COT 6410 Spring 2019 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 μ **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 μ **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 μ **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 Φ 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 **f(x)**$\downarrow$ means that **f** converges when computing with input **x** (**x ∈ Dom(f)**). The notation **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 $\mathbf{A \leq_m B}$, 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, $\mathbf{A \equiv_m B}$, requires that you show $\mathbf{A \leq_m B}$ and $\mathbf{B \leq_m A}$. 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 $\mathbf{A \leq_1 B}$. We can also replace the **m** or **1** with a **t**, as in $\mathbf{A \leq_t B}$, to indicate the notion of Turing reducibility. When we say $\mathbf{A \leq_t B}$, 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)↓**. 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)↓** 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 **K ≤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 **A ≤p B** and **A ≡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 **Q ≤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 **Q ≤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 (**≤tp**) to **F**. By saying **Q ≤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 (**≤tp**) to some **NP** problem **Q**. By saying **F ≤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 Spring 2019 Final Exam Name: _____
Raw Score ___ / 84 Grade:_____ Semester Grade:_____

4 **1.** Let set **A** be a **non-empty Regular Language**, and let **B** be a **non-Regular Context-Free Language**. For each of **(REG) Regular**, and **(CFL) non-Regular Context-Free**, specify if the language **C** can or cannot be of the given language type and prove your assertions. As always, assume **A**, **B** and **C** are over some finite alphabet, **Σ**.

C = B/A = { x | w = xy, where w ∈ B and y ∈ A }

Note: / is called **Quotient** and was extensively discussed in Class.
You may use any well-known Regular and Context-Free Languages. E.g., every language described by a Regular Expression is **Regular** and the set **{ $a^n b^n$ | n>0 }** is a **CFL**.

REG: (**Big Hint: C** can be **Regular** so show **A** and **B** where **B/A** is **Regular**.
Explicitly describe languages **A**, **B** and **C** and you are done).

CFL: (**Big Hint: C** can be a **CFL**, so show **A** and **B** where **B/A** is a **CFL**.
Explicitly describe languages **A**, **B** and **C** and you are done).

6 **2.** Let set **A** be a **non-empty recursive** set, and let **B** be an **RE non-recursive** set. For each of **(REC) recursive**, **(RE) RE non-recursive**, and **(NRE) non-re**, specify if the set **C** can or cannot be of the given set type and prove your assertions. Sets are subsets of the **Natural Numbers**.

C = B // A = { x | x = y // z, where y ∈ B, z ∈ A and y // z is floor(y/z) if z>0; else y // 0 = 0 }

Note: // is called limited division and was shown primitive recursive in Class Notes.
You may assume that **A** has the characteristic function χ_A and also that it is the range of some total recursive function (algorithm) $\mathbf{f_A}$, and **B** is the range of some total recursive function $\mathbf{f_B}$.

REC: (**Big Hint: C** can be **REC** so show **non-empty REC** set **A** and **RE non-recursive** set **B** such that **B // A** is **REC**. Explicitly describe sets **A**, **B** and **C** and you are done).

RE: (**Big Hint: C** can be **RE** so show **non-empty REC** set **A** and **RE non-recursive** set **B** such that **B // A** is **RE**. Explicitly describe sets **A**, **B** and **C** and you are done).

NRE: (**Big Hint: C** must be **RE**. Prove this by showing a total recursive function whose range is **C**).

8 **3.** Let **S** be some **RE** set. Prove that **S** is **infinite recursive** (decidable) if and only if **S** is the range of some **monotonically increasing total recursive function** (algorithm). Hints: To show **S** is infinite recursive, you must use its monotonically increasing enumerating function to show it is infinite and to provide its **characteristic function** χ_S. To show **S** is the range of some monotonically increasing function you must explicitly present that function, $\mathbf{f_S}$, and argue it is monotonically increasing and its range is **S**. Be sure to justify that the functions you create achieve the desired goals.

