COT 4210 Fall 2019 Sample Final Exam Name: *KEY*

Generally useful information.

- The notation $z = \langle x,y \rangle$ denotes some 1-1 onto pairing function with inverses $x = \langle z \rangle_1$ and $y = \langle z \rangle_2$.
- The minimization notation $\mu\, y\, [P(\ldots,y,\ldots)]$ means the least **y** (starting at **0**) such that $P(\ldots,y,\ldots)$ is true.
- 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 $y \times P(x)$, which would evaluate to either **y** (if **P(x)**) or **0** (if **not P(x)**).
- 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 set **S** is recursive (decidable) if **S** has a total recursive characteristic function χ_S, such that $x \in S \Leftrightarrow \chi_S(x)$. Note χ_S is a total predicate. Thus, it evaluates to **0** (**false**), iff $x \notin S$.
- 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 (algorithm) f_S.
  2. **S** is the domain of a partial recursive function (one that may diverge on some input) g_S.
  3. **S** is the range of a partial recursive function (one that may diverge on some input) h_S.
  4. **S** is recognizable by a Turing Machine.
  5. **S** is the language generated by a phrase structured grammar.
- 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 $\varphi_g(x) = \varphi(g, x) = g(x)$. Here φ is a universal partial recursive function (an interpreter).
  Moreover, there is a primitive recursive predicate **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**, but we don't care about the value produced. In effect, this just means that **x** is in the domain of **f**.
- The notation $f(x)\uparrow$ 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 $f(x)\downarrow$. The set of all such pairs is a classic re non-recursive one. The set of all such $\langle f,x \rangle$ is denoted K_0 or **HALT**. A related set **K** is the set of all **f** that halt on their own indices. Thus, $K = \{ f \mid \varphi_f(f) \downarrow \}$ and $K_0 = \{ \langle f,x \rangle \mid \varphi_f(x) \downarrow \}$
- 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**. It can be described as $\{ f \mid \forall x\, \varphi_f(x) \downarrow \}$.

- Remember that Context Free Languages can be generated by Context Free Grammars and recognized by non-deterministic Pushdown Automata.
- Remember that Context Sensitive Grammar rules are non-length reducing, but Phrase Structured Grammars may have length-reducing rules. Also, recall that Context Sensitive Languages are generated by Context Sensitive Grammars and recognized by Linear Bounded Automata. Phrase Structured Languages are generated by Phrase Structured Grammars and recognized by Turing Machines.
- The language **{ww | w** is a word in some alphabet with more than one letter**}** is not a **CFL**. The language **{$a^n b^n c^n$ | n≥0 }** is not a **CFL**. Both of these are, however, **CSL**s. The language **{ ww^R | w** is a word in some alphabet with more than one letter**}** is a **CFL,** but is not **Regular.** The language **{$a^n b^n$ | n≥0 }** is also a **CFL**, but is not **Regular**.
- The **Post Correspondence Problem** (**PCP**) is known to be undecidable. This problem is characterized by instances that are described by a finite alphabet, **Σ**, a number **n>0** and two **n**-ary sequences of non-empty words **<$x_1,x_2,\ldots,x_n$>, <$y_1,y_2,\ldots,y_n$>**, each in Σ^+. The question is whether or not there exists a sequence, **$i_1,i_2,\ldots,i_k$**, such that **1≤i_j≤n**, **1≤j≤k**, and **$x_{i_1}x_{i_2}\cdots x_{i_k} = y_{i_1}y_{i_2}\cdots y_{i_k}$**
- When I ask for a reduction of one set of indices to another, the formal rule is that you must produce a computable function that takes an index and produces another index having whatever property you require. However, I allow some laxness here. For example, you can start with a function, given its index, and constructively produce another function, knowing it will have a computable index.
- When I ask you to show one set of indices, **A**, is many-one reducible (or simply reducible) to another, **B**, denoted **A ≤m B,** you must demonstrate a total computable function **f**, such that **x ∈ A ⇔ f(x) ∈ B**. The stronger relationship that **A** and **B** are many-one equivalent, **A ≡m B**, requires that you show **A ≤m B** and **B ≤m A**.
- The related notions 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 **A** is in **P** if it can be solved by a deterministic Turing machine in polynomial time.
- A decision problem **A** is in **NP** if it can be solved by a non-deterministic Turing machine in polynomial time. Alternatively, **A** is in **NP** if a proposed proof of any instance having answer yes can be verified by a deterministic Turing machine in polynomial time.
- A decision problem **A** is **NP**-complete if and only if it is in **NP** and, for any problem **B** in **NP**, it is the case that **B ≤p A**.

