Generally useful information.

- 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. Unlike the text, I find it convenient to 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 predicate **~P(x)** is the logical complement of predicate **P(x).**
- The minus symbol, –, when applied to sets is set difference, so $\mathbf{S - T = \{x \mid x \in S \ \&\& \ x \notin T\}}$.
- The absolute value, **|z|**, is the magnitude of **z**. Thus, **|x-y|** is the difference between **x** and **y**, when **x** and **y** are both non-negative.
- 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 predicate. Thus, it evaluates to **0** (**false**), if $\mathbf{x \notin S}$.
- When I say a set **S** is 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 $\mathbf{g_S}$.
  3. **S** is recognizable by a Turing Machine.
- If I say a function **g** is partially computable, then there is an index **g** (I know that's overloading, but that's okay as long as we understand each other), such that $\mathbf{\Phi_g(x) = \Phi(g, x) = g(x)}$. Here $\mathbf{\Phi}$ is a universal partially recursive function.
  Moreover, there is a total 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 total 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**, but we don't care about the value produced. In effect, this just means that **x** is in the domain of **f**.
- The notation $\mathbf{f(x)}\uparrow$ means **f** diverges when computing with input **x**. In effect, this just means that **x** is <u>**not**</u> in the domain of **f**.
- 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 $\mathbf{f(x)}\downarrow$. The set of all such pairs is a classic re non-recursive one. The set of all such $\mathbf{\langle f,x \rangle}$ is denoted $\mathbf{K_0}$. A related set **K** is the set of all **f** that halt on their own indices. Thus, $\mathbf{K = \{ f \mid \Phi_f(f)\downarrow \}}$ and $\mathbf{K_0 = \{\langle f,x \rangle \mid \Phi_f(x)\downarrow \}}$
- 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). The set of all such function indices is a classic non re one and is often called **TOTAL**.

COT 4210 Fall 2014 Final Exam Sample Questions

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. Prove that the **Halting Problem** (the set K_0) is not recursive (decidable) within any formal model of computation. (Hint: A diagonalization proof is required here.)

Assume we can decide the halting problem. Then there exists some total function Halt such that

$$\text{Halt}(x,y) = \begin{cases} 1 & \text{if } [x]\,(y) \text{ is defined} \\ 0 & \text{if } [x]\,(y) \text{ is not defined} \end{cases}$$

Here, we have numbered all programs and [x] refers to the x-th program in this ordering. We can view Halt as a mapping from ℵ into ℵ by treating its input as a single number representing the pairing of two numbers via the one-one onto function

pair(x,y) = <x,y> = 2^x (2y + 1) – 1

with inverses

$$\langle z\rangle_1 = \exp(z+1,1)$$

$$\langle z\rangle_2 = (((z + 1) \,//\, 2^{\langle z\rangle 1}) - 1) \,//\, 2$$

Now if Halt exist, then so does Disagree, where

$$\text{Disagree}(x) = \begin{cases} 0 & \text{if Halt}(x,x) = 0\text{, i.e, if } \Phi_x(x) \text{ is not defined} \\ \mu y\,(y == y+1) & \text{if Halt}(x,x) = 1\text{, i.e, if } \Phi_x(x) \text{ is defined} \end{cases}$$

Since Disagree is a program from ℵ into ℵ , Disagree can be reasoned about by Halt. Let d be such that Disagree = Φ_d, then

Disagree(d) is defined ⇔ Halt(d,d) = 0 ⇔ Φ_d (d) is undefined ⇔ Disagree(d) is undefined

But this means that Disagree contradicts its own existence. Since every step we took was constructive, except for the original assumption, we must presume that the original assumption was in error. Thus, the Halting Problem is not solvable.

3. Using reduction from the known undecidable **HasZero, HZ = { f | ∃x f(x) = 0 }**, show the non-recursiveness (undecidability) of the problem to decide if an arbitrary primitive recursive function **g** has the property **IsZero, Z = { f | ∀x f(x) = 0 }**.

HZ = { f | ∃x ∃t [STP(f, x, t) & VALUE(f, x, t) == 0] }
Let f be the index of an arbitrary effective procedure.
Define g_f(y) = 1 - ∃x ∃t [STP(f, x, t) & VALUE(f, x, t) == 0]
If ∃x f(x) = 0, we will find the x and the run-time t, and so we will return 0 (1 – 1)
If ∀x f(x) ≠ 0, then we will diverge in the search process and never return a value.
Thus, f ∈ HZ iff g_f ∈ Z = { f | ∀x f(x) = 0 }.

4. 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**
L = Σ* ?	***D***	***U***
L = ϕ ?	***D***	***D***
x ∈ L^2, for arbitrary x ?	***D***	***D***

5. Choosing from among **(Y) yes**, **(N) No**, **(?) unknown**, categorize each of the following closure properties. No proofs are required.

Problem / Language Class	**Regular**	**Context Free**
Closed under intersection?	***Y***	***N***
Closed under quotient?	***Y***	***N***
Closed under quotient with Regular languages?	***Y***	***Y***
Closed under complement?	***Y***	***N***

6. Prove that any class of languages, ***C***, closed under union, concatenation, intersection with regular languages, homomorphism and substitution (e.g., the Context-Free Languages) is closed under **MissingMiddle**, where, assuming L is over the alphabet Σ,
MissingMiddle(L) = { xz | $\exists$y $\in \Sigma^*$ such that xyz $\in$ L }
You must be very explicit, describing what is produced by each transformation you apply.

