James Dollard
Theory of Computation
Final Test - Part 2

1. (30 points) Consider the problem of trying to determine if two DFAs have exactly the same language. Formulate this problem as a language and then state whether or not this language is decidable. If it is decidable, then describe the behavior of the decider that decides it. If it is not decidable, then provide a proof for why not.

EQ_{DFA} = {<D_1,D_2>: D_1 and D_2 are DFAs with the same language}

The language of a DFA is regular, so it can be described by a regular expression. A decider would first check if each DFA is properly encoded. Then, it would convert both DFAs to a regular expression, which can be done in finite time because a DFA has a finite number of states. It would check if these regular expressions are equivalent by converting them into a standard form. This takes a finite amount of computation to determine. If the expressions are equivalent, the decider accepts. If not, it rejects. Since in all cases the decider terminates, EQ_{DFA} is decidable.

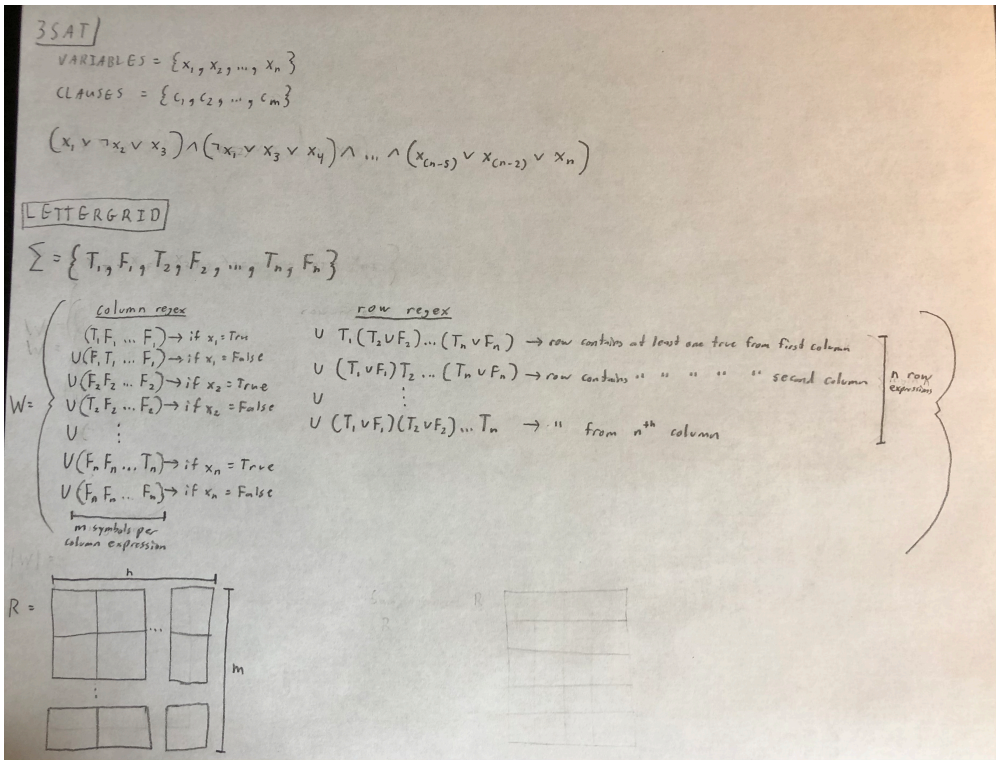
2. (30 points) Our usual definition of TMs is that they use a pencil to write symbols, so the symbols can be erased and rewritten as many times as we like. Suppose we have a new kind of TM with a single tape that uses a pen instead, so once a symbol is written onto a blank cell of the tape, that cell can never be changed again. Show that this pen-based TM is equivalent in computational power to a normal, pencil-based TM.