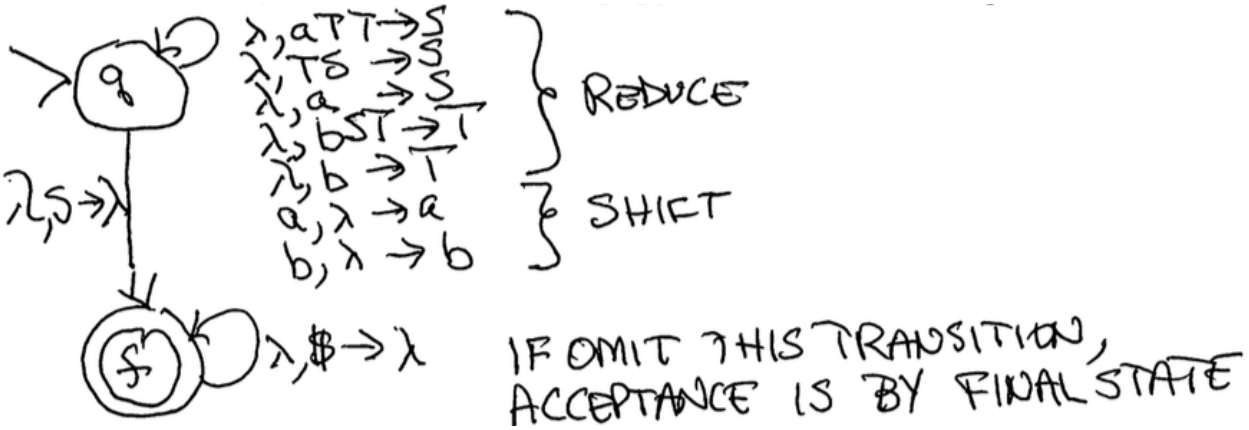
COT 4210 Fall 2019 Sample Final Exam Name: *KEY*

1. Consider the CFG **G = ({ S, T }, { a, b }, R, S)** where **R** is:

S → a T T | T S | a

T → b S T | b

a.) Present a pushdown automaton that accepts the language generated by this grammar. You may (and are encouraged) to use a transition diagram where transitions have arcs with labels of form **a, α → β** where **a ∈ Σ∪{λ}**, **α, β ∈ Γ***. Note: I am encouraging you to use extended stack operation.



b.)

What parsing technique are you using? (Circle one) **top-down** or **bottom-up**

How does your PDA accept? (Circle one) **final state** or **empty stack** or **final state and empty stack**

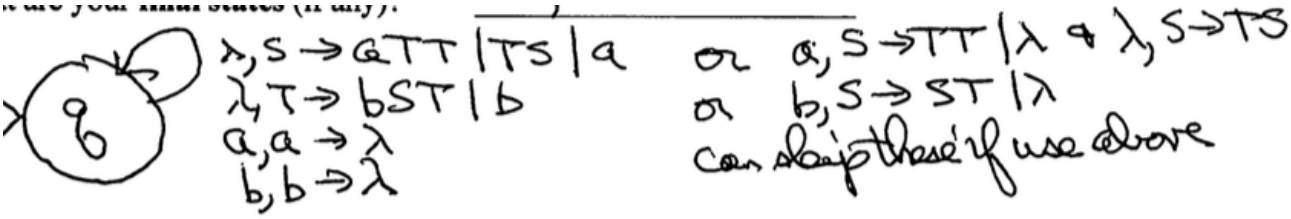
What is the **initial state**? **q**

What is the **initial stack content**? **$**

What are your **final states** (if any)? **f**

Or can have **a,S → TT | λ; λ,S → TS**

b,T → ST | λ



What parsing technique are you using? (Circle one) **top-down** or **bottom-up**

How does your PDA accept? (Circle one) **final state** or **empty stack** or **final state and empty stack**

