**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 $\mu$ **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 $\mu$ **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 $\mu$ **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 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 $\chi_S$, such that $\mathbf{x \in S \Leftrightarrow \chi_S(x)}$. Note $\chi_S$ 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 $\mathbf{g_S}$.
- 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 $\mathbf{\Phi_g(x) = \Phi(x, g) = g(x)}$. 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**, 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 **not** 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, $\mathbf{K_0}$, is a classic re non-recursive one.
- 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 **(TOTAL)** is a classic non re one.
- $\mathbf{A \le_m B}$ (**A** many-one reduces to **B**) means that there exists a total recursive function **f** such that $\mathbf{x \in A \Leftrightarrow f(x) \in B}$. If $\mathbf{A \le_m B}$ and $\mathbf{B \le_m A}$ then we say that $\mathbf{A \equiv_m B}$ (**A** is many-one equivalent to **B**). If the reducing function is 1-1, then we say $\mathbf{A \le_1 B}$ (**A** one-one reduces to **B**) and $\mathbf{A \equiv_1 B}$ (**A** is one-one equivalent to **B**).

**COT 6410 Spring 2015 Midterm#1 Name:____________**

*12* **1**. 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.) { f | f is a Fibonacci function, i.e. f(0)=f(1)=1 and f(x+2)=f(x)+f(x+1) }** ____________________

**Justification:**

**b.) { f | if f(x) converges, it does so in more than ($2^x$) units of time }** ____________________

**Justification:**

**c.) { <f,x> | if f(x) converges, it does so in more than ($2^x$) units of time }** ____________________

**Justification:**

**d.) { f | f(x) = f(x+1) for at least one value of x }** ____________________

**Justification:**

*2* **2.** 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)** ____

*6* **3**. 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 – A (set difference)** ________________________________________

**b.) A ⊆ D (set containment)** ________________________________________

**c.) D = A × B (cross product)** ________________________________________

**d.) D = A – B (set difference)** ________________________________________

**4.** Define **NON_TRIVIAL_RANGE = ( f | |range(f)| > 1 }**.

2 **a.)** Show some minimal quantification of some known recursive predicate that provides an upper bound for the complexity of this set. (Hint: Look at **c.)** and **d.)** to get a clue as to what this must be.)

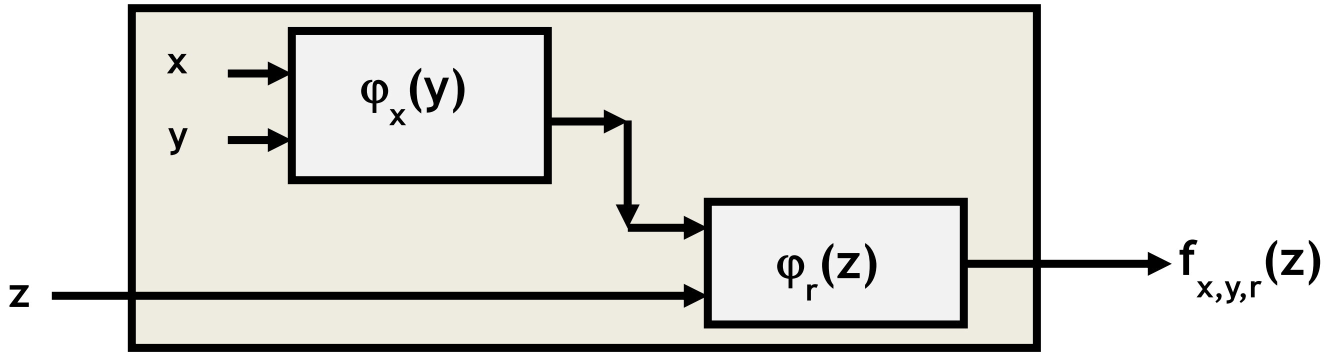
5 **b.)** Use Rice's Theorem to prove that **NON_TRIVIAL_RANGE** is undecidable.

4 **c.)** Show that $\mathbf{K_0 \le_m}$ **NON_TRIVIAL_RANGE**, where $\mathbf{K_0 = \{ <x,y> \mid \varphi_x(y)\downarrow \}}$.

4 **d.)** Show that **NON_TRIVIAL_RANGE** $\mathbf{\le_m K_0}$.

2 **e.)** From **a.)** through **d.)** what can you conclude about the complexity of **NON_TRIVIAL_RANGE (Recursive, RE, RE-COMPLETE, CO-RE, CO-RE-COMPLETE, NON-RE/NON-CO-RE)**?

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




Given the hypotheses **P** is non-trivial and is an I/O behavior and that we assume, without loss of generality that all functions with empty domains/ranges do not have property **P**, explain the meaning of this diagram by doing the following:

*2* **a.)** Indicate what **r** is, how it is chosen and how we can guarantee its existence.

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

*5* **c.)** Specify how the function $\mathbf{f_{x,y,r}}$ behaves with respect to **x**,**y** and **r**, and how this relates to the original problem, **P**, and set, $\mathbf{S_P}$.

*6* **6.** Let **S** be an arbitrary semi-decidable set. This means that **S** is the domain of some partial recursive function $\mathbf{f_s}$, whose domain is infinite. Using $\mathbf{f_S}$, show that **S** has an infinite recursive subset, call it **R**. To be complete you will need to create a characteristic function for **R**, $\chi_\mathbf{R}$, and argue that the set **R** you defined is infinite. **Hint**: Inductively define a monotonically increasing algorithm that enumerates **R**. I'll even do this part for you.

$\mathbf{f_R(0) = < \mu<x,t> [\ STP(f_S, x, t)\ ] >_1}$ **// Extract first component of $\mathbf{<x, t>}$**
$\mathbf{f_R(y+1) =}$ **// You fill this part in**
You now need to argue that $\mathbf{f_R}$ is total and monotonically increasing. From that you must argue that the set **R** enumerated by $\mathbf{f_R}$ is an infinite subset of **S** and then you must define the characteristic function $\chi_\mathbf{R}$ for **R**. I started the hardest part.

*3* **7.** We proved that $\mathbf{TOTAL = \{ f \mid \forall x\ \varphi_f(x)\downarrow \}}$ is not recursively enumerable. The proof is straightforward in that we assume the property to be so and that implies there is an algorithm **A** that enumerates the indices of all algorithms. Using the universal machine, $\varphi$, where $\mathbf{\varphi(f,x) = \varphi_f(x)}$, we have that $\mathbf{\varphi(A(f),x) = \varphi_{Af}(x)}$, that is, the value of the **f**-th algorithm at the input **x**. We then can define a new algorithm $\mathbf{D(x) = \varphi(A(x),x) + 1}$. Now you must finish the arguments that show that **D** contradicts its own existence and hence of the existence of the enumerating algorithm **A**.