarrow$. The set of all such pairs, $\mathbf{K_0}$, is a classic re non-recursive set. $\mathbf{K_0}$ is also known as $\mathbf{L_u}$, the universal language. The related set, **K**, is the set of all effective procedures **f** such that **f(f)**$\downarrow$ or more precisely $\mathbf{\Phi_f(f)}$. **K** and $\mathbf{K_0}$ are classic **RE-Complete** sets, meaning that every **RE** set many-one reduces to these hardest **RE** sets.
- The **Uniform Halting Problem** is the problem to determine of an arbitrary effective procedure **f**, whether or not **f** is an algorithm (halts on all input). This set, **TOTAL**, is a classic **non-RE**, **non-Co-RE** set. It is also called **RE-Hard** in our terminology.
- In the computability domain, we usually categorize problems as **Recursive**, **Recursively Enumerable (RE)**, **Co-Recursively Enumerable (co-RE)**, **RE-Complete**, **Co-RE-Complete**, and **RE-Hard** (my own term to describe a set for which we can show a Turing reduction from some **RE-Complete**, e.g., **TOTAL** is **RE-Hard** since $\mathbf{K \le_t TOTAL}$).
- When I ask for a reduction of one set of indices to another, the formal rule is that you must produce a function that takes an index of one function and produces the index of another having whatever property you require. However, I allow some laxness here. You can start with a function, given its index, and produce another function, knowing it will have a computable index. For example, given **f**, a unary function, I might define $\mathbf{G_f}$, another unary function, by
  $\mathbf{G_f(0) = f(0);\ G_f(y+1) = G_f(y) + f(y+1)}$
  This would get $\mathbf{G_f(x)}$ as the sum of the values of **f(0)+f(1)+…+f(x)**.
- The **Post Correspondence Problem** (**PCP**) is known to be undecidable. This problem is characterized by instances that are described by a number **n>0** and two **n**-ary sequences of non-empty words $\mathbf{<x_1,x_2,\ldots,x_n>}$, $\mathbf{<y_1,y_2,\ldots,y_n>}$. The question is whether or not there exists a sequence, $\mathbf{i_1,i_2,\ldots,i_k}$, such that $\mathbf{1 \le i_j \le n}$, $\mathbf{1 \le j \le k}$, and $\mathbf{x_{i_1}x_{i_2}\cdots x_{i_k} = y_{i_1}y_{i_2}\cdots y_{i_k}}$
- The related notion of polynomial reducibility and equivalence require that the reducing function, **f** above, be computable in polynomial time in the size of the instance of the element being checked. The notation just replaces the **m** with a **p**, as in $\mathbf{A \le_p B}$ and $\mathbf{A \equiv_p B}$.
- A decision problem **R** is in **NP** if it can be solved by a non-deterministic Turing machine in polynomial time. Alternatively, **Q** is in **NP** if a proposed proof of any instance having answer yes can be verified by a deterministic Turing machine in polynomial time. The set **Co-NP** contains the complements of all problems in **NP**.
- A decision problem **R** is **NP-Complete** if and only if it is in **NP** and, for any problem **Q** in **NP**, it is the case that $\mathbf{Q \le_p R}$. A decision problem **R** is **Co-NP-Complete** if and only if it is in **Co-NP** and, for any problem **Q** in **Co-NP**, it is the case that $\mathbf{Q \le_p R}$.
- A function problem (typically optimization problem) **F** is **NP-Hard** if and only if there is an **NP-Complete** problem **Q** that is polynomial time Turing-reducible ($\mathbf{\le_{tp}}$) to **F**. By saying $\mathbf{Q \le_{tp} F}$, we mean that **Q** can be solved in polynomial time so long as it has an oracle for **F**. We often limit our domain of consideration to decision problems when talking of **NP-Hard**, but the concept also applies to function problems, especially to optimizations of problems in **NP-Complete**.
- A function problem **F** is **NP-Easy** if and only if it is polynomial time Turing-reducible ($\mathbf{\le_{tp}}$) to some **NP** problem **Q**. By saying $\mathbf{F \le_{tp} Q}$, we mean that **F** can be solved in polynomial time so long as it has an oracle for **Q**.
- A function problem **F** is **NP-Equivalent** if and only if it is both **NP-Hard** and **NP-Easy**.

