COT 6410 Spring 2021 Legal Cheat Sheet Name:_____

- Formal languages are categorized via a hierarchy from the simplest to the more complex – regular, content-free, context sensitive, phrase structured. Each class in the hierarchy is properly contained in the next with the phrase-structured being equivalent to class of re (Turing recognizable) languages.
- Every class of formal languages has an associated accepting automata family and an associated generative grammar family.
- The Regular Languages, in addition to being recognized by finite state automata and generated by either right or left linear grammars, can be specified by regular expressions, the solutions to regular equations and the union of some of the classes of a right-invariant equivalence relation. Moreover, Regular Languages have unique minimal state deterministic recognizers.
- Remember that Context Free Languages can be generated by Context Free Grammars and recognized by non-deterministic Pushdown Automata. Also, membership of a string **w** in the language associated with a CFG can be checked in ($\mathbf{n^3}$), where **n = |w|**.
- 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 languages **{ww | w** is a word in some alphabet with more than one letter**}**, **{$a^n b^n c^n$ | n≥0 },** **{$a^i b^j c^k$ | k = min(i,j)},** and **{$a^i b^j c^k$ | k = max(i,j)}** are not **CFLs**. All of these are, however, **CSL**s. The languages **{ww^R | w** is a word in some alphabet with more than one letter**}, {xy | |x| = |y|, x ≠ y, x,y** are words in some alphabet with more than one letter**}** and **{$a^n b^n$ | n≥0 }** are **CFLs**, but none is **Regular**.
- Languages can be shown non-regular using the Pumping Lemma for Regular Languages or by employing Myhill-Nerode. Languages can be shown non-context-free using the Pumping Lemma for Context-Free Languages.
- Non-determinism is necessary for PDAs to recognize all CFLs, but adds nothing to the power of finite state automata, linear-bounded automata or Turing Maxhines. Oddly non-determinism detracts from the power of Factor Replacement System.
- The notation **z = <x,y>** denotes some 1-1 onto pairing function with inverses **$x = \langle z \rangle_1$** and **$y = \langle z \rangle_2$**.
- The notation **f(x)↓** 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)↑** means **f** diverges when computing with input **x**. In effect, this just means that **x** is **not** in the domain of **f**.
- 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. I define **μ y (u≤y≤v) [P(…,y)]** to be **v+1**, when no **y** satisfies this bounded minimization.
- 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 **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 **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, K_0, is a classic re non-recursive set. K_0 is also known as L_u, the universal language. The related set, **K**, is the set of all effective procedures **f** such that $f(f)\downarrow$ or more precisely $\Phi_f(f)$. **K** and K_0 are classic **RE-Complete** sets, meaning that every **RE** set many-one reduces these hardest **RE** sets.
- The **Uniform Halting Problem** is the problem to determine of an arbitrary effective procedure **f**, whether **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 $\{ f \mid \forall x\, \varphi_f(x) \downarrow \}$.
- When I ask you to show one set of indices, **A**, is many-one reducible (or simply reducible) to another, **B**, denoted $A \leq_m B$ (or simply $A \leq B$), you must demonstrate a total computable function **f**, such that $x \in A \Leftrightarrow f(x) \in B$. The stronger relationship that **A** and **B** are many-one equivalent, $A \equiv_m B$, requires that you show $A \leq_m B$ and $B \leq_m A$.
- In the computability domain, we usually categorize problems as **Recursive**, **Recursively Enumerable (RE)**, **Co-Recursively Enumerable (co-RE)**, **RE-Complete**, **Co-RE-Complete**, and **NRNC** (my own term to describe a set for which we can show a reduction from some **RE-Complete**, e.g., **TOTAL** is in **NRNC** since $K \leq_t TOTAL$).
- While reduction is a general tool to show undecidability, Rice's Theorem, though constrained, si a very easy tool to employ, provided you use it properly. Quantification is also great for estimating complexity; well at least to put an upper bound.
- 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 $\langle x_1,x_2,\ldots,x_n\rangle$, $\langle y_1,y_2,\ldots,y_n\rangle$, each in Σ^+. The question is whether or not there exists a sequence, $i_1,i_2,\ldots,i_k$, such that $1 \leq i_j \leq n$, $1 \leq j \leq k$, and $x_{i_1}x_{i_2}\cdots x_{i_k} = y_{i_1}y_{i_2}\cdots y_{i_k}$
- **PCP** can be used to show the undecidability of Ambiguity in CFGs, the undecidability of two CFGs producing overlapping words and the undecidability of non-emptiness problem for CSGs.

- The undecidability of a CFG producing Σ^* is based on traces of terminating computations in Turing machines. Really, it's based on the complements of such traces. Traces can also show the complexity of the quotient of CFLs and of power problems for CFLs.