COT 6410 Spring2015 Sample Final and Midterm Makeup Exams Key

1. Let set **A** be recursive, **B** be re non-recursive and **C** be non-re. Choosing from among **(REC) recursive**, **(RE) re non-recursive**, **(NR) non-re**, categorize the set **D** in each of a) through d) by listing **all** possible categories. No justification is required.

   a.) D = ~C **RE, NR**

   b.) D ⊆ (A∪C) **REC, RE, NR**

   c.) D = ~B **NR**

   d.) D = B – A **REC, RE**

2. Choosing from among **(D) decidable**, **(U) undecidable**, **(?) unknown**, categorize each of the following decision problems. No proofs are required.

Problem / Language Class	Regular	Context Free	Context Sensitive
$L = \Sigma^*$?	*D*	*U*	*U*
$L = \phi$?	*D*	*D*	*U*
$L = L^2$?	*D*	*U*	*U*
$x \in L^2$, for arbitrary x ?	*D*	*D*	*D*

3. Use **PCP** to show the undecidability of the problem to determine if the intersection of two context free languages is non-empty. That is, show how to create two grammars G_A and G_B based on some instance $P = \langle\langle x_1,x_2,\ldots,x_n\rangle, \langle y_1,y_2,\ldots,y_n\rangle\rangle$ of **PCP**, such that $L(G_A) \cap L(G_B) \neq \phi$ iff **P** has a solution. Assume that **P** is over the alphabet **Σ**. You should discuss what languages your grammars produce and why this is relevant, but no formal proof is required.

$G_A = (\{ A \} , \Sigma \cup \{ [\,i\,] \mid 1 \le i \le n \} , A , P_A)$ $G_B = (\{ B \} , \Sigma \cup \{ [\,i\,] \mid 1 \le i \le n \} , B , P_B)$

$P_A : A \rightarrow x_i\, A\, [\,i\,] \mid x_i\, [\,i\,]$ $P_B : A \rightarrow y_i\, B\, [\,i\,] \mid y_i\, [\,i\,]$

$L(G_A) = \{ x_{i_1} x_{i_2} \ldots x_{i_p} [i_p] \ldots [i_2] [i_1] \mid p \ge 1, 1 \le i_t \le n, 1 \le t \le p \}$

$L(G_B) = \{ y_{j_1} y_{j_2} \ldots y_{j_q} [j_q] \ldots [j_2] [j_1] \mid q \ge 1, 1 \le j_u \le n, 1 \le u \le q \}$

$L(G_A) \cap L(G_B) = \{ w\, [k_r] \ldots [k_2] [k_1] \mid r \ge 1, 1 \le k_v \le n, 1 \le v \le r \}$, where

$$w = x_{k_1} x_{k_2} \ldots x_{k_r} = y_{k_1} y_{k_2} \ldots y_{k_r}$$

If $L(G_A) \cap L(G_B) \neq \phi$ then such a w exists and thus $k_1, k_2, \ldots, k_r$ is a solution to this instance of PCP. This shows that a decision procedure for the non-emptiness of the intersection of CFLs implies a decision procedure for PCP, which we have already shown is undecidable. Hence, the non-emptiness of the intersection of CFLs is undecidable. Q.E.D.

4. Consider the set of indices **CONSTANT = { f | ∃K ∀y [φ_f(y) = K] }**. Use Rice's Theorem to show that **CONSTANT** is not recursive. Hint: There are two properties that must be demonstrated.

First, show CONSTANT is non-trivial.
Z(x) = 0, which can be implemented as the TM R, is in CONSTANT
S(x) = x+1, which can be implemented by the TM C_1R, is not in CONSTANT
Thus, CONSTANT is non-trivial

Second, let f, g be two arbitrary computable functions with the same I/O behavior.
That is, ∀x, if f(x) is defined, then f(x) = g(x); otherwise both diverge, i.e., f(x)↑ and g(x)↑
Now, f ∈ CONSTANT
⇔ ∃K ∀x [f(x) = K] by definition of CONSTANT
⇔ ∀x [g(x) = C] where C is the instance of K above, since ∀x [f(x) = g(x)]
⇔ ∃K ∀x [g(x) = K] from above
⇔ g ∈ CONSTANT by definition of CONSTANT

Since CONSTANT meets both conditions of Rice's Theorem, it is undecidable. Q.E.D.

