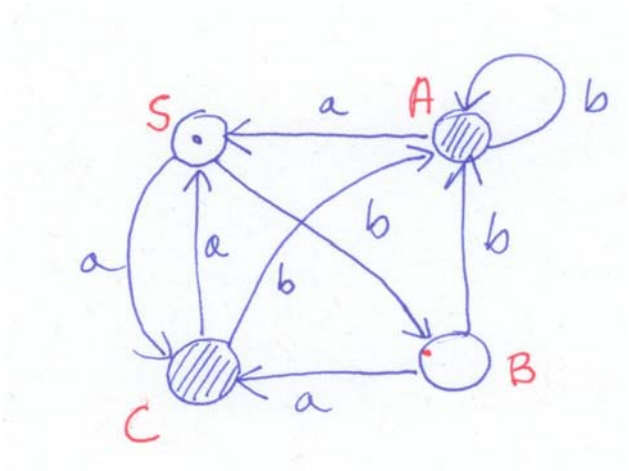
CS2MJ3
Sample solutions to the assignment 3. Total of this assignment is 109 pts. 100% of this assignment is 88 pts. Each assignment is worth 7%.

If you think your solution has been marked wrongly, write a short memo stating where marking in wrong and what you think is right, and resubmit to me during class, office hours, or just slip under the door to my office.

1.[6] Take the finite state automaton from Question 3 of assignment 2 and

a.[3] give a context free grammar that generates it,



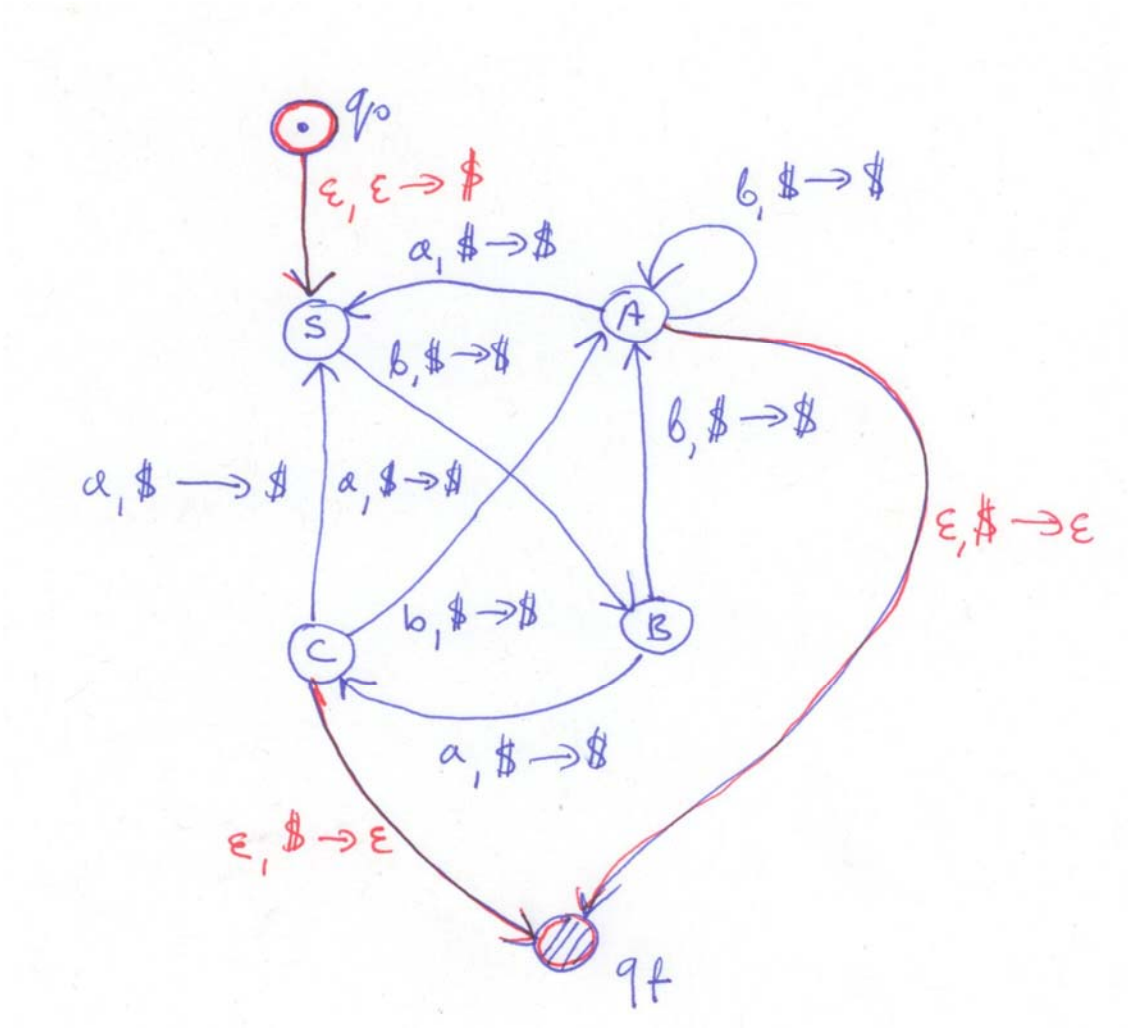
Solution 1:

S → aC | bB
A → aS | bA | ε
B → aC | bA
C → aS | bA | ε

Solution 2:

S → aC | bB | a
A → aS | bA | b
B → aC | bA | b | a
C → aS | bA

b.[3] give a push-down automaton that accepts it,



In fact the solution is 'almost' deterministic. We just need to add the empty set and appropriate arrows.

2.[6] Show that the class of context-free languages is closed under regular operations, union, concatenation, and Kleene star.

Solution:

Let $L_1 = L(G_1)$, $L_2 = L(G_2)$ where $G_1=(V_1, \Sigma_1, S_1, P_1)$, $G_2=(V_2, \Sigma_2, S_2, P_2)$. We may assume that $V_1 \cap V_2= \emptyset$.

Union: Let $G_\cup = (V_1 \cup V_2 \cup \{S_\cup\}, \Sigma_1 \cup \Sigma_2, S_\cup, P_\cup)$, where $P_\cup = \{S_\cup \to S_1, S_\cup \to S_2\} \cup P_1 \cup P_2$.

Clearly $L(G_\cup) = L(G_1) \cup L(G_2) = L_1 \cup L_2$.

Concatenation: $G_\cdot = (V_1 \cup V_2 \cup \{S_\cdot\}, \Sigma_1 \cup \Sigma_2, S_\cdot, P_\cdot)$, where $P_\cdot = \{S_\cdot \rightarrow S_1 S_2\} \cup P_1 \cup P_2$.
Clearly $L(G_\cdot) = L(G_1)\, L(G_2) = L_1 L_2$.

Kleene star: $G_* = (V_1 \cup \{S_*\}, \Sigma_1, S_*, P_*)$, where $P_* = \{S_* \rightarrow S_1 S_*, S_* \rightarrow \varepsilon\} \cup P_1$.
Clearly $L(G_*) = L(G_1)^* = (L_1)^*$.

3.[4] Give a context-free grammar generating the language:

$L = \{a^m b^n c^p d^q \mid m+n = p+q \}$.

Justify your construction.

Solution:

$S \rightarrow aSd|A|B$
$A \rightarrow aAc\ |C$
$B \rightarrow bBd\ |C$
$C \rightarrow bCc \mid \varepsilon$

For all productions, we generate one "a" or "b" on left side, and one "c" or "d" on the right side, which can guarantee that m+n = p+q. Variable "A" cover cases that a ≥q, while variable "B" cover cases that a ≤q.

4.[6] Give a context-free grammar generating the language:

$L = \{ x \mid x \in \{a,b\}^*$ and each prefix of x has at least as many a's as b's $\}$.

Justify your construction.

Solution:

What patterns x must follow to belong to L? It would help if we could express longer strings in L by shorter strings in L. If $x = \varepsilon$ then $x \in L$. If x is non-empty string then x = ay for some y. We have to consider two obvious choices, $y \in L$ and $y \notin L$. If $y \in L$ then we have a pattern $x \in L$ implies x = ay where $y \in L$. Suppose that $y \notin L$. In this case, y must have a prefix which contains one more b than a. Let u be the shortest such prefix of y. Then, the last symbol of u must be b. This means u = wb and x = awbz, some b and z. Now, we argue that both w and z must be in L. First, by the definition of u, each prefix of w has at least as many a's as b's, so $w \in L$. Furthermore, u =wb has exactly one more b than a. So, awb has as many a's as b's. Now assume that t is a prefix of z. Then, awbt is a prefix of x and, so has at least as many a's as b's. Hence t has also at least as many a's as b's, so $z \in L$.

