Assignment 6

CSCE 4323: Formal Languages and Computability

Fall 2018

Solutions to exercises 1 and 2 should be provided as code in a text document following the format for a TM on the web site http://web.cs.ucdavis.edu/~doty/automata/. A template ﬁle “CSCE4323-F18-HW-template.txt” can be found on the course site on Blackboard. Solutions to the remaining problems should be typed and contained in a PDF, but may include state diagrams or other ﬁgures neatly drawn by hand and included in the PDF. All binary numbers should be represented so that their most signiﬁcant bits are leftmost and their least signiﬁcant bits are rightmost.

1. Give the formal deﬁnition of a Turing machine with Σ = {0,1} which decides the language {w | w contains twice as many 0s as 1s }.

**Simulation in problem1.txt in zip file**

2. Give the formal deﬁnition of a Turing machine with Σ = {0,1} which decides the language {w | w represents n ∈N in binary, n > 0, and n mod 4 = 3}.

3. A ***Turing machine with a doubly inﬁnite tape*** has a tape which extends inﬁnitely in both directions. It initially contains blanks everywhere other than the input string, and the head is assumed to begin at the leftmost symbol of the input. Clearly show that the class of languages recognized by this class of Turing machines is identical to the class of languages recognized by regular Turing machines (i.e. those with tapes which are inﬁnite only to the right).

In order to show that a doubly infinite taped TM, denoted as ***T***, is equal to a regular TM, denoted as ***M***, then we should show that

1. Any language ***L*** that can be recognized by ***M*** can also be recognized by ***T***
2. Any language ***L’*** that can be recognized by ***T*** can also be recognized by ***M***

In order to prove these cases, we must show a simulation of ***T***that acts like ***M*** and vice versa

***Simulation of M that acts like T***

1) Mark the left end of ***T***

2) Prevent ***T*** from moving its head left of that mark

This shows that ***T***simulates ***M***

***Simulation of T that acts like M***

In order to simulate ***T***to act like ***M***,we must make them both doubly infinite taped TM’s since a 2 tape TM is already equivalent in power to a regular TM

***Create another doubly infinite taped TM, denoted by M1***

***M1­***is written with the content of the input alphabet and ***T*** is left blank

Now ***T*** is cut into 2 pieces at the starting cell of the input string, with one part containing the input string and all the blank spaces to the right and the second part containing everything left of the input string in reverse order

This shows that ***M*** simulates ***T***

Therefore, ***T*** is equivalent to ***M*** and shows that doubly infinite taped TM’s are equivalent to regular TM’s

4. A ***Turing machine with left reset*** is similar to an ordinary Turing machine, but the transition function has the form

δ : Q×Γ → Q×Γ×{R,RESET}.

If δ(q,a) = (r,b,RESET), when the machine is in state q reading an a, the machine’s head jumps to the left-hand end of the tape after it writes b on the tape and enters state r. Note that these machines do not have the usual ability to move the head one symbol left. Clearly show that Turing machines with left reset recognize the class of Turing-recognizable languages (i.e. they recognize the same class of languages as regular Turing machines).

In order to show that TM’s with left reset recognizes the class of Turing-Recognizable languages, the TM with left reset must be shown to simulate a regular TM. To show this, let ***M*** be an ordinary TM and let ***MLeft*** be the TM with left reset.

***MLeft*** simulates ***M*** when

* ***M*** makes a right transition then ***MLeft*** does the same
* ***M*** makes a left transition with symbols ***a, b*** then ***MLeft*** replaces those symbols with ***A, B*** which makes the alphabet set and does a left reset

This shifts all content of the tape by one to the right for all symbol other than {***A,B***}. This is then repeated until all content of the tape is shifted in the same manner and:

* ***MLeft***resets again
* All transitions are checked
* Whenever some {***A,B***} is reached, it works the same way that ***M*** does



5. Show that the collection of decidable languages is closed under the operation of

**a. complementation**

Let ***L*** be a decidable language and let ***M*** be a TM that decides ***L***

There must exist a TM, ***M’***, such that *L(M’) = the complement of* ***M***. On input ***w***:

* ***Accept*** if ***M*** rejects
* Otherwise ***reject***

Since ***M’*** does the opposite of what ***M*** does then it decides the complement of ***L***

Therefore, decidable languages are closed under complementation

**b. intersection**

Let ***L1*** and ***L2*** be two decidable languages and let ***M1*** and ***M2*** be the TM’s that decide ***L1*** and ***L2***.

There must exist a TM, ***M’***, such that . On input ***w***:

* Run ***M1­*** on ***w*** and if ***M1*** rejects then ***reject***
* Otherwise, run ***M2*** on ***w*** and if ***M2*** rejects then reject
* If all else fails, then accept

***M’*** accepts ***w*** both ***M1*** and ***M2*** accept it. Otherwise, if either reject then ***M’*** rejects ***w***.

Therefore, and the decidable languages are closed under intersection

**c. concatenation**

Let ***L1*** and ***L2*** be two decidable languages and let ***M1*** and ***M2*** be the TM’s that decide ***L1*** and ***L2***.

There must exist a TM, ***M’***, such that . On input ***w***:

* Split ***w*** into two parts, ***w1, w2***, such that
* Run ***M1*** on ***w1*** and if ***M1*** rejects then ***reject***
* Otherwise, run ***M2*** on ***w2*** and if ***M2*** rejects then ***reject***
* If all else fails, then ***accept***

You must try every possible cut of ***w*** and if the first part and the second part are accepts by ***M1*** and ***M2***, respectively, then ***w*** is accepted by ***M’***. If not, then ***w*** does not belong to the concatenation of languages and is rejected

Therefore, and the decidable languages are closed under concatenation

6. Assume that the digital representation of a picture consists of a single number, representing a color value, for each pixel location. Let I be a picture and 〈I〉 be an encoding (to be speciﬁed by you) of I as a string over Σ, for Σ = {0,1, ; ,#}. Give an implementation-level description of a Turing machine M which takes as input a string w = “x,y,c##〈I〉” where x is the x-coordinate of a pixel location, y is the y-coordinate, c is a number representing a color, and 〈I〉 is an encoding of a picture. M should accept w if the pixel of I at coordinate (x,y) has color c, and should reject otherwise. (Note that the representation of 〈I〉 given as input must be generic and cannot have any special marking which distinguishes the pixel at location (x,y).) Before giving the description of M, clearly specify your encoding of I as 〈I〉, i.e. how the image is represented as a string over Σ. Your description of M should also specify how the numbers for x, y, and c are represented. (Hint: a long representation of numbers may make your implementation easier to deﬁne.)

Let ***M*** be the TM which takes an input ***w*** = “x,y,c##〈I〉” with x being the x-coordinate, y being the y-coordinate, c a color represented as a number, and 〈**I**〉 being the encoding of ***I***. Scan over 〈**I**〉 and match each x-coordinate, y-coordinate, and color to ***I***. The TM ***M*** will accept when there is a match and reject if there isn’t. In order to show boundaries between each ***w***, use the ; as a delimiter (i.e xyc;xyc;xyc …. ). Use ## to denote the beginning and end of the string.