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

Statement	**T or F**
Membership in Phrase-Structured Languages is semi-decidable	
Membership in Context-Sensitive Languages is unsolvable	
Membership in Context-Free Languages can be solved in polynomial time	
Membership in Regular Languages can be solved in linear time	
The set of programs representing all and only algorithms is a Phrase-Structured Language	
Every recursive set is recognized by some primitive recursive function	
Every re set is enumerated by some primitive recursive function	
PCP over a one-letter alphabet is decidable	
An algorithm exists to determine if a Context-Sensitive Language is finite	
Every problem solvable in linear space is of linear time complexity	
A proposed solution to an instance of Vertex Cover can be checked in linear time	
Petri Net Reachability is at least a doubly exponential Problem	
Every NP-Hard decision problem is NP-Complete	
Every NP-Complete decision problem is NP-Equivalent	

4 **5.** Let $\mathbf{P = \langle\langle x_1,x_2,\ldots,x_n\rangle, \langle y_1,y_2,\ldots,y_n\rangle\rangle}$, $\mathbf{x_i,y_1 \in \Sigma^+}$, $\mathbf{1 \le i \le n}$, be an arbitrary instance of **PCP**. We can use **PCP's undecidability** to show the undecidability of the problem to determine if a **Context Sensitive Grammar** generates a non-empty language. I will start the grammar, **G**. You must complete it so it maps an instance of PCP to the non-emptiness problem for this $\mathcal{L}$**(G)**.

Define **G = ({S, T} ∪ Σ, {*}, S, R)**, where **R** is the set of rules:
$\mathbf{S \rightarrow x_i\ S\ y_i^R \mid x_i\ T\ y_i^R \qquad 1 \le i \le n}$ **(Note: the superscripted R means Reversed)**
// Write the rules for the rest of this CSG.

6. Define **OddsRule(OR) = { f | for all x: f(2x+1) > f(2x) }**.

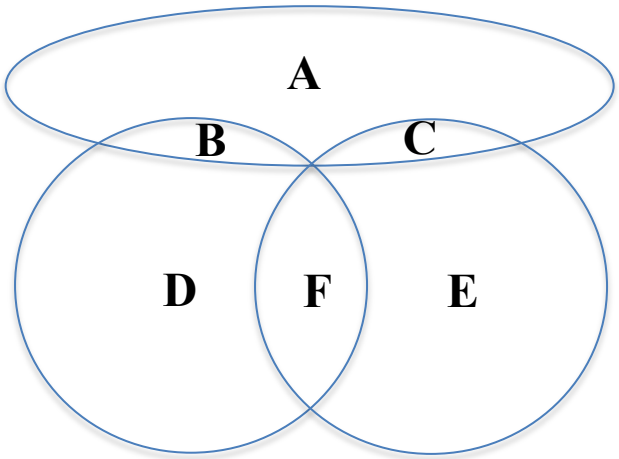
2 **a.)** Show some minimal quantification of some known primitive recursive predicate that provides an upper bound for the complexity of **OR**.

5 **b.)** Use Rice's Theorem to prove that **OR** is undecidable. Be Complete.

4 **c.)** Show that **OR** is many-one reducible to **Total = { f | $\forall$x f(x)↓ }**.

6 **7.** Consider the Venn diagram below. Identify the alphabetically labeled regions below (each representing a set of decision problems) to indicate characterizations as **co-RE**, **co-RE-Complete**, **RE**, **RE-Complete**, **RE-Hard**, and **Recursive**. To answer this, just write in the labels associated with **A**, **B**, **C**, **D**, **E**, and **F**, choosing the most precise label for each case.

A_____ **B**_____ **C**_____ **D**_____ **E**_____ **F**_____



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

Assuming a **3SAT** expression **(~a + ~b + ~c) (a + b + c) (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 + ~b + ~c**	**a + b + c**	**a + ~b + c**
a						
~a						
b						
~b						
c						
~c						
C1						
C1'						
C2						
C2'						
C3						
C3'						
	1	**1**	**1**	**3**	**3**	**3**

9. Consider the decision problem to determine if there is an **Independent Set** of vertices of size **k>0** in some undirected graph **G = (V, E)**. Here we always assume that **k ≤ |V|** and **|V| > 0**, for if not the answer is a resounding **NO**. An independent set, **V'**, is any subset of **V**, such that if **t** and **u** are in **V'** then **(t, u)** is not an edge in **E**. A **Maximum Independent Set** is an independent set of largest possible size for a given graph **G= (V, E)**. This size is called the **Independence Number** of **G**, and denoted **α(G)**. The problem of finding such a set is called the **Maximum Independent Set Problem**.

