CSci 435: Formal Languages and Automata

Instructor: Dr. M. E. Kim Date: December 8th, 2020

**Due: by the end of the day, 12/15 (Tue.)**

**Final Exam: 120 points + 30 points (optional)**

Name: \_Sam Dressler \_

1. Write your answer below the corresponding question.
2. Do not include the photo image of your handwriting for a text.

Abbreviation:

REG: Regular Language, REC: Recursive Language, RE: Recursive Enumerable Language,

(D)CFL: (Deterministic) Context-Free Language, FA: Finite Automata, TM: Turing Machine,

UG: Unrestricted Grammar, CSG: Context-Sensitive Grammar, etc.

Mark the followings.

Difficulty:

Very Easy: \_\_\_\_\_\_ Easy: \_\_\_\_\_\_ Moderate: \_\_X\_\_ Difficult: \_\_\_\_\_\_ Very Difficulty: \_\_\_\_\_\_

Time:

\_\_\_\_\_~8\_\_\_\_ Hours and \_\_\_\_30\_\_\_\_\_ minutes

\*Q1. [10] For a language *L*(*aa*\*(*ab*+*a*)\*),

1. [5] Find a regular grammar that generates L.

G = {V, T, S, P}

V = {S, A, B}

T = {a, b}

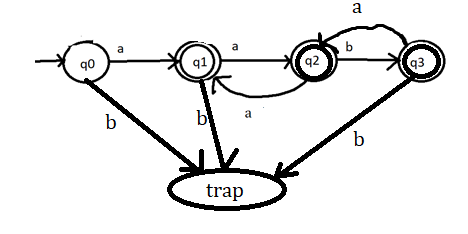
P = {S -> aA,

A -> aA | aB | λ,

B -> bA}

1. [5] Construct the minimal DFA that accepts L.

Hint: the number of states is four.



\*Q2. [10] Prove that L= { *anbn ambm* | *n, m* ≥ 1 } is Context-Free Language but not Linear.

Hint: Construct its CFG and prove its non-linearity using P.L. for linear language.

CFG for L = {an bn am bm | n, m >= 1}

S -> S1S2

S1 -> aS1b | ab

S2 -> aS2b| ab

Assume L is linear and apply pumping lemma

Let w = anbnambm ∊ L where n = 2 and m =2 w = aabbaabb

Split w into 5 parts uvxyz

u = a, v =a, x = bbaa, y = b, z = b

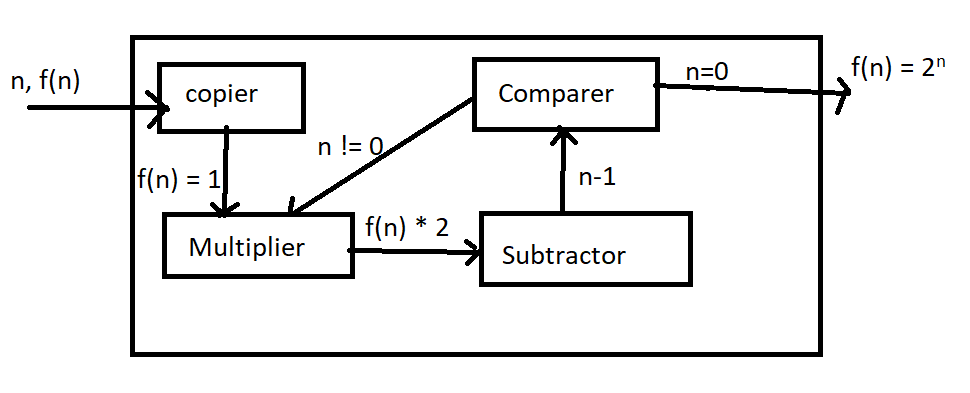
We see that |vy| >= 1 since |vy| = |ab| = 2 which satisfies condition 1.

We also see that |uvyz| <= m where m >0 by taking |uvyz| = |aabb| = 4 which is less than |w| = 8 and >= 0

Now let a variable i = 2 where uvixyiz = uv2xy2z = aaabbaabbb.

