THEORY OF COMPUTATION (AUBER) EXAM CRIB SHEET

Regular Languages and FA

A **language** is a set of strings over a finite alphabet Σ. All languages are finite or countably infinite.

The set of all languages over Σ* is uncountably infinite.

$\forall L, L\varnothing = \varnothing L = \varnothing$ (analogue to multiplication by 0)

$\forall L, L\{\varepsilon\} = \{\varepsilon\}L = L$ (multiplication by 1 analogue)

$\mathbf{L}^n = L \cdot L \cdot L \cdot \ldots \cdot L$ (n times)

$\mathbf{L}^0 = \{\varepsilon\}$

$\mathbf{L}^1 = L$

$\mathbf{L}^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots \; \forall L, \varepsilon \in L^*$

$\mathbf{L}^+ = L^1 \cup L^2 \cup L^3 \cup \ldots$

$\varnothing^* = \varnothing^0 = \{\varepsilon\}$

String Reversal (w^R) inductive definition:
if $|w| = 0$, $w = w^R = \varepsilon$.
If $|w| \geq 1$, then $\exists a \in \Sigma$, $w = ua$ and $w^R = au^R$.
A **palindrome** is a string w, such that $w = w^R$.
Language Reversal: $L^R = \{w^R : w \in L\}$.

The "pure" **regular expressions** are finite strings over the alphabet $\Sigma \cup \{"(", ")", \varnothing, \cup, *\}$:

- $\varnothing$ and each particular member of Σ is a regular expression.
- If α, β are regular expressions, then so are αβ, α∪β, α*, and (α).

Operator precedence (from highest to lowest): ( ), *, concatenation, ∩, ∪.

A **regular grammar** has a set of non-terminals, terminals, rules, and a starting non-terminal. {Terminals} = Σ.

The regular grammar rules must be one of the following four forms:

⟨non-terminal⟩ → ⟨terminal⟩ ⟨non-terminal⟩ | ⟨terminal⟩ | ⟨non-terminal⟩ | ε.

A **Deterministic Finite Automata DFA** is
$M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states; Σ is an alphabet; $q_0 \in Q$ is the initial state; $F \subseteq Q$ is the set of final states; and δ is the transition *function* from $(Q \times \Sigma)$ to Q.

A **Nondeterministic Finite Automata NFA** is $M = (Q, \Sigma, \Delta, q_0, F)$, where, Q is a finite set of states, Σ is an alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and Δ is the transition *relation*, which is a finite subset of $(Q \times (\Sigma \cup \{\varepsilon\})) \times Q$.

A **language is regular** if and only if it can be described by a regular expression, regular grammar, NFA, or DFA or made up using regular language closure properties. All finite languages are regular.

Regular languages are closed under: union, concatenation, Kleene closure *, complementation, intersection, and reversal.

Subset construction (State set method) for converting an NFA to a DFA: States in the DFA (M') will correspond to sets of states in the NFA (M). Σ'=Σ. Compute E(q) for all q in Q (the set of states reachable on ε-transitions).
s' = E(s). δ' (Q', a) = ∪{E(p) : p ∈ Q
and (q, a, p) ∈ Δ
for some q ∈ Q'}. Q' ⊆ 2^Q that appear in δ'.
F' = {Q' ∈ Q' : Q' ∩ F ≠ ∅}.

Creating a regular expression from an NFA (State Elimination): Convert transitions to simple regular expressions. Add new start and final state with appropriate ε-transitions. Remove all other states, one state at a time reconstructing regular expression paths through the state being removed. For each pair of incoming and outgoing paths via the

state being removed (from x to y, labeled α and β, respectively), add a transition from x to y with the regular expression $\alpha\gamma^*\beta$ (where γ is the regular expression on the self-loop on the state being removed, or ε). Union together the regular expressions for parallel transitions. When only the start and final state remain with one transition, that regular expression is the answer.

Pumping Lemma for Regular Languages: Let L be an infinite regular language. Then $\exists$ m $\geq$ 1, such that for any string w $\in$ L, where |w| $\geq$ m, $\exists$ x, y, z, such that w = xyz and |xy| $\leq$ m, and |y| $\geq$ 1, and $\forall$ i $\geq$ 0, $\mathbf{xy^iz}$ is in L.

The pumping lemma is used to prove languages are not regular by contradiction.

Strategy: After you assume the language is regular, you may use closure properties to make your proof easier/possible by constructing a related language that you can more easily prove is not regular. Choose w that depends only on m. Show that for each possible partitioning (there may be several cases) that you can find a i that contradicts the pumping theorem.

A **decision procedure** is an algorithm that answers a question (usually "yes" or "no") and terminates. There are decision procedures for deciding whether: L(M) = $\varnothing$, L(M) = Σ^*, L(M) is infinite, L(M) is finite, w $\in$L(M), L(M_1) = L(M_2), L(M_1) $\subseteq$ L(M_2), etc. for NFAs M, M_1, M_2.