COT 6410 Spring2020 Sample Final Exam

1. Let set **A** be **infinite recursive**, **B** be **re non-recursive** and **C** be **non-re**. Using the terminology **(REC) recursive**, **(RE) re, non-recursive**, **(NR) non-re (possibly co-re)**, categorize each set by dealing with the cases I present, saying whether or not the set can be of the given category and briefly, but convincingly, justifying each answer (BE COMPLETE). You may assume sets like $\aleph$ are infinite REC; **K** and $\mathbf{K_0}$ are RE; and **TOTAL** is non-re. You may also assume, for any set **S**, the existence of comparably hard sets
$S_E = \{2x|x\in S\}$ and $S_D = \{2x+1|x\in S\}$.

   a.) $\mathbf{A + B = \{ x \mid x = y + z}$**, for some** $\mathbf{y \in A}$ **and some** $\mathbf{z \in B \}}$

   REC:

   b.) $\mathbf{A \cap C = \{ x \mid x \in A}$ **and** $\mathbf{x \in C}$ **and** $\mathbf{x \notin A \}}$

   RE:

2. Choosing from among **(REC) recursive**, **(RE) re non-recursive, (coRE) co-re non-recursive**, **(NRNC) non-re/non-co-re**, categorize each of the sets in a) through d). Justify your answer by showing some minimal quantification of some known recursive predicate.

   a) $\mathbf{A = \{ <f,g> \mid \exists x\ \varphi_f(x)\downarrow}$ **and** $\mathbf{\varphi_g(x) = \varphi_f(x) \}}$.

   ______________________________ ______

   b.) $\mathbf{B = \{ f \mid}$ **range**$\mathbf{(\varphi_f)}$ **is empty }**

   ______________________________ ______

   c.) $\mathbf{C = \{ <f,x> \mid \varphi_f(x)\downarrow}$ **but takes at least 10 steps to do so }**

   ______________________________ ______

   d.) $\mathbf{D = \{ f \mid \varphi_f}$ **diverges for some value of x }**

   ______________________________ ______

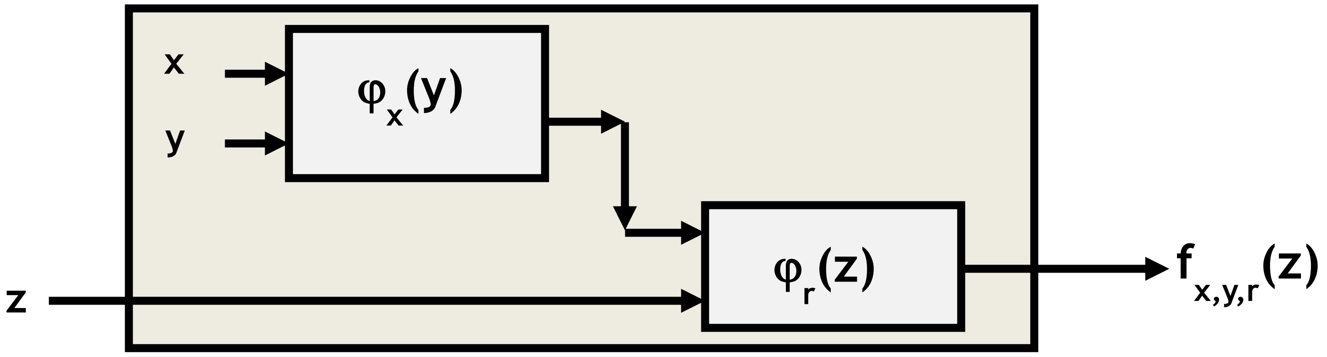
3. Looking back at Question 1, which of these are candidates for using Rice's Theorem to show their unsolvability? Check all for which Rice Theorem might apply.

   a) ____ **b)** ____ **c)** ____ **d)** ____

4. Let **S** be an arbitrary semi-decidable set. By definition, **S** is the domain of some partial recursive function $\mathbf{g_S}$. Using $\mathbf{g_S}$, constructively show that **S** is the range of some partial recursive function, $\mathbf{f_S}$. No proof is required; just the construction is needed here.