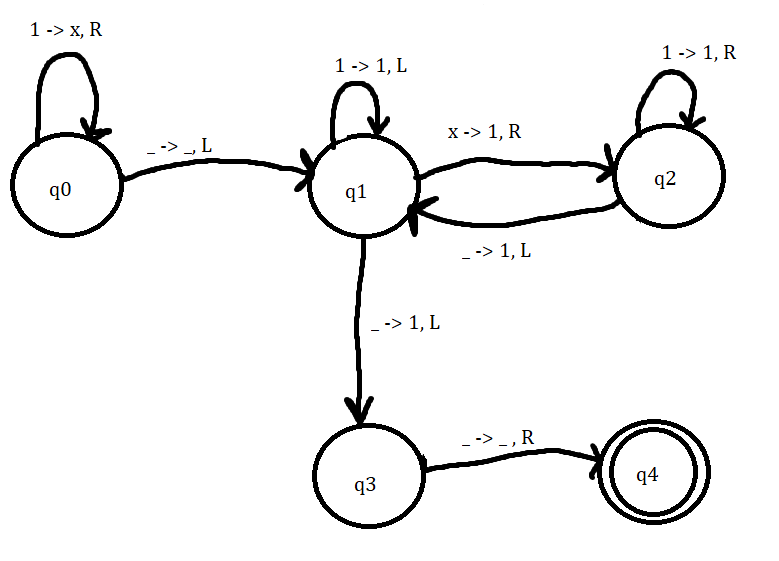
This string is not in L and therefore L is not a linear language.

\*Q3. [10] Using adders, subtracters, comparers, copiers, or multipliers, construct a Turing Machine that compute the functions: *f*(*x*) = 2*n.* Draw its block diagram.



\*Q4. [10] Design a TM that computes the function: *f*(*x*) = 2*x* + 1, where *x* is given in unary notation with 1’s only to TM.

Use the tape symbol Γ = {1, x, €}. Draw its transition function.



\*Q5. [10] Prove that the complement of a Context-Free Language must be Recursive.

Context-Free Languages can be classified as recursive since Context-Free Languages are a subset of the set of all languages that are recursive. Then we see that the complement of a CFL must also be recursive.

\* Q6. [10] Give the language generated by the following unrestricted grammar in a formal expression.

e.g.) { *anbn* | *n* ≥ 1 }

S → AB,

A → *a*Ab,

bB → bbbB,

*a*Ab → *aa*,

B→λ

Let l be a string formed using the unrestricted grammar G with productions listed above.

‘l’ can be derived by the grammar

n = 2, m = 2

l = aaabbbbb

S -> aAbB -> aaAbbB -> aaabB -> aaabB -> aaabbbB -> aaabbbbbB -> aaabbbbb

n =1, m = 0

l = aa

S -> aAbB -> aaB -> aa

A language L can be given:

L = {an+1bn-1b2m | n >= 1, m >= 0}

\*Q7. [10] Find a context-sensitive grammar for L = {*anbna2*n | *n* ≥1} and give a derivation of any string *w*∈ L by your grammar: S ⇒\* *w*.

S -> abaa | aAbaa

Ab -> bA

Aaa -> Bbaaaa

bB -> Bb

aB -> aa | aaA

Let m be a string from the Language L = {anbna2n | n >= 1}

m = aaabbbaaaaaa

S -> aAbaa

* abAaa
* abBbaaaa
* aBbbaaaa
* aaAbbaaaa
* aabAbaaaa
* aabbAaaaa
* aabbBbaaaa
* aabBbbaaaa
* aaBbbbaaaa
* aaabbbaaaa – END

Q8. [10] Let M1 and M2 be arbitrary Turing machines. Show that the problem “L(M1) ⊆ L(M2)" is undecidable.

We can show that L(M1) ⊆ L(M2) is undecidable by first reducing the problem’s input and then proving its correctness.

