CS 3120 Quizzes 1-5 (including 1-4 retakes)

This packet contains the quizzes for each module, to be taken during the final quiz day. This **cover sheet** is here to provide instructions, and to cover the questions until the quiz begins. **do not open this packet** until your proctor instructs you to do so.

You will have the entire class period to complete these quizzes. Each module quiz is two pages (front and back of one sheet of paper) worth of questions. Make sure to **write your name and computing id at the top of each individual quiz**. The quizzes are ordered in ascending order.

This quiz is CLOSED text book, closed-notes, closed-calculator, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge below.

In theory, there is no difference between theory and practice.
But, in practice, there is.

THIS COVER SHEET WILL NOT BE SUBMITTED. DO NOT PUT WORK YOU WANT GRADED ON THIS PAGE

Quiz - Module 1: Proofs, Computers, and Cardinality

Name ______________________________

1. [10 points] Answer the following True/False questions.

All *computational models* read over their input exactly once, and then halt	**True**	**False**
All *computation models* can be described as *accepting* or *rejecting* their input	**True**	**False**
The *computational models* we have studied can accept strings of *arbitrary length* (i.e., the language of accepting strings can be infinite)	**True**	**False**
All *finite sets* are *countable*	**True**	**False**
Suppose that for sets A and B, $\lvert A\rvert < \lvert B\rvert < \lvert\mathbb{N}\rvert$. It is possible that there is a *bijection* between A and $\mathbb{N}$	**True**	**False**
For any *finite set* A, $\lvert\mathbb{N} \cup A\rvert \leq \lvert\mathbb{N}\rvert$	**True**	**False**
The *intersection* of two *uncountably infinite* sets can never be *finite*	**True**	**False**
According to the *pigeonhole principle*, if $\lvert A\rvert < \lvert B\rvert$, then any function from A to B cannot be *injective*	**True**	**False**
The *power set* of any set is always larger than the original set	**True**	**False**
The set of all *real numbers* strictly between 0 and 1 is *uncountably infinite*	**True**	**False**

2. [4 points] For each of the following sets, circle whether the cardinality of the set is *Finite*, *Countably Infinite*, or *Uncountably Infinite*

The set of all unique *DFA*s such that $\lvert Q\rvert = n$ and $\lvert\delta\rvert = m$ for some $n, m \in \mathbb{N}$	**Finite**	**Countably Infinite**	**Uncountably Infinite**
The set of all *DFA*s of any size	**Finite**	**Countably Infinite**	**Uncountably Infinite**
$\{w \mid w \in ab^*ab^*\}$	**Finite**	**Countably Infinite**	**Uncountably Infinite**
The set of functions $\{f \mid f : \Sigma^* \rightarrow \Sigma^*\}$	**Finite**	**Countably Infinite**	**Uncountably Infinite**

In class, we saw a proof by *diagonalization* that the set of infinite bitstrings ($\{0, 1\}^\infty$) is *uncountable*. On this page, you will reconstruct this proof.

3. [3 points] We begin by stating that *for the sake of contradiction, assume the set* $\{0, 1\}^\infty$ *is countable*. State what artifact *must exist* given this assumption, and draw out a table showing an example of that artifact.

4. [3 points] Now, identify an element that is not present in the artifact you listed in the previous question. Explain how to find this element AND why it is not present.

NOTHING BELOW THIS POINT WILL BE GRADED

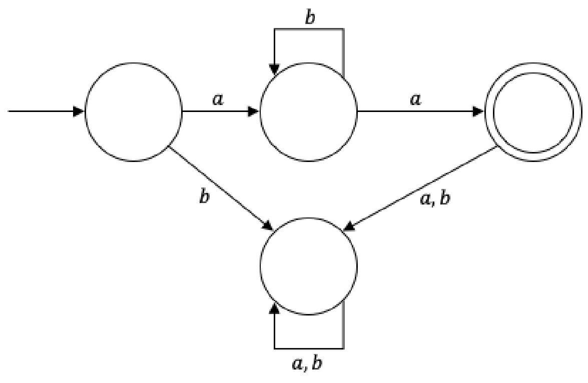
Quiz - Module 2: Regular Languages

Name ______________________________

1. [8 points] Answer the following True/False questions.

A *DFA* always has some transition for every combination of *state* and *input character* that is possible	**True**	**False**
A *DFA* can have an *infinite* number of states, as long as you specify how each is reached in some systematic way	**True**	**False**
An *NFA* can be in an *infinite* number of states at once	**True**	**False**
When an *NFA* uses non-determinism to be in multiple states, each unique branch of computation reads in the remaining input characters one at a time	**True**	**False**
*DFA*s and *NFA*s are equivalent in their expressive power	**True**	**False**
Regular languages are closed under *complement* and *intersection*	**True**	**False**
All strings in a regular language A of least length p (pumping length) have a substring in the *LAST* p characters that can be pumped indefinitely	**True**	**False**
The pumping lemma states (among other things) that all strings in a regular language A can be pumped	**True**	**False**

2. [2 points] Give a succinct regular expression for the dfa below.