Equivalence classes of a language: Based on the equivalence relation x $\approx_L$ y if, $\forall$z $\in \Sigma^*$, xz $\in$ L iff yz $\in$ L. If a z exists that violates the test, it is called a *distinguisher*. $\approx_L$ is an equivalence class that partitions Σ^*.

The **Myhill-Nerode Theorem** states that if L is regular that there is a DFA that accepts L that has precisely $\approx_L$ states. Also, a language is regular iff $\approx_L$ is finite.

Algorithm for minimizing a DFA (State Reduction by Set Partitioning): Remove any unreachable states as a first step. Break apart the equivalence classes of states at each iteration based on their distinguishability against the equivalence classes of the previous iteration.

Initially, $\equiv_0$ = {[F], [Q - F]}. For any two states p and q $\in$ Q and any n $\geq$ 1, q $\equiv_n$ p iff:
q $\equiv_{n-1}$ p, AND for all a $\in \Sigma$, δ(p, a) $\equiv_{n-1}$ δ(q, a). Iterate until $\equiv_{n-1}$ equals $\equiv_n$. The result, $\equiv$, is this final equivalence relation. Join together states from the equivalence classes to create the minimal state DFA.

Context Free Languages and PDA

A **context-free grammar (CFG)** is a quadruple G = (V, Σ, S, R), where:

- **V** is the rule alphabet, which contains **nonterminals** (symbols that are used in the grammar but that do not appear in strings in the language) and **terminals**,
- **Σ** (the set of **terminals**) is a subset of V,
- **S** (the **start symbol**) is an element of V - Σ.
- **R** (the set of **rules** (productions)) is a finite subset of (V - Σ) $\times$ V^*,

Any sequence of the form: $w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \ldots \Rightarrow_G w_n$ e.g., (S) $\Rightarrow$ (SS) $\Rightarrow$ ((S)S)

is called a **derivation in G**. Each w_i is called a **sentential** form where in each step, one nonterminal is replaced with the RHS (right hand side) of one of its associated rules. The language generated by G is $\{w \in \Sigma^* : S \Rightarrow_G^* w\}$.

A **language L is context free** if L = L(G) for some context -free grammar G. All regular languages are context free.

In a **leftmost derivation**, the leftmost nonterminal is always the one that is replaced.
In a **rightmost derivation**, the rightmost nonterminal is always the one that is replaced.

A **parse tree** captures the essential structure of a derivation, but does not contain complete information about the time ordering of rule applications. Several derivations can therefore be associated with the same parse tree. Each parse tree has a unique leftmost derivation and a unique rightmost derivation. The **root** of the parse tree is as a rule S.

The **leaves** of a parse tree are terminals or ε's. The **yield** of a parse tree is the concatenation of the leaves.

A context free **grammar is ambiguous** if it has multiple parse trees for some string in the language. A context free **language is inherently ambiguous** if there is no unambiguous grammar for the language.

A **pushdown automata (PDA) is** a sextuple $M = (Q, \Sigma, \Gamma, \Delta, s, F)$, , where:

• **Q** is a finite set of states; • **Σ** is the input alphabet; • **Γ** is the stack alphabet;

• **s** $\in$ Q is the initial state; • **F** $\subseteq$ Q is the set of final states, and

• **Δ** is the transition relation. It is a finite subset of $(Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*) \times (Q \times \Gamma^*)$.

The first grouping describes the machine computer that allows the rule to follow (state, optional input symbol, and symbols at the top of the stack). The second grouping describes what to do (go to a new state and replace the top of stack.)

For the **machine to accept** a string, some computation must leave the machine in a final state, consume all the input. **L(M)** is the set of all such strings. Other computations on that string which fail are ignored.

A PDA is **deterministic** if, for each input and state, there is at most one possible transition to follow in any configuration. Determinism implies uniquely defined machine behavior.

Nondeterministic PDAs accept the class of context free languages.

There are **algorithms to convert from a PDA to a CFG and from a CFG to a PDA**.

"Top-down parse" conversion: Given G = (V, Σ, R, S), construct M such that L(M) = L(G).

M = ({p, q}, Σ, V, Δ, p, {q}), where Δ contains:

• ((p, ε, ε), (q, S)) push the start symbol on the stack as the "goal"

• ((q, ε, A), (q, x)) for each rule A $\rightarrow$ x in R replace left hand side with right hand side

• ((q, a, a), (q, ε)) for each a $\in$ Σ read an input character and pop it from the stack

The resulting machine can execute a leftmost derivation of an input string and it will usually be nondeterministic.

"Bottom-up parse" conversion: Given G = (V, Σ, R, S), construct M such that L(M) = L(G).

M = ({p, q}, Σ, V, Δ, p, {q}), where Δ contains:

• ((p, ε, xR), (p, A)) for each rule A $\rightarrow$ x in R **reduce** using the rule A $\rightarrow$ x

• ((p, a, ε), (p, a)) for each a $\in$ Σ **shift** input symbol 'a' onto the stack

• ((p, ε, S), (q, ε)) **accept** by popping the start symbol off the stack

The resulting machine can execute a rightmost derivation (backwards) of an input string and it will usually be nondeterministic.