Given L = {<m1,m2> | L(m1) ⋂ L(m2) = 0}

By using ATM <= m

1. Reduction of Input
   1. Send input (M, w)
   2. First create TM X­1 that ignores its input x, simulates M on w and accepts that if and only if M accepts w
   3. Now create a second TM X­2 that always accepts its input.
   4. Return output (X1, X2)
2. Proof of correctness
   1. If M accepts w, then L(X1) = ∑\* so we see L(M1) ⊆ L(M2)
   2. Else if M does not accept w, then L(N1) = 0, this L(M1) ⊈ L(M2)
   3. Therefore, M accepts w if and only if L(M1) ⊆ L(M2).

Finally, we see that ATM is undecidable, therefore L­1 is undecidable.

\*Q9. [10] Show that for A = {*wi*| 10, 00, 11, 01} and B = {*vi* | 0, 001, 1, 101}, there exists a Post Correspondence solution. 0 1 2 3 0 1 2 3

The existing post correspondence solution: (Note i begins at 0)

1. w1w2 = v1v2

0011 = 0011

1. w2w3 = v2v3

1101 = 1101

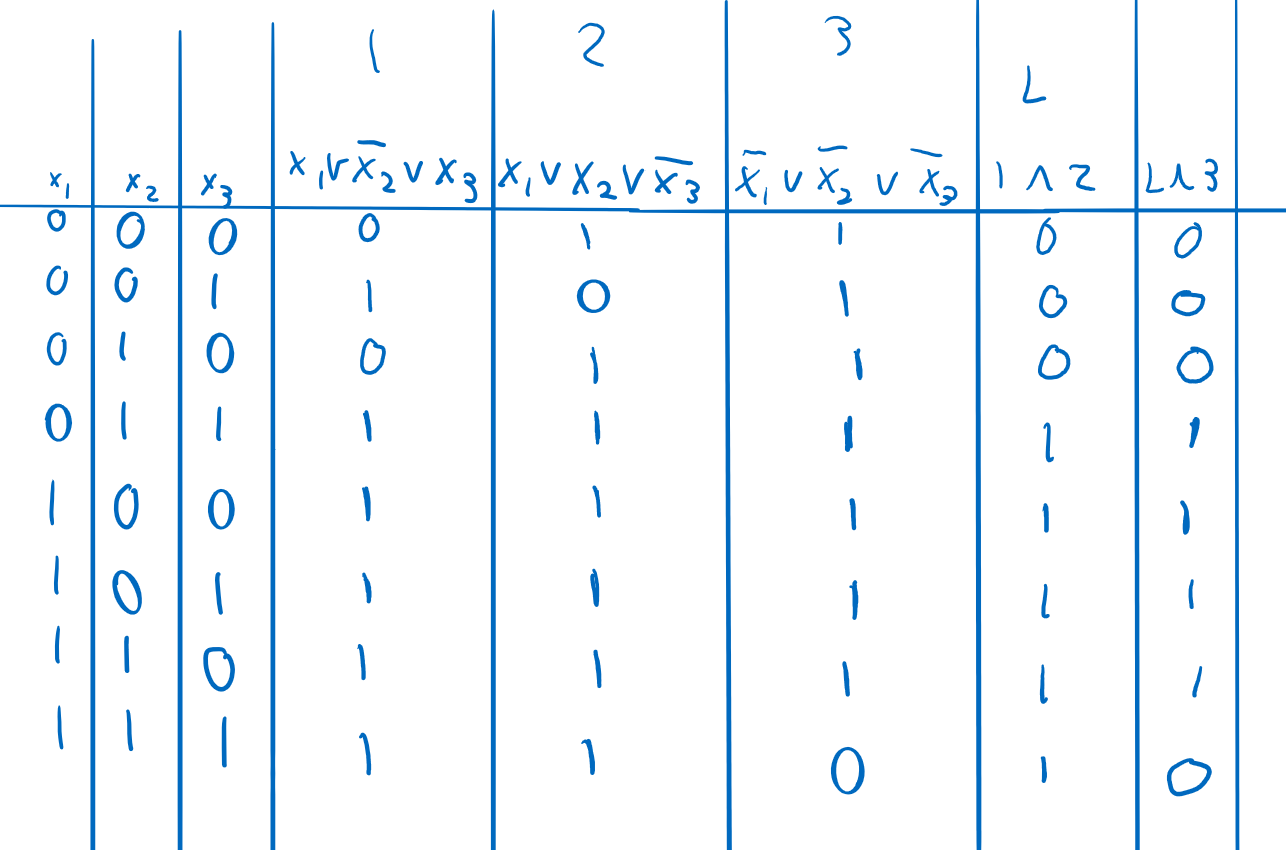
1. w0w3 = v2v1

1001 = 1001

1. w1w3 = v0v1

0001 = 0001

\*Q10. [10] Determine whether the given Boolean expression is satisfiable or not.



