COT 6410 Spring 2019 Final Exam Name: _KEY_
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).

B = { a^n b^n | n>0 }   A = {c}   C = ∅, which is clearly regular

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).

B = { a^n b^n | n>0 }   A = {λ}   C = { a^n b^n | n>0 }, which is clearly a non-regular CFL

6 **2.** Let set **A** be a **non-empty recursive** set, and let **B** be an **RE non-recursive** set. For each of **(REC) non-empty 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) f_A, and **B** is the range of some total recursive function 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).

B = K = { f | f(f)↓ }   A = {0}   C = {0}, which is clearly REC

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).

B = K = { f | f(f)↓ }   A = {1}   C = K, which is clearly RE non-recursive

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

f_C(<x,y>) = f_B(x) // f_A(y)   This enumerates all and only the members of B // A

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 <u>and</u> 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.

Assume S is the range of f_S, which is monotonically increasing. Then S can be decided by

$$\chi_S(x) = \exists y_{\, y \le x} \, [\, f_S(y) == x \,]$$

As f_S is mon. increasing, it will enumerate x on or before the x-th element listed, or it will never enumerate x if it is not in S. This ensures χ_S is a proper characteristic function for S.

Assume S is infinite recursive. Then S has a characteristic function χ_S. We can then enumerate S by

$f_S(0) = \mu y \, [\, \chi_S(y) \,]$ ***First item in S***

$f_S(x+1) = \mu y_{\, y > f_S(x)} \, [\, \chi_S(y) \,]$ ***Next item from an infinite set***

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

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

4 **5.** Let $\mathbf{P} = \langle\langle x_1,x_2,\ldots,x_n\rangle, \langle y_1,y_2,\ldots,y_n\rangle\rangle$, $x_i,y_1 \in \Sigma^+$, $1\leq i\leq n$, 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 $\mathcal{L}(\mathbf{G})$.

Define $\mathbf{G} = (\{S, T\} \cup \Sigma, \{*\}, S, R)$, where **R** is the set of rules:

$S \rightarrow x_i\, S\, y_i^R \mid x_i\, T\, y_i^R \qquad 1 \leq i \leq n$ **(Note: the superscripted R means Reversed)**

// Write the rules for the rest of this CSG.

$a\,T\,a \rightarrow *\,T\,* \qquad \forall a \in \Sigma$

$*\,a \rightarrow a\,* \qquad \forall a \in \Sigma$

$a\,* \rightarrow *\,a \qquad \forall a \in \Sigma$

$T \rightarrow *$

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**.

$f \in OR \Leftrightarrow \forall x\, \exists\, t\, [\, STP(f, 2x, t)\ \&\ STP(f, 2x+1, t)\ \&\ (\, VALUE(f, 2x+1, t) > VALUE(f, 2x, t)\,)\,]$

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

Non-Trivial: $I(x) = x \in OR$

$Z(x) = 0 \notin OR$

Immune to Implementation (Rice's Strong Version)

Let f and g be two arbitrary indices of partial recursive functions such that $\forall x\, f(x) = g(x)$

$f \in OR \Leftrightarrow \forall x\, f(2x+1) > f(2x) \Leftrightarrow \forall x\, g(2x+1) > g(2x) \Leftrightarrow g \in OR$

Thus, OR is undecidable based on Rice's Theorem (Strong Version)

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

Let f be an arbitrary index of some partial recursive function

Define $G_f(x) = \exists\, y\, [\, f(2x+1) > f(2x)\,]$

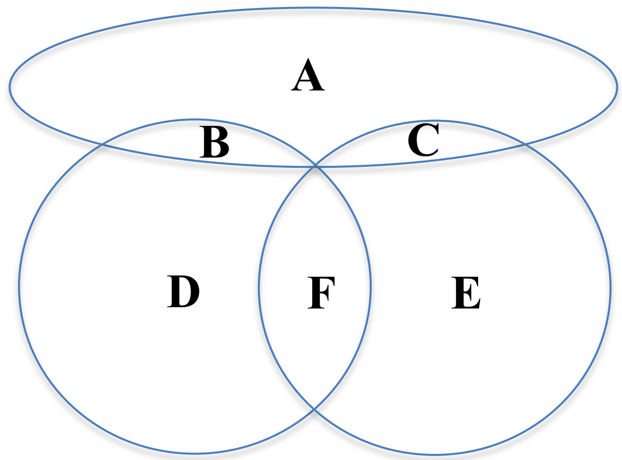
Clearly $\forall x\, G_f(x) = 1$ ***if*** $f \in OR$ ***and*** $\exists\, x\, G_f(x)\uparrow$ ***if*** $f \notin OR$