3. [3 points] Draw a DFA with four states or less for the following language:
$\{w \in \{a, b, c\}^* \mid$ the sum of the number of a's in w and the number of b's in w is less than 3$\}$.

For this question, you will show you understand the proof for converting an arbitrary *regular expression* into an equivalent *NFA* by doing so on one example. Your conversion must be done in *EXACTLY* the way we did to prove the closure properties in class.

All of the questions on this page involve the following regular expression: $(a \cup b)^*$

4. [3 points] First, draw the base case *NFA*s for the simple expressions a and b (these are two separate machines).

5. [2 points] Now, draw the combined *NFA* for $a \cup b$.

6. [2 points] Lastly, extend your drawings above to produce the final *NFA* for $(a \cup b)^*$.

Quiz - Module 3: Context-Free Grammars

Name ______________________________

1. [8 points] Answer the following True/False questions.

Non-determinism allows *PDA*s to split into computational branches. Each branch has its own stack (independent of the stack for other branches)	**True**	**False**
It is possible to convert any *PDA* into an equivalent one that always *accepts* with an empty stack only	**True**	**False**
If a *grammar G* produce any string a that has repeated variables *anywhere* in its derivation tree, then the language of G is *infinite*	**True**	**False**
For *context-free grammars*, only single *variables* can appear on the left side of a substituion rule	**True**	**False**
There are some *context-free grammars* for which no equivalent *pushdown automata* exists	**True**	**False**
For the language 0^n1^n, the string 0^p1^p cannot be pumped because only 0s can be chosen (by the pumping lemma constraints) and doing so increases the number of 0s, but not the number of 1s	**True**	**False**
Every *DFA* can be converted into an equivalent *PDA* by not utilizing the stack	**True**	**False**
Pushdown Automata always have a finite number of *states*	**True**	**False**

2. [3 points] Draw a *pushdown automata* for the language of palindromes: $ww^R \mid \Sigma \in \{0,1\}^*$

3. [3 points] Consider the *context-free* language $wtw^R,\ |w| = |t|$. Provide a working context-free grammar for this language.

For these two problems, you will show some *closure properties* of *context-free languages* by showing how to do constructions with *grammars*, as opposed to using the *machines (PDAs)*

4. [3 points] First suppose I have a context-free grammar A and another context-free grammar B. These grammars provide substitution rules that generate strings for languages A_L and B_L respectively. Describe how to combine these grammars to produce a new grammar C for the language $C_L = A_L \cup B_L$.

5. [3 points] Now, do the same but for the *star operator*. Describe how to take one grammar A for language A_L and produce a new grammar C for language $C_L = A_L^*$.

NOTHING BELOW THIS POINT WILL BE GRADED

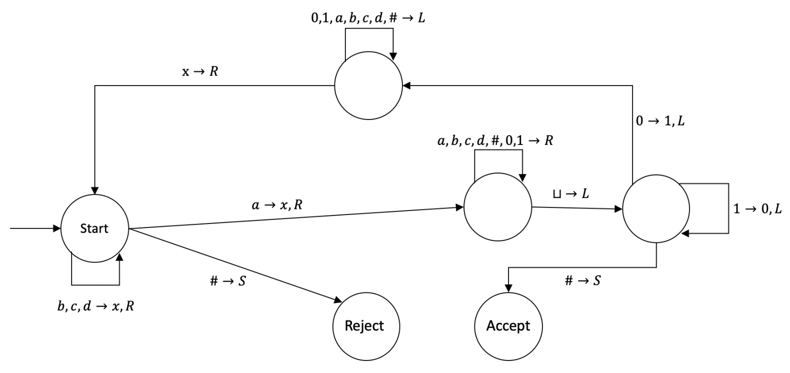
Quiz - Module 4: Turing Machines

Name ______________________________

1. [10 points] Answer the following True/False questions.