Thus $x \in L$ iff either x=ay for some $y \in L$ or x = awbz for some $w, z \in L$. This lead to the following grammar:

$S \rightarrow \varepsilon \mid aS|aSbS$

5.[8] Which of the following grammars is ambiguous and which is unambiguous. In each case justify your answer in as much formal way as you can.

(a)[2] $S \rightarrow AB \mid aaB$
$A \rightarrow a \mid Aa$
$B \rightarrow b$

It is ambiguous. See following two left derivations for "aab".
$S \rightarrow AB \rightarrow AaB \rightarrow aaB \rightarrow aab$
$S \rightarrow aaB \rightarrow aab$

(b)[2] $S \rightarrow aAb \mid A \mid \varepsilon$
$A \rightarrow aAbb \mid abb$

It is unambiguous. In each derivation S occurs only once, at the beginning. If $x \neq \varepsilon$, $A \rightarrow aAbb$ is then applied several times and a derivation is completed with $A \rightarrow abb$. So, no split can happen, i.e. the grammar is unambiguous.

(c)[2] $S \rightarrow aSb \mid aAb$
$A \rightarrow cAd \mid B$
$B \rightarrow aBb \mid \varepsilon$

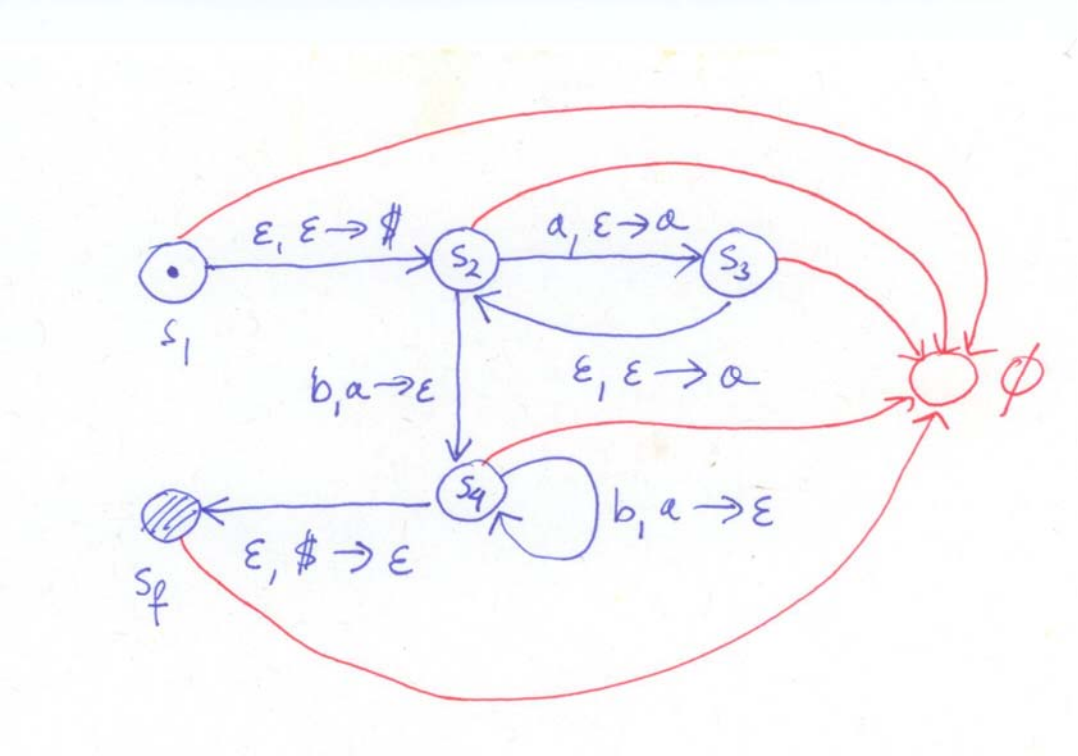
It is ambiguous. See following two left derivations for "aabb".
$S \rightarrow aSb \rightarrow aaAbb \rightarrow aaBbb \rightarrow aabb$
$S \rightarrow aAb \rightarrow aBb \rightarrow aaBbb \rightarrow aabb$