$f \in OR \Leftrightarrow \forall x\, f(2x+1) > f(2x) \Leftrightarrow \forall x\, G_f(x) = 1 \Rightarrow G_f \in Total$

$f \notin OR \Leftrightarrow \exists x$ ***either*** $f(2x+1)\uparrow$ ***or*** $f(2x)\uparrow$ ***or*** $f(2x+1) \leq f(2x) \Leftrightarrow \exists\, x\, G_f(x)\uparrow \Rightarrow G_f \notin Total$

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 *RE-Hard* **B** *RE-Complete* **C** *Co-RE-Complete* **D** *RE* **E** *Co-RE* **F** *REC*



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	*1*				*1*	*1*
~a	*1*			*1*		
b		*1*			*1*	
~b		*1*		*1*		*1*
c			*1*		*1*	*1*
~c			*1*	*1*		
C1				*1*		
C1'				*1*		
C2					*1*	
C2'					*1*	
C3						*1*
C3'						*1*
	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.

G has an Independent set of size k iff α(G) ≥ k

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|.**

```
int lo = 1; int hi = |V|;
while (lo < hi) {
        int mid = ceiling((hi+lo)/2);          // Integer divide occurs here
        if (IS(V,mid)) lo = mid;
        else hi = mid-1;
return hi;
```

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**?

The problem is NP-Equivalent.

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:

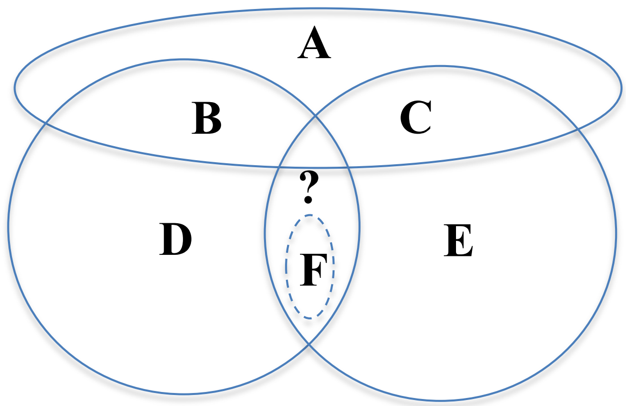
T1	*T1*	*T1*	*T3*	*T3*	*T3*	*T3*	*T3*	*T3*	*T3*	*T5*	*T5*	*T7*						
T2	*T2*	*T2*	*T2*	*T2*	*T4*	*T4*	*T4*	*T4*	*T4*	*T4*	*T6*	*T6*	*T6*	*T6*	*T6*	*T6*	*T6*	*T6*

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

T6	*T6*	*T6*	*T6*	*T6*	*T6*	*T6*	*T6*	*T2*	*T2*	*T2*	*T2*	*T2*	*T1*	*T1*	*T1*			
T3	*T3*	*T3*	*T3*	*T3*	*T3*	*T3*	*T4*	*T4*	*T4*	*T4*	*T4*	*T4*	*T5*	*T5*	*T7*			

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** ***NP-Hard*** **B** ***NP-Complete*** **C** ***Co-NP-Complete*** **D** ***NP*** **E** ***Co-NP*** **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?

This mean that there is an element in NP $\cap$ co-NP that is not in F (the set P). Since such an item must be in NP to be in the intersection, the implication is then that P $\neq$ NP.

3 What is the consequence of a proof that set **F** = **P** = **D** $\cap$ **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?

This mean that NP $\cap$ Co-NP = F = P. That is similar to REC being the intersection of RE and Co-RE. As there, this does not mean RE = Co-RE (known to not be true) or RE = REC (also known to not be true), so we cannot conclude or deny that NP = Co-NP or that P = NP.

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?

A node can be an "or" ($\vee$) or and "and" ($\wedge$) type.
If it's an "or", it returns true if any of the branches of the subtree return "true". If it's an "and", it returns true if all of the branches of the subtree return "true".

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.

Linear complexity: The depth of a tree branch is no greater than the number of variables, so it is limited by N as is checking an expression to see if it is true, so the entire process is O(N).