Turing Machines can store data on the tape, but they cannot overwrite the *input* which is present on that tape	**True**	**False**
We can implement arithmetic operations, like *addition*, with *Turing Machines*	**True**	**False**
A *recognizer* will always halt and say *No* when the correct answer is *No*	**True**	**False**
Some *NTM*s cannot be simulated with an equivalent *DTM* because the number of non-deterministic branches in the original machine is too high	**True**	**False**
One of the strongest features of the *Turing Machine* is its ability to generate new states in its control, effectively allowing the number of states to be *infinite*	**True**	**False**
If a problem A is *recognizable* but not *co-recognizable*, then A cannot be *decidable*	**True**	**False**
If we were to find a *working decider* for A_{TM}, then we would be able to use it to solve the halting problem ($Halt_{TM}$)	**True**	**False**
Decidable Problems are ones for which a *DTM* exists that always solves the problem $\Theta(n^c) \mid c \in \mathbb{N}$ time	**True**	**False**
A *NTM* can be configured to accept if exactly *three branches of computation accept*	**True**	**False**
When simulating an *NTM* with a *DTM* we use three tapes. The third contains one integer, the depth to which to explore the *NTM*s computation tree	**True**	**False**

2. [4 points] Take a look at the Turing Machine below. In the box on the write, describe what this Turing Machine does. Make sure to address *what strings will be accepted* as well as *what the machine writes to the tape. HINT: Input comes in the form* $\{a, b, c, d\}^*\#000 \sqcup \sqcup \ldots$ *The string of letters is either accepted or rejected and the answer is written to the tape where the three 0's are located.*



The problems on this page all involve the same problem. Consider the function $TM_{R3}(M')$, which returns *True* if and only if Turing Machine M' ever *moves its head to the right three times in a row*.

3. [6 points] Show that TM_{R3} is undecidable by providing a reduction from $Halt_{TM}(M, w)$. We will split this reduction into two parts:

First, given input to $Halt_{TM}$ (i.e., a machine M and string w), describe (write the pseudocode for) how to construct a new machine M' that maps outputs from $Halt_{TM}$ to outputs of TM_{R3}

Now, assuming the construction above works, provide the pseudocode steps for a decider to the *undecidable problem* $Halt_{TM}$. You may invoke your construction from the previous problem by simply stating *Construct* M'. Assume the construction works even if your answer to the previous problem isn't correct.

We provide the beginning of the proof here, which leads into the description of your decider: *Assume that* TM_{R3} *is decidable, thus a decider* M_{R3} *exists that decides it. We can construct a working decider for* $Halt_{tm}$ *as follows:*

NOTHING BELOW THIS POINT WILL BE GRADED

Quiz - Module 5: Complexity Theory

Name ______________________________

1. [8 points] Answer the following True/False questions.

All problems in P can be solved with an *NTM* is polynomial time	**True**	**False**
It is still unknown today if the *3-SAT* problem has a working decider because the problem is *NP-Complete*	**True**	**False**
NP-Hard problems are strictly harder than every problem in *NP*	**True**	**False**
The *Clique Problem* can be solved in Polynomial Time using an *NTM*	**True**	**False**
If a *DTM* can verify a problem in polynomial time, then an *NTM* can solve that same problem in polynomial time	**True**	**False**
The *Cook-Levin Theorem* involved showing *SAT* was NP-Complete by showing a reduction from *3-SAT*	**True**	**False**
For all *NP-Complete* problems A and B, $A \leq_p B$ and $B \leq_p A$	**True**	**False**
All *NP-Hard* problems are also *NP-Complete*	**True**	**False**

2. [2 points] In class, we proved the following claim: *If there exists a DTM verifier for a language A that runs in polynomial time ($\Theta(n^c) \mid c \in \mathbb{N}$), then there exists an NTM solver for the language A that also runs in polynomial time.* Briefly, describe the algorithm that solved A.

3. [2 points] Suppose you are given a problem C and it is known that C is *harder than NP*. What can you say about an algorithm that attempts to *Verify* inputs to C? Briefly explain your answer.

For this question, you will prove that a simple problem is *NP-Complete*. For the *House* problem, consider a list of houses H of families that all live in the same small city (let $|H| = n$). You are going to send out an advertisement to every house in H, but you want it to seem as if every house received a unique advertisement with a different style, art, color scheme, etc. Obviously, you could design and print n unique advertisements, but that is too expensive. You want to design the minimum number of unique advertisements such that no two houses in H will have their families *compare ads* and notice that they received the same one. To do, this you have also acquired a list of pairs $F = \{(i, j) \mid i, j \in H\}$ that tells you if each pair of households is friendly (thus they might show each other the ads and compare) or not friendly (it is safe to send these houses the same ad). Given H and F, what is the minimum number of unique ads you need to design in order to gaurantee that there is a way to give them out and have no two friendly houses in H receive the same ad?

4. [2 points] First, reframe the *House* problem as a decision problem.

5. [2 points] Next, describe pseudocode for a *verification* algorithm for the *House* problem.

6. [4 points] Now, we will reduce *3-Coloring* to the *House* problem. Given a graph G, describe how to convert this input into an equivalent instance of the *House* problem. Make sure to use the *decision problem* version of the *House* problem.