5. Show that **CONSTANT $\equiv_m$ TOT**, where **TOT = { f | ∀y φ_f(y)↓ }**.

CONSTANT $\leq_m$ TOT
Let f be an arbitrary effective procedure.
Define g_f by
g_f (0) = f(0)
g_f (y+1) = f(y+1) + μ z [f(y+1) = f(y)]
Now, if f ∈ CONSTANT then ∀y [f(y)↓ and [f(y+1) = f(y)]].
Under this circumstance, μ z [f(y+1) = f(y)] is 0 for all y and g_f (y) = f(y) for all y.
Clearly, then g_f ∈ TOT
If, however, f ∉ CONSTANT then ∃y [f(y+1) ≠ f(y)] and thus, ∃y g_f (y)↑.
Choose the least y meeting this condition.
If f(y)↑ then g_f (y)↑ since f(y) is in g_f (y)'s definition (the 1st term).
If f(y)↓ but [f(y+1) ≠ f(y)] then g_f (y)↑ since μ z [f(y+1) = f(y)]↑ (the 2nd term).
Clearly, then g_f ∉ TOT
Combining these, f ∈ CONSTANT ⇔ g_f ∈ TOT and thus CONSTANT $\leq_m$ TOT

TOT $\leq_m$ CONSTANT
Let f be an an arbitrary effective procedure.
Define g_f by
g_f (y) = f(y) – f(y)
Now, if f ∈ TOT then ∀y [f(y)↓] and thus ∀y g_f (y) = 0 . Clearly, then g_f ∈ CONSTANT
If, however, f ∉ TOT then ∃y [f(y)↑] and thus, ∃y [g_f (y)↑]. Clearly , then g_f ∉ CONSTANT
Combining these, f ∈ TOT ⇔ g_f ∈ CONSTANT and thus TOT $\leq_m$ CONSTANT

Hence, CONSTANT $\equiv_m$ TOT. Q.E.D.

6. 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 φ_f(y) converges in no fewer than y steps }.
We can deny the 2nd condition of Rice's Theorem since
Z, where Z(x) = 0, implemented by the TM R converges in one step no matter what x is and hence is not in AT-LEAST-LINEAR

Z', defined by the TM $\mathcal{R}\mathcal{L}$ R, is in AT-LEAST-LINEAR

However, $\forall$x [Z(x) = Z'(x)], so they have the same I/O behavior and yet one is in and the other is out of AT-LEAST-LINEAR, denying the 2nd condition of Rice's Theorem

7. The trace language of a computational device like a Turing Machine is a language of the form
Trace = { $C_1\#C_2\#\ \dots\ C_n\#$ | $C_i \Rightarrow C_{i+1}$, $1 \le i < n$ }
Trace is Context Sensitive, non-Context Free. Actually, a trace language typically has every other configuration word reversed, but the concept is the same. Oddly, the complement of such a trace is Context Free. Explain what makes its complement a **CFL**. In other words, describe the characteristics of this complement and why these characteristics are amenable to a **CFG** description.
The complement of a trace needs to include strings that either do not look like a trace (that's easy) or look like one, but have one or more errors. By one or more errors, we just mean that there is a pair $C_j\#C_{j+1}\#$ where it is not the case that $C_j \Rightarrow C_{j+1}$. A PDA can guess which configuration starts this pair, push that configuration into its stack and check that the next one is in error (of course, this generally means one element of the pair is reversed). Such checking is within the capabilities of a PDA.

8. We demonstrated a proof that the context sensitive languages are not closed under homomorphism, To start, we assumed **G = (N, Σ, S, P)** is an arbitrary Phrase Structured Grammar, with **N** its set of non-terminals, **Σ** its terminal alphabet, **S** its starting non-terminal and **P** its productions (rules). Since **G** is a PSG, it can have length increasing, length preserving and length decreasing rules. We wished to convert **G** to a CSG, **G' = (N', Σ', S', P')** where there are no rules that are length decreasing (since a CSG cannot have these). We developed a way to pad the length decreasing rules from **G** and then a homomorphism that gets rid of these padding characters. Define **G'** and the homomorphism **h** that we discussed in class and then briefly discuss why this new grammar and homomorphism combine so **h(L(G')) = L(G)**, thereby showing that all re sets are the homomorphic images of CSLs.
Define N' = N ∪ {S', D}, where D and S' are new symbols;
Σ' = Σ ∪ {\$}, where \$ is a new symbol;
P' contains
- **S' → S\$ is in P'**
- **If $\alpha \to \beta$ is in P and $|\alpha| \le |\beta|$, then $\alpha \to \beta$ is in P'**
- **If $\alpha \to \beta$ is in P and $|\alpha| > |\beta|$, then $\alpha \to \beta D^k$ is in P', where k = $|\alpha|$ - $|\beta$**
- **Dx → xD is in P', for all x ∈ N ∪ Σ**
- **D\$ → \$\$ is in P'**

It is clear that these rules are all length increasing or length preserving and hence G' is a CSG.
L(G') = {w$\$^j$ | w ∈ L(G) and j is some integer >0 }
Define the homomorphism h by
- **h(a) = a for all a ∈ Σ**
- **h(\$) = λ (the string of length 0)**

h(L(G')) = { w | w∈ L(G) } = L(G)
This completes our constructive justification.