What is the **initial state**? ***q***

What is the **initial stack content**? ***S***

What are your **final states** (if any)? ***None***

c.) Now, using the notation of **ID**s (Instantaneous Descriptions, **[q, x, z]**), describe how your PDA accepts strings generated by **G**.

[q, w, $] ⇒* [f, λ, λ] if by final state and empty stack (my solution on (a) Bottom-Up)

[q, w, S] ⇒* [q, λ, λ] if by empty stack (my solution on (a) Top-Down)

2. Choosing from among **(D) decidable**, **(U) undecidable**, categorize each of the following decision problems about grammars, **G**, and their associated languages, **L(G)**. No proofs are required. Note: Read $\supseteq$ as "contains and may equal."

Problem / Grammar Class of G	**Regular (Right Linear)**	**Context Free**	**Context Sensitive**	**Phrase Structured**
L(G) $\supseteq$ {λ}?	***D***	***D***	***D***	***U***
L(G) is infinite?	***D***	***D***	***U***	***U***
L(G) = Σ^* ?	***D***	***U***	***U***	***U***

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. **(Note that exam could require justification)**

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

b.) D $\subseteq$ (A$\cup$C) **REC, RE, NR**

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

d.) D = B – A **REC, RE**

4. Consider the following instance of the Post Correspondence Problem (**PCP**) (Look at fact sheet for definition). Out instance, **P**, is over the alphabet {a,b} and the two vectors **X** and **Y** are each of length 3.

X = (aba, bb, a); Y = (bab, b, baa)

Using the construction shown in class, produce a context free grammar, **G**, that is ambiguous if and only if the instance **P** of PCP has a solution. That is, create your context free grammar **G** based on this instance **P**, such that some string **w** has two or more distinct parses iff **P** has a solution. As **P** has a solution, **G** is ambiguous. One such solution is **2, 3, 1, 2**. To illustrate this, show the associated string that can be derived ambiguously in your grammar **G**.

S → X | Y
X → aba X [1] | bb X [2] | a X [3]
X → aba [1] | bb [2] | a [3]
Y → bab Y [1] | b Y [2] | baa Y [3]
Y → bab [1] | b [2] | baa [3]

bbaababb

5. Fill in True/False (**T**/**F**) answers for each of the following statements:

Statement	**True/False**
Any problem that is both **RE** and **CO-RE** is **Recursive**	***T***
The **UNIV** (universal) function is a **PRF***	***F***
The pairing function **<x,y>** is 1-1 onto the natural numbers	***T***
PRFs* are closed under unbounded minimization	***F***

* ***PRF = primitive recursive function***

6. Choosing from among **(REC) recursive/decidable**, **(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) A = { <f,x> | if $\varphi_f(x)$ ever converges (it might not), it takes at least 10 steps to do so }.

~STP(f,x,9) — *REC*

b.) B = { f | range(φ_f) is empty }

$\forall$<x, t> [~STP(f,x,t)]) — *coRE*

c.) C = { <f ,x> | $\varphi_f(x)\downarrow$ but takes at least 10 steps to do so }

$\exists$t [STP(f,x,t) & ~STP(f,x,9)] — *RE*

d.) D = { f | φ_f diverges for some value of x }

$\exists$x $\forall$t [~STP(f,x,t)] — *NRNC*

7. Looking back at Question 6, which of these are candidates for using Rice's Theorem to show their unsolvability? Check all for which Rice Theorem might apply.

a) _____ **b)** ___*X*___ **c)** _____ **d)** ___*X*___

8. Using the definition that **S** is recursively enumerable iff **S** is the domain of some effective procedure $\mathbf{f_S}$ (partial recursive function), prove that if both **S** and its complement **~S** are recursively enumerable (using semi-decision effective procedures $\mathbf{f_S}$ and $\mathbf{f_{\sim S}}$) then **S** is decidable. To get full credit, you must show the characteristic function for **S**, χ_S, in all cases. Also, be sure to discuss why your χ_S works.