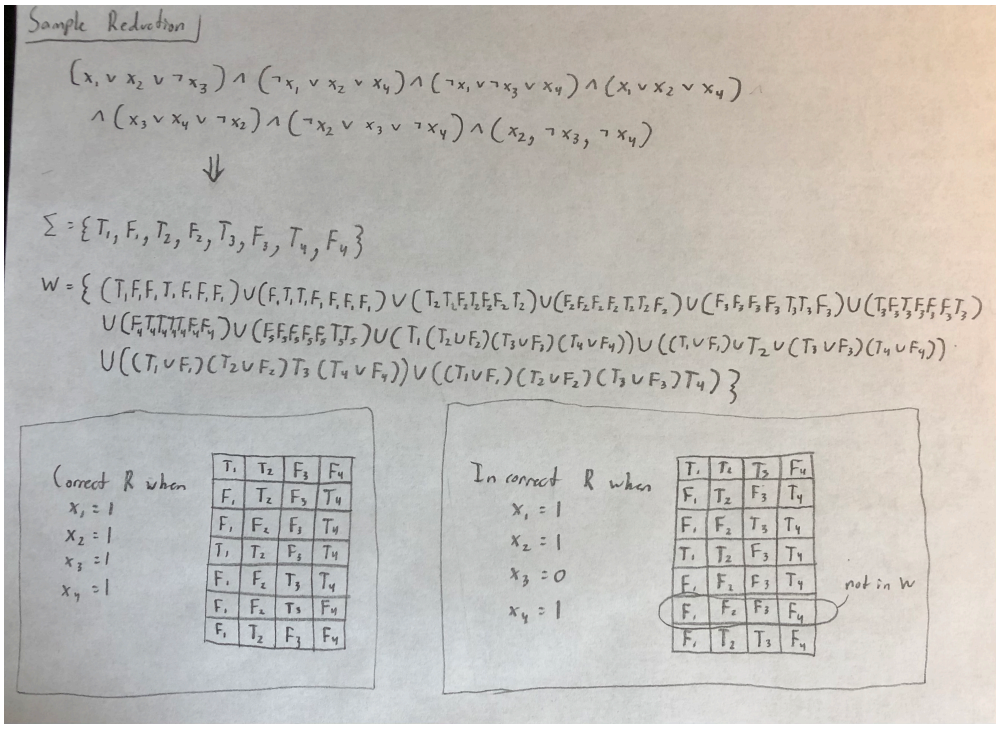
Let M and N be deterministic Turing machines where the tape alphabet of N doesn't contain the blank symbol. Let M and N be restricted to using one tape. Finally, let the special symbols * and O be elements of N's tape alphabet. On receiving the input string, N writes * to the left and right of it. Suppose that M and N perform computation on the same string, w, and that this computation requires M to occasionally write blank symbols to its tape. N performs the same transitions as M, except when M writes a blank symbol, N writes O and copies the contents of the tape between the two * to the empty space after the right-most *. When it encounters O, it knows to leave that cell blank. When done copying, it adds a new * to the right of the copy and continues following the computation of M. Now, suppose that this computation requires M to write a symbol where N encounters *. N simply performs the same copy operation, this time leaving a blank cell between the copy and the equivalent * it encountered. By forever restricting the tape before the left * and taking advantage of the infinite tape after the right *, N is computationally equivalent in power to M.

3. (40 points)
Define LETTERGRID as the problem of putting letters from an alphabet, Σ, into a rectangular grid, R, of squares so that every row (read left to right) and every column (read top to bottom)is a word from a list of legal words, W.

Prove that LETTERGRID is NP-complete by defining a polynomial-time reduction from 3SAT (satisfiability of boolean expressions with clauses of size 3).

Note: Sorry that I couldn't find the more simple reduction… unless I overlooked something, I think this one still works. It's a long proof, so I'm attaching the following for reference. I'd appreciate knowing the better way of doing this if you get a chance to email it to me!





Let the number of clauses (c_1, c_2, … , c_m) in 3SAT be m and the number of variables (x_1, x_2, … , x_n) be n.