9. We described the proof that **3SAT** is polynomial reducible to Subset-Sum.

a.) Describe **Subset-Sum**

Let n_1, n_2, …, n_k, G be a set of k positive whole numbers and G be a goal number. The decision problem is to determine if there is a subset n_{i1}, n_{i2}, .., n_{ij} of the original set that sums to G.

b.) Show that **Subset-Sum** is in **NP**

Let n_1, n_2, …, n_k, G be an instance of SubsetSum and let n_{i1}, n_{i2}, .., n_{ij} be a proposed subset. We merely need to add these j numbers together and check that they sum to G. If so, we verify the proposed solution; else we reject it. That process is linear and hence there is a polynomial time verifier, and so SubsetSum is in NP.

c.) Assuming a **3SAT** expression **(a + ~b + c) (~a + b + ~c)**, fill in the upper right part of the reduction from **3SAT** to **Subset-Sum**.

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

d.) 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**.

a = T; b = T; c = T

a	1 0 0 1 0
b	0 1 0 0 1
c	0 0 1 1 0
C1	0 0 0 1 0
C2	0 0 0 0 1
C2'	0 0 0 0 1
SUM	1 1 1 3 3

10. **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**.

Look at notes

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

Look at notes

11. Consider the decision problem asking if there is a coloring of a graph with at most k colors, and the optimization version that asks what is the minimum coloring number of a graph. You can reduce in both directions. So, do that. Make sure you carefully explain for each direction just what it is that you are proving.

1. Show that k-Color is polynomial time Turing reducible to MinColor:
Let G=(V, E) be an arbitrary graph and k>0 an arbitrary positive whole number. We can solve k-Color by asking an oracle for MinColor to provide the minimum coloring of G. We then check the MinColor answer. If it is less than or equal to k, we answer yes; else we answer no. This proves that MinColor is NP-Hard, based on our knowledge that k-Color is NP-Complete.

2. Show that MinColor is polynomial time Turing reducible to k-Color:
Let G=(V, E) be an arbitrary graph. Let n=|V|. Do a binary search using an oracle for k-Color in order to find the minimum k such that k-Color returns yes. This takes at most log n questions of the oracle. Thus, MinColor is NP-Easy since we asked just log n questions, and spent just polynomial time at each stage of this Turing reduction. Together, these show MinColor is in NP-Equivalent.

12. 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.

QSAT is NP-Hard. This is so since SAT trivially reduces to QSAT (it is a subproblem of QSAT). Since SAT is known to be NP-Complete then some NP-Complete problem polynomially reduces to QSAT. This makes QSAT NP-Hard. As we cannot (at least not yet) show QSAT is in NP, then NP-Hard is the best we can do.

13. Consider the following set of independent tasks with associated task times:
(T1,7), (T2,6), (T3,2), (T4,5), (T5,6), (T7,1), (T8,2)
Fill in the schedules for these tasks under the associated strategies below.

Greedy using the list order above:

T1	T1	T1	T1	T1	T1	T1	T4	T4	T4	T4	T4	T7	T8	T8
T2	T2	T2	T2	T2	T2	T3	T3	T5	T5	T5	T5	T5	T5	

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

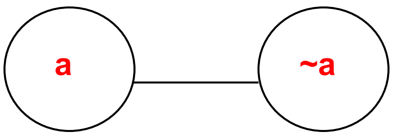
T1	T1	T1	T1	T1	T1	T1	T4	T4	T4	T4	T4	T3	T3	T7
T2	T2	T2	T2	T2	T2	T5	T5	T5	T5	T5	T5	T8	T8	

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

T7	T8	T8	T2	T2	T2	T2	T2	T2	T2	T1	T1	T1	T1	T1	T1	T1
T3	T3	T4	T4	T4	T4	T4	T5	T5	T5	T5	T5	T5				

14. 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.

Vertex Cover
Must Cover each Edge
Set goal to min vertices
Must choose one but not both are needed
This translates to choosing a or ~a



3-Color
Cannot choose B for either a or ~a
So one must be T and other F