χ_S, (x) = *STP(f_S, x, μt [STP(f_S, x, t) || STP($f_{\sim S}$, x, t)])*

Justification: ***Despite the fact that we have an unbounded search, we know it will be successful as x is either in the domain of f_S or $f_{\sim S}$. The search returns the minimum time for convergence of one of these functions and we then use that to see if x was in f_S's domain. If so, we return true, else false.***

9. Define **NAT = { f | range(f) = $\aleph$ }**. That is, **f $\in$NAT** iff **f**'s range includes every natural number.

a.) Use Rice's Theorem to prove that **NAT** is undecidable.

First, NAT is non-trivial as I(x) = x is in NAT, but C0(x) = 0 is not.

Second, let f and g be two functions whose ranges are the same.

f $\in$ NAT iff RANGE(f) = $\aleph$ iff RANGE(g) = $\aleph$ iff g $\in$ NAT

b.) Show that **TOT $\leq_m$ NAT**, where **TOT = { f | $\forall$x $\varphi_f(x)\downarrow$ }**.

Let f be an arbitrary function. Define $G_f(x) = f(x) - f(x) + x$

f $\in$ TOTAL iff $\forall$x f(x)$\downarrow$ iff $\forall$x $G_f(x) = f(x) - f(x) + x = x$ iff $\forall$x $G_f(x) = x$ implies $G_f \in NAT$

f $\notin$TOTAL iff $\exists$x f(x)$\uparrow$ iff $\exists$x $G_f(x)\uparrow$ implies $G_f \notin NAT$

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 $\varphi_f(y)$ 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 $\mathcal{L}$ $\mathcal{R}$ 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 [$\varphi_g(y) = \varphi_f(y)$]] }.**

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. Match concepts in the left with related descriptions in the right column (see first answer). Note: Every Concept must be aligned to a Description.

#	Concept	Description	Concept #
1	Problem A is in NP	The classic NP-Complete problem	*10*
2	Problem A is in co-NP	A is the problem TOTAL (set of Algorithms)	*4*
3	Problem A is in P	A is decidable in deterministic polynomial time	*3*
4	Problem A is non-RE/non-Co-RE	If B is in NP then B $\leq_P$ A	*9*
5	Problem A is NP-Complete	A is in RE and, if B is in RE, then B $\leq_m$ A	*8*
6	Problem A is RE	A is verifiable in deterministic polynomial time	*1*
7	Problem A is Co-RE	A is in NP and if B is in NP then B $\leq_P$ A	*5*
8	Problem A is RE-Complete	A is semi-decidable	*6*
9	Problem A is NP-Hard	A is the complement of B and B is RE	*7*
10	Satisfiability	A's complement is in NP	*2*

12. 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)**, fill in the right two columns of the reduction from **3SAT** to **Subset-Sum**.

	a	b	c	~a + b + ~c	~a + ~b + c
a	**1**	**0**	**0**	***0***	***0***
~a	**1**	**0**	**0**	***1***	***1***
b	**0**	**1**	**0**	***1***	***0***
~b	**0**	**1**	**0**	***0***	***1***
c	**0**	**0**	**1**	***0***	***1***
~c	**0**	**0**	**1**	***1***	***0***
C1	**0**	**0**	**0**	***1***	***0***
C1'	**0**	**0**	**0**	***1***	***0***
C2	**0**	**0**	**0**	***0***	***1***
C2'	**0**	**0**	**0**	***0***	***1***
	1	**1**	**1**	**3**	**3**

Write down a solution to above. That is, write down a subset of the rows in the main body of the matrix that sums to **11133**. Please write down leading and trailing zeroes, i.e., 5 digit numbers for each value, along with the label of the chosen row.