CFLs are closed under union, concatenation, Kleene closure, and intersection with a regular language.

CFL Pumping Lemma: For every infinite CFL, L, there is an integer m, such that any string $w \in L$ where $|w| \geq m$ can be rewritten as w = uvxyz in such a way that: $|vy| \geq 1$; $|vxy| \leq m$; and $\forall i \geq 0$, $\mathbf{uv^ixy^iz \in L}$.

There are **decision procedures** that will determine for a CFG, G, whether $w \in L(G)$, whether $L(G) = \varnothing$, or whether L(G) is infinite.

Unrestricted Languages and Turing Machines

A **Turing machine** is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, \#, F)$:

- Q is a finite set of internal states;
- Σ is the input alphabet;
- Γ is a tape alphabet;
- δ is the transition function;
- $\# \in \Gamma$ is a special blank symbol
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final states.

Let Σ_0 be the input alphabet. Then M **decides** a language $L \subseteq \Sigma 0^*$ iff for any string $w \in \Sigma_0^*$ it is true that: if $w \in L$ then M accepts w (halts in the "Y" state), and if $w \notin L$ then M rejects w (halts in the "N" state).

A language L is **Turing decidable (recursive)** if there is a Turing machine M that **decides** it.

The class of **recursive languages is closed** under union, intersection, concatenation, Kleene closure, complement, and reversal.

We say that M **lexicographically enumerates** L if M enumerates the elements of L in lexicographic order. A language L is **lexicographically Turing-enumerable** iff there is a Turing machine that lexicographically enumerates it.

Theorem: A language is recursive iff it is lexicographically Turing-enumerable.

A **function is recursive** iff there is a TM that computes it and halts for every input in the domain.

Let Σ_0 be the input alphabet to a Turing machine M. Let $L \subseteq \Sigma_0^*$. M **accepts (semidecides)** L iff for any string $w \in \Sigma_0^*$, $w \in L \Rightarrow$ M halts on input w, and $w \notin L \Rightarrow$ M does not halt on input w (M(w) =).

L is **Turing recognizable (recursively enumerable)** iff there is a Turing machine that **accepts** (semidecides) it.

The class of **recursively enumerable languages is closed** under union, intersection, concatenation, Kleene closure, and reversal, but *NOT* complement.

Theorem: A language is recursive iff both it and its complement are recursively enumerable.

Theorem: If a language is RE and not recursive, then its complement is not RE.

A language is **Turing-enumerable** iff there is a Turing machine that enumerates it.

A language is recursively enumerable iff it is Turing-enumerable.

A **function is partially recursive** iff there is a TM that computes it and halts for every input in its domain, but the domain is restricted to a recursively enumerable set.

An **unrestricted, or Type 0, or phrase structure grammar G** is a quadruple (V, Σ, R, S), where

- V is an alphabet;
- Σ (the set of terminals) is a subset of V;
- R (the set of rules) is a finite subset of $V^*\cdot(V-\Sigma)\cdot V^* \times V^*$ (left hand sides must contain one nonterminal); and
- S (the start symbol) is an element of $V - \Sigma$.

We define derivations just as we did for context-free grammars. The language generated by G is $\{w \in \Sigma^* : S \Rightarrow_G^* w\}$.

There is no notion of a derivation tree or rightmost/leftmost derivation for unrestricted grammars.

Theorem: A language is generated by an unrestricted grammar if and only if it is recursively enumerable (i.e., it is accepted (semidecided) by some Turing machine M).

We say that **G computes f** if, for all w, v $\in\Sigma^*$, $SwS \Rightarrow_G^* v$ iff v = f(w).

A function f is called **grammatically computable** iff there is a grammar G that computes it.

Theorem: A function f is recursive iff it is grammatically computable.

The **universal TM** (U), is a TM interpreter. L(U) = H = {⟨M, w⟩: TM M halts on input w}.

H is RE, but not recursive. ¬H is not RE.

Let $L_1, L_2 \subseteq \Sigma^*$ be languages. A **reduction** from L_1 to L_2 is a recursive function ρ: $\Sigma^* \rightarrow \Sigma^*$ such that $x \in L_1$ iff $\rho(x) \in L_2$.

Theorem: If L_1 is not recursive and there is a reduction from L_1 to L_2, then L_2 is not recursive.

Theorem: If L_1 is not RE and there is a reduction from L_1 to L_2, then L_2 is not RE.

Proof technique for showing a language is undecidable: Choose some known undecidable language L_1. (H will usually work for this.) Assume that L_2 (your language) is decidable. Construct a reduction from L_1 to L_2. You have just shown that L_1 is decidable. Contradiction. Therefore L_2 is undecidable.

Rice's Theorem: No nontrivial property of the recursively enumerable languages is decidable.

Nontrivial means that for all RE languages the property is not always true or always false.

Primitive recursive function template, by which a function can be defined recursively through

$f(x, 0) = g_1(x)$

$f(x, y+1) = h(g_2(x, y), f(x, y))$

from defined functions g_1, g_2 and h.