Define the alphabet $\Sigma' = \{ a' \mid a \in \Sigma \}$, where, of course, a' is a "new" symbol, i.e., one not in Σ.

Define homomorphisms g and h, and substitution f as follows:
$g(a) = a' \;\; \forall a \in \Sigma$ $\quad$ $h(a) = a$; $h(a') = \lambda \;\; \forall a \in \Sigma$ $\quad$ $f(a) = \{a, a'\} \;\; \forall a \in \Sigma$

Consider $R = \Sigma^* \bullet g(\Sigma^*) \bullet \Sigma^* = \{ x\,y'\,z \mid x, y, z \in \Sigma^* \text{ and } y' = g(y) \in \Sigma'^* \}$
Σ^* is regular since it is the Kleene star closure of a finite set.
$g(\Sigma^*)$ is regular since it is the homomorphic image of a regular language.
R is regular as it is the concatenation of regular languages.

Now, $f(L) = \{ f(w) \mid w \in L \}$ is in C since C is closed under substitution. This language is the set of strings in L with randomly selected letters primed. Any string w$\in$L gives rise to $2^{|w|}$ strings in f(L).

$f(L) \cap R = \{ x\,y'\,z \mid x\,y\,z \in L \text{ and } y' = g(y) \}$ is in C since C is closed under intersection with regular languages.

MissingMiddle(L) = $h(\, f(L) \cap R\,) = \{ x\,z \mid \exists y \in \Sigma^* \text{ such that } xyz \in L \}$ which is in C, since C is closed under homomorphism. Q.E.D.

7. 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 $\mathbf{G_A}$ and $\mathbf{G_B}$ based on some instance $\mathbf{P} = \langle\langle x_1,x_2,\ldots,x_n\rangle, \langle y_1,y_2,\ldots,y_n\rangle\rangle$ of **PCP**, such that $\mathbf{L(G_A) \cap L(G_B) \neq \phi}$ 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.

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

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

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

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

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

8. 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 is in CONSTANT
S(x) = x+1 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.

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

CONSTANT $\le_m$ 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 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 $\le_m$ TOT

TOT $\le_m$ 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 $\le_m$ CONSTANT

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

10. Why does Rice's Theorem have nothing to say about each of the following? Explain by showing some condition of Rice's Theorem that is not met by the stated property.

a.) **AT-LEAST-LINEAR = { f | ∀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 TM $\mathcal{R}\ \mathcal{L}$ R, is in AT-LEAST-LINEAR since takes over 2*|input| steps.

However, ∀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

b.) **HAS-IMPOSTER = { f | ∃ g [g≠f** and **∀y [φ_g(y) = φ_f(y)]] }**.

We can deny the 1st condition of Rice's Theorem since all functions have an imposter. To see this, consider, for any function f, the equivalent but distinct function g(x) = f(x) + 0. Thus, HAS-IMPOSTER is trivial since it is equal to ℵ, the set of all indices.

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

a.) Describe **Subset-Sum**

Given a sequence of n positive integers, <i1,…in> and a goal, G, which is also a positive integer, is there a subset of the integers from the sequence that sums to the goal value?

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

Give a proposed solutions we can check if their sum equals G in linear time. Any decision problem where a solution can be verified in linear time is in NP, so we are done.

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

	a	**b**	**c**	**a + ~b + c**	**b + b + ~c**
a	**1**			**1**	
~a	**1**				
b		**1**			**1 or 2**
~b		**1**		**1**	
c			**1**	**1**	
~c			**1**		**1**
C1				**1**	
C1'				**1**	
C2					**1**
C2'					**1**
	1	**1**	**1**	**3**	**3**

12. Use the appropriate Pumping Lemmas to show:

a.) **{ ww | w is over {a,b} }** is not Regular

Assume the language L = { ww | w is over {a,b} } is Regular. Let N>0 be the value associated with L by the Pumping Lemma for Regular languages.
$a^Nba^Nb \in L$.
By Pumping Lemma, $a^Nba^Nb = xyz$, for some strings x,y,z over {a,b},
where |y| > 0, |xy| ≤ N and $\forall i \geq 0\ xy^iz \in L$. Because |xy| ≤ N, y is a string in $\{a\}^+$.
Let i=0, then $xy^iz = xz = a^{N-|y|}ba^Nb$. Since |y|>0, there are fewer a's preceding the first b in the string than preceding the second one, so it is not in L contradicting the Pumping Lemma.

b.) **{ ww | w is over {a,b} }** is not Context Free

Assume the language L = { ww | w is over {a,b} } is Context Free. Let N>0 be the value associated with L by the Pumping Lemma for Context Free languages.
$a^Nb^Na^Nb^N \in L$.
By Pumping Lemma, $a^Nb^Na^Nb^N = uvwxy$, for some strings u,v,w,x,y over {a,b},
where |vx| > 0, |vwx| ≤ N and $\forall i \geq 0\ uv^iwx^iy \in L$.
All cases collapse into the following analysis. vwx must include at most one of the 'a' sequences and at most one of the 'b' sequences; moreover it must have at least one of these cases (first 'a' sequence but not second; first 'b' sequence but not second; second 'a' sequence but not first; or second 'b' sequence but not first). Set i=0 and we have removed letters from one of the 'a' sequences and/or one of the 'b' sequences, but not the other. This denies that uwy is in L, thereby contradicting the Pumping Lemma.

13. Write a context-sensitive grammar for the complement of **{ ww | w is over {a,b} }**

S → L<Odd> | AB | BA
<Odd> → L<Even> | λ
<Even> → L<Odd>
A → L A L | a
B → L B L | b
L → a | b