10011(~a) + 010010(b) + 00101(c) + 00010(C1) + 00001(C2) = 11133

13. Consider the following set of independent tasks with associated task times:
(T1,4), (T2,5), (T3,2), (T4,7), (T5,1), (T6,4), (T7,8)
Fill in the schedules for these tasks under the associated strategies below.

Greedy using the list order above:

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

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

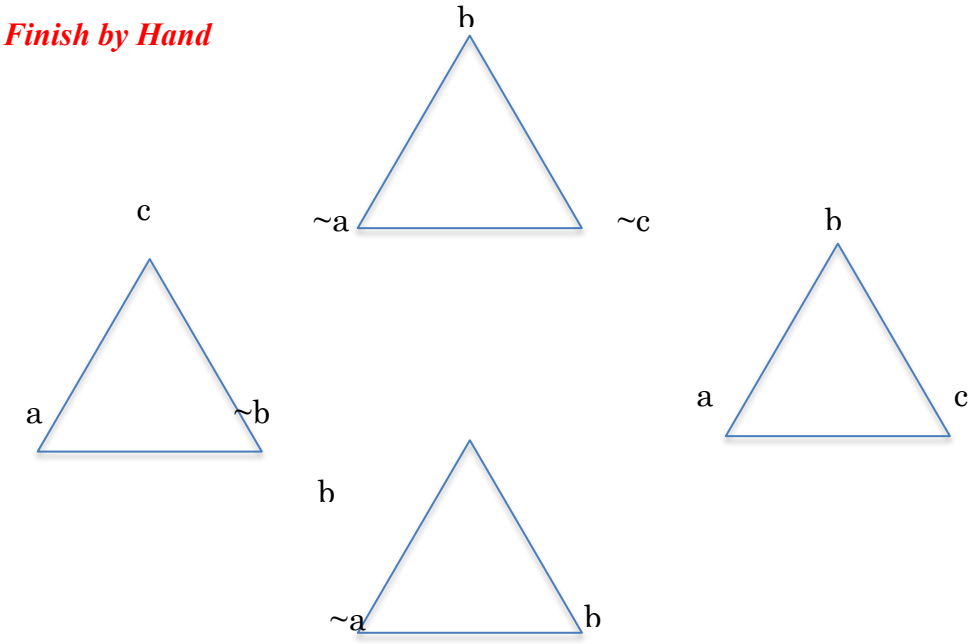
T7	*T7*	*T7*	*T7*	*T7*	*T7*	*T7*	*T7*	*T1*	*T1*	*T1*	*T1*	*T6*	*T6*	*T6*	*T6*			
T4	*T4*	*T4*	*T4*	*T4*	*T4*	*T4*	*T2*	*T2*	*T2*	*T2*	*T2*	*T3*	*T3*	*T5*				

14. 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**.

Using **3-SAT** as a known **NP-Complete** Problem, show **Independent Set** is **NP-Hard**. Just showing the construct with its gadgets for the **3-SAT** expression
(a + ~b + c) (~a + b + ~c) (a + b + c) (~a + b + b)
and indicating the value of **k**, along with a choice of **k** independent vertices, is sufficient.

k=4

Finish by Hand



Complete the proof that **Independent Set** is **NP-Complete** by effectively arguing that **Independent Set** is in **NP**.

To show this, we just need to show we can verify (or deny) a proposed solution in deterministic polynomial time. A solution is a proposed Independent Set. Given such a set, we can check first that it includes exactly k nodes, where k is the given IS size. If it passes that first test, we then make sure each such node has no neighbors that are also in the chosen set. This takes at most k* |E| steps, as we never need to look at any edge more than once since, if it were to be seen more than once, we would have already found the set to be "dependent."