5. Using the definition that **S** is recursively enumerable iff **S** is the range of some effective procedure $\mathbf{f_S}$ (partial recursive function), prove that if both **S** and its complement **~S** are recursively enumerable (using enumerating effective procedures $\mathbf{f_S}$ and $\mathbf{f_{\sim S}}$) then **S** is decidable. To get full credit, you must show the characteristic function for **S**, $\boldsymbol{\chi_S}$, in all cases. Also, be sure to discuss why your $\boldsymbol{\chi_S}$ works.

6. Rice's Theorem deals with attributes of certain types of problems **P** about partial recursive functions and their corresponding sets of indices $\mathbf{S_P}$. The following image describing a function $\mathbf{f_{x,y,r}}$ is central to understanding Rice's Theorem.



Explain the meaning of this by indicating:

a.) What assumption do we make about what kind of functions are not in P?

b.) What is **r**, how is it chosen and how can we guarantee its existence?

c.) Using recursive function notations, write down precisely what $\mathbf{f_{x,y,r}}$ computes for the Strong Form of Rice's Theorem.

How does this function $\mathbf{f_{x,y,r}}$ behave with respect to **x**,**y** and **r**, and how does that relate to the original problem, **P**, and set, $\mathbf{S_P}$?

7. Define **NAT = { f | range(f) = $\aleph$ }**. That is, **f $\in$NAT** iff **f**'s range includes every natural number.

a.) Show some minimal quantification of some known recursive predicate that provides an upper bound for the complexity of **NAT**.

b.) Use Rice's Theorem to prove that **NAT** is undecidable.

c.) Show that **TOT $\le_m$ NAT**, where **TOT = { f | $\forall$x $\varphi_f(x)\downarrow$ }**.

8. Why does Rice's Theorem have nothing to say about the following? Explain by showing some condition of Rice's Theorem that is not met by the stated property.
AT-LEAST-LINEAR = { f | $\forall$y $\varphi_f(y)$ converges in no fewer than y steps }.

9. Consider the following set of independent tasks with associated task times:
(T1,4), (T2,5), (T3,2), (T4,7), (T5,1), (T6,4), (T7,8)
Fill in the schedules for these tasks under the associated strategies below.
Greedy using the list order above:

Greedy using a reordering of the list so that longest running tasks appear earliest in the list:

10. We described the proof that **3SAT** is polynomial reducible to **Subset-Sum**. You must repeat that.

a.) Assuming a **3SAT** expression **(a + a + ~b) (~a + b + c)**, fill in all omitted values (zeroes elements can be left as omitted) of the reduction from **3SAT** to **Subset-Sum**.

	a	**b**	**c**	**a + a + ~b**	**~a + b + c**
a					
~a					
b					
~b					
c					
~c					
C1					
C1'					
C2					
C2'					
	1	**1**	**1**	**3**	**3**

b.) List some subset of the numbers above (each associated with a row) that sums to **1 1 1 3 3**. Indicate what the related truth values are for **a**, **b** and **c**.

11. Present a gadget used in the reduction of **3-SAT** to some graph theoretic problem where the gadget guarantees that each variable is assigned either **True** or **False**, but not both. Of course, you must tell me what graph theoretic problem is being shown **NP-Complete** and you must explain why the gadget works.

12. Let **Q** be some problem (an optimization or decision problem). Assuming **≤ p** means many-one reducible in polynomial time and **≤ tp** means Turing-reducible in polynomial time, categorize **Q** as being in one of **P**, **NP**, **co-NP**, **NP-Complete**, **NP-Easy**, **NP-Hard**, or **NP-Equivalent** (see first two pages for definitions of each of these concepts). For each case, choose the most precise category. I filled in one answer already.

Description of Q	**Category**
Q is decidable in deterministic polynomial time	
For some **R** in **NP**, **Q ≤ tp R**	
Q is both **NP-Easy** and **NP-Hard**	
Q is in **NP** and if **R** is in **NP** then **R ≤ p Q**	
A solution to **Q** is verifiable in deterministic polynomial time	
Q's complement is in **NP**	

13. A graph **G** is **k-Colorable** if its vertices can be colored using just **k** (or fewer colors) such that adjacent vertices have different colors. The **Chromatic Number** of a graph **G** is the smallest number **k** for which **G** is **k-Colorable**. **k-Colorable** is a decision problem that has parameters (**G**, **k**), whereas the **Chromatic Number** problem is a function with a single parameter **G**. In all cases, assume **G** has **n** vertices.

a.) Show that **k-Colorable ≤ tp Chromatic Number** (**≤ tp** means Turing reducible in polynomial time).

b.) Show that **Chromatic Number ≤ tp k-Colorable** (**≤ tp** means Turing reducible in polynomial time).