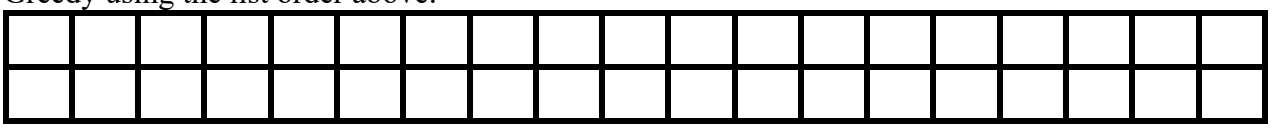
2 **a.)** Show that this decision problem, computing whether or not **G** has an independent set of vertices of size **k**, is Turing reducible in polynomial time, relative to the size of the representation of **G**, to the **Maximum Independent Set Problem**. Hint: **α(G)** is useful here and no proof is required.

5 **b.)** Show that the **Maximum Independent Set Problem**, computing **α(G)**, is Turing reducible in polynomial time, relative to the size of the representation of **G**, to the **Independent Set** decision problem. Your algorithm should ask no more than $\log_2(|V|)$ questions of an Oracle for the **Independent Set** decision problem, **IS(V,k)** (does **V** have an independent set of size **k**?) You must present detailed pseudo code. I recommend you consider using the **ceiling** function in your code. Hint: **α(G) = |V|** is the largest possible **Independent Set** size but can only be the correct value if **|E| = 0** (totally disconnected graph). The worst case, **α(G) = 1**, occurs only for a totally connected graph (every vertex connected to every other one). Thus, the range of values is **1 ≤ α(G) ≤ |V|.**

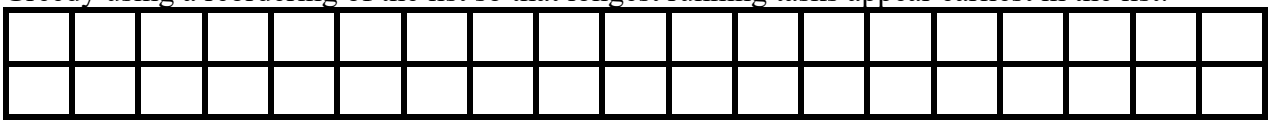
1 **c.)** Using terms like **P**, **NP**, **NP-Complete**, **NP-Equivalent**, **NP-Hard**, and **NP-Easy**, what is the most precise categorization you can provide for the **Maximum Independent Set Problem**?

6 **10.** Consider the following set of independent tasks with associated task times:
(T1,3), (T2,5), (T3,7), (T4,6), (T5,2), (T6,8), (T7,1)
Fill in the schedules for these tasks under the associated strategies below.

Greedy using the list order above:

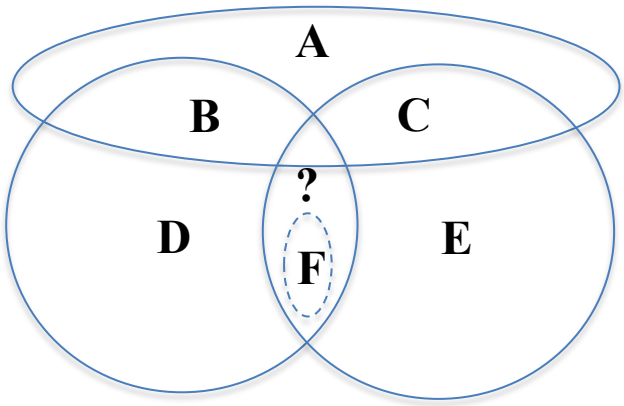


Greedy using a reordering of the list so that longest running tasks appear earliest in the list:



11. Consider the Venn diagram below. Identify the alphabetically labeled regions below (each representing a set of decision problems) to indicate characterizations as **co-NP**, **co-NP Complete**, **NP**, **NP-Complete**, **NP-Hard** and **P**. To answer this, just write in the labels associated with **A**, **B**, **C**, **D**, **E**, and **F**. I gave you **F** for free.

6 **A**_____ **B**_____ **C**_____ **D**_____ **E**_____ **F** ***P* (det. poly)**



3 If the area labeled "**?**" contains any sets outside of **F (P)**, what characteristics would these sets have and would their existence settle the question, **P = NP?** If so, how; if not, why not?

3 What is the consequence of a proof that set **F = P = D ∩ E**? This means that the area labeled "**?**" is empty. In particular, would that settle the question, **P = NP?** If so, how; if not, why not?

12. An **ATM (Alternating Turing Machine)** can be used to solve many interesting problems.

3 **a.)** What are the two possible node types for the root of each non-empty subtree used in an **ATM**? What are the semantics of each type?

2 **b.)** What is the order of execution, relative to the size, **N**, of a given CNF Boolean expression, of the fastest parallel algorithm to solve **QSAT** in this model of computation? Briefly justify your claim.