To convert 3SAT to LETTERGRID, first construct Σ, which is an alphabet with two symbols per variable (i.e. 2n symbols in total, so O(n) steps). Each variable will be associated with two symbols in the alphabet, which represent true and false, and those two symbols will only be associated with that variable. Next, construct the language W. This language will be described as a union of regular expressions. For each variable, create two regular expressions of the form $r_1r_2…r_m$ where each r is a symbol in Σ. For both expressions, set all r to the false symbol associated with that variable. For the first word, if the variable is in clause c_i for any index i, if the variable is not negated, set r_i to the true symbol associated with that variable. For the second word, if the variable is in clause c_i and the variable is negated, set r_i to the same true symbol. This process is O(n*m*2), so it is done in polynomial time. Now, create n regular expressions where the j^{th} regular expression is in the form $(x_{1T} \cup x_{1F})(x_{2T} \cup x_{2F})…x_{jT}…(x_{nF} \cup x_{nF})$ such that x_{jT} is the true symbol associated with the j^{th} variable. This is completed in $O(n^2)$. Finally, describe W as the union of all of these regular expressions, and let the grid of squares, R, have m rows and n columns. The sum of these steps is within polynomial time.

In the special case that 3SAT has the same number of variables as it does clauses, this conversion will be slightly different. R will have m+1 rows and m columns. And for each variable, the expressions of the form $r_1r_2…r_m$ are now of the form $r_1r_2…r_{m+1}$ such that the final r contains the special symbol '$', and $\$^m$ is now part of the union of expressions in W. Everything else stays the same, including the time complexity. The purpose of this is to force rows and columns to have different categories of strings in a correct solution.

LETTERGRID has a solution if 3SAT has a solution.
Assume that 3SAT has a solution. This means that the set of variables x_1, x_2, … , x_n is some combination of true and false such that each clause c_1, c_2, … , c_m contains at least one true boolean value. In the conversion from 3SAT to LETTERGRID, the number of columns are equivalent to the number of variables and the same for rows to clauses. The length of the rows always differ from the length of the columns so that each row must be of the form $(x_{1T} \cup x_{1F})(x_{2T} \cup x_{2F})…x_{jT}…(x_{nF} \cup x_{nF})$. This forces the first column to only contain symbols related to the first variable, the second column the second variable, and so on, for LETTERGRID to have a correct answer. It also forces each row to contain at least one 'true' variable. The columns must be of the form $r_1r_2…r_m$. Each column, forced to stick to symbols related to the same variable, can only choose from two words in W of the form $r_1r_2…r_m$. The first word describes, for each symbol index, how that variable being true affects the boolean truth value in each clause of the same index. The second word describes how the variable being false would affect it. Therefore, if we know that 3SAT has a solution, each column can adopt a string in W detailing the boolean effect on each clause of its associated variable being true or false, such that each row contains at least one truth value. Each row now reads a string that's also in W, so LETTERGRID has a solution.

LETTERGRID has a solution only if 3SAT has a solution.
We can construct the 3SAT problem by first creating m empty clauses separated by a boolean “and”. Next, we iterate through each word in W of length m. There are two words of this length associated with each variable. For the first word, if the i^{th} symbol is true, add the variable to the clause of the same index. For the second word, if the i^{th} symbol is true, add the variable to the clause of the same index and negate it. Repeating for each variable and separating the contents of the clause with a boolean “or” will add three variables per clause, making a complete 3SAT problem. Assume that LETTERGRID has a solution. This implies that columns only contain the boolean symbols of the variable of the same index, where the symbol in each row describes the boolean value in the clause of the row’s index, and each row contains at least one true boolean value. Since each row contains at least one true value, we know that each clause in 3SAT contains at least one true value, so 3SAT has a solution.

Therefore, LETTERGRID has a solution if and only if 3SAT has a solution.