14. **Partition** refers to the decision problem as to whether some set of positive integers **S** can be partitioned into two disjoint subsets whose elements have equal sums. **Subset-Sum** refers to the decision problem as to whether there is a subset of some set of positive integers **S** that precisely sums to some goal number **G**.

a.) Show that **Partition ≤p Subset-Sum**.

b.) Show that **Subset-Sum ≤p Partition**.

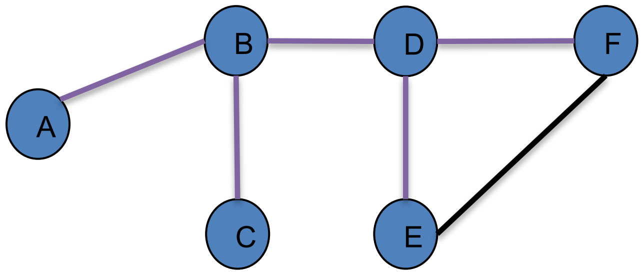
15. **QSAT** is the decision problem to determine if an arbitrary fully quantified Boolean expression is true. Note: **SAT** only uses existential, whereas **QSAT** can have universal qualifiers as well so it includes checking for Tautologies as well as testing Satisfiability. What can you say about the complexity of **QSAT** (is it in **P**, **NP**, **NP-Complete**, **NP-Hard**)? Justify your conclusion.

16. Given the following instance of 2SAT, $E=(a \lor b) \land (\neg a \lor \neg b) \land (\neg a \lor \neg c) \land (a \lor c)$, display the associated implication graph, show its strongly connected components and then show how this leads to an assignment of variables that satisfies E.

17. Specify True (**T**) or False (**F**) for each statement.

Statement	**T or F**
Every Regular Language is also a Context Free Language	
Phrase Structured Languages are the same as RE Languages	
The Context Free Languages are closed under Complement	
A language is recursive iff it and its complement are re	
PCP is undecidable even for one letter systems	
Membership in Context Sensitive Languages is undecidable	
Every RE language is Turing reducible to its complement	
Emptiness is undecidable for Context Sensitive Languages	
The complement of a trace language is Context Free	
The word problem for two-letter Semi-Thue Systems is decidable	

18. Consider the following graph. We wish to show it has vertex cover solution of 3. Our approach is to reduce this to the 2SAT related problem of determine if we can satisfy some associated positive 2SAT expression, S, so that the minimum solution for S involves at most 3 variables being set to true? What is that corresponding positive 2SAT expression S? What is a minimal positive solution for the expression S and the vertex cover solution for the graph with which we started?



19. Let L be an arbitrary CFL. Show that $\mathbf{L = L^2}$ is undecidable by reducing $\mathbf{L = \Sigma^*}$ to $\mathbf{L = L^2}$.

Alternative 1 to #5

5. Using the definition that **S** is a recursively enumerable, non-empty set iff **S** is the range of some algorithm $\mathbf{f_S}$, prove that if both **S** and its complement **~S** are recursively enumerable (using enumerating algorithms $\mathbf{f_S}$ and $\mathbf{f_{\sim S}}$) then **S** is decidable. To get full credit, you must show the characteristic function for **S**, $\boldsymbol{\chi}_\mathbf{S}$, in all cases. Also, be sure to discuss why your $\boldsymbol{\chi}_\mathbf{S}$ works

Alternative 2 to #5

5. Using the definition that **S** is a recursively enumerable, non-empty set iff **S** is the domain of some effective procedure $\mathbf{f_S}$, prove that if both **S** and its complement **~S** are recursively enumerable (using the domains of procedures $\mathbf{f_S}$ and $\mathbf{f_{\sim S}}$) then **S** is decidable. To get full credit, you must show the characteristic function for **S**, $\boldsymbol{\chi}_\mathbf{S}$, in all cases. Also, be sure to discuss why your $\boldsymbol{\chi}_\mathbf{S}$ works