There are several configurations that result in the boolean expression being satisfied.

Q11. [10] Show that L = {*www* | w ∈{*a, b*}+} is in DTIME(*n*). Explain how a Deterministic-TM runs on *w* ∈ L in O(*n*).

To show that L is in DTIME(n), we must construct a deterministic multitape – TM that decides L in O(T(n)).

L ∊ DTIME(n) if we can determine where the first-string ends in O(n) time.

1. We can do this by loading the string derived from the language onto tape 1 and looking through each symbol. We can then store a count of the symbols only counting every third symbol. This will give us the length of the first w.
2. We will then use this count to find the starting location for the 2nd substring. Copy this substring to the end of tape 2. Then copy the remaining symbols that represent the third substring to tape 3.
3. The last step is to match tape 1, 2, and 3 symbol by symbol in order to determine if the string is an element of L

Since the moves required only take O(n) time, we see that the overall complexity is also O(n).

Q12. [10] Explain a Halting Problem in detail.

The halting problem solves the problem of determining whether a program/algorithm will ever halt.

There is no generalized algorithm which can correctly determine whether a program/algorithm will half when given a certain input.

We can prove that the halting problem is undecidable by constructing a Turing machine M that outputs a yes if it halts but a no if it does not halt. Since we have no way of ever getting an answer of no, we can never tell if the machine has halted given a certain input.

Q13. [10, optional]

1. Explain P-Problem, NP-Problem and NP-Complete Problem, respectively (P/NP/NP-Complete-language, equivalently).

P, NP, and NP-Complete problems are unsolved problems involving P, Polynomial time, and NP, or Non-deterministic Polynomial time. The first set of problems are those that can be solved in polynomial time and include algorithms for testing to see if a number is prime.

The second set of problems are NP- problems and cannot be solved in polynomial time. One example of a NP-Problem is an algorithm for Integer Factorization. These kinds of problems are classified as such if they can’t be solved in O(nk) time but can also have a yes or no answer that can be verified in polynomial time by a Deterministic Turing Machine.

The final kind of problem is the NP-Complete problems. These are the hardest of the three but is still close to the NP problems. NP-Complete problems can also be a subset of the NP-Hard problems which are problems that are difficult to verify as well as solve. NP-Complete problems have an additional characteristic known as completeness, hence the name. These problems that satisfy the “completeness” attribute will be able to be reduced into any other NP-Complete problem in polynomial time. One example of an NP-Complete Algorithm is the Traveling Salesman problem.

(B) Give one example of NP-Complete problem and define/explain it.

A NP-Complete Problem is defined as a problem that is both an element of the class NP as well as a problem that is reducible in polynomial time.

One example of an NP-Complete problem is the Traveling Salesman Problem. This problem seeks to solve the problem of finding the shortest route a “salesman” can make to n cities. The parameters for this problem are a list of cities as well as the distances between each pair of cities. This problem is an intensively studied to better understand optimization since the salesman can only visit each city once before returning to the starting location.

Q14. [20, optional] In the given table of the closure property, mark whether each language family is closed (O) under the given operation or not closed (X) at (1) – (35). E.g.) ∀L1, L2 ∈ REG → L1 ∪ L2 ∈REG?

O – Closed

X – Not Closed

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **REG** | **DCFL** | **CFL** | **REC** | **RE** |
| **Union** | O | X | O | O | O |
| **Complement** | O | O | X | O | X |
| **Intersection** | O | X | X | O | O |
| **Concatenation** | O | X | O | O | O |
| **Kleene Star (L\*)** | O | X | O | O | O |
| **Reversal (LR)** | O | X | O | O | O |
| **Intersection**  **with REG** | O | X | X | O | O |