(d)[2] $S \rightarrow aB \mid aA \mid \varepsilon$
$A \rightarrow aS \mid bAA$
$B \rightarrow bS \mid aBB$

It is ambiguous. See following two left derivations for "abaa".
$S \rightarrow aB \rightarrow abS \rightarrow abaA \rightarrow abaaS \rightarrow abaa$
$S \rightarrow aA \rightarrow abAA \rightarrow abaSA \rightarrow abaA \rightarrow abaaS \rightarrow abaa$

6.[4] Show that L = $\{ a^n b^{2n} \mid n \geq 0\}$ is a deterministic context-free language.

Solution: Labels that should be attached to red arrows are omitted.



7.[4] Show that any regular language is a deterministic context-free language.

Solution:

Let L be a regular language and let M = (Q, Σ, δ, s_0, F) be a deterministic finite state automaton such that L = L(M).

Define a deterministic push-down automaton M' = (Q∪{q_0,q_f}, Σ, Γ, δ', q_0, {q_f}), where $q_0 \notin Q$, $q_f \notin Q$, Γ = {$}, and δ' is defined as follows:

a. for all $s \in Q$, $a \in \Sigma$, $\delta'(s, a, \$) = (\delta(s, a), \$)$ and $\delta'(s, a, \varepsilon) = \emptyset$,
b. $\delta'(q_0, \varepsilon, \varepsilon) = (s_0, \$)$ and $\delta'(q_0, \varepsilon, \$) = \emptyset$,
c. for all $s \in F$, $\delta'(s, \varepsilon, \$) = (q_f, \varepsilon)$,
d. for all $a \in \Sigma$, $\delta'(q_0, a, \varepsilon) = \delta'(q_0, a, \$) = \emptyset$,
e. for all $s \in Q$, $\delta'(s, \varepsilon, \$) = \delta'(s, \varepsilon, \varepsilon) = \emptyset$.

Clearly M' is deterministic and L(M') = L(M).

The construction is used in the solution to Question 2(b). In principle we ignore the stack - see rule above (a).

8.[4] Exercise 3.1(c) and 3.1(d) from page 187 of the Sipser's textbook.

We will denote 'blank' by 'B'.

3.1(c) [2] q_1000, Bq_200, Bxq_30, $Bx0q_4B$, $Bx0Bq_{reject}$

3.1(d)[2] $q_1000000$, Bq_200000, Bxq_30000, $Bx0q_4000$, $Bx0xq_300$, $Bx0x0q_40$, $Bx0x0xq_3B$, $Bx0x0q_5xB$, $Bx0xq_50xB$, $Bx0q_5x0xB$, Bxq_50x0xB, $Bq_5x0x0xB$, $q_5Bx0x0xB$, $Bq_2x0x0xB$, Bxq_20x0xB, $Bxxq_3x0xB$, $Bxxxq_30xB$, $Bxxx0q_4xB$, $Bxxx0xq_4B$, $Bxxx0xBq_{reject}$

9.[4] Exercise 3.2(c) and 3.2(d) from page 187 of the Sipser's textbook.

We will denote 'blank' by 'B'.

3.2(c)[2] $q_11\#\#1$, $xq_3\#\#1$, $x\#q_5\#1$, $x\#\#q_{reject}1$

3.2(d)[2] $q_110\#11$, $xq_30\#11$, $x0q_3\#11$, $x0\#q_511$, $x0\#xq_51$, $x0\#xxq_5B$, $x0\#xxBq_{reject}$

10.[10] Exercise 3.8(b) and 3.8(c) from page 188 of the Sipser's textbook.

Solution:
3.8(b)[5] { w | w contains twice as many 0s as 1s}.

1. Scan the tape from the left until the first 0 is found. If it is not found goto 2. If it is found, replace it by *a* and scan for the next 0. If it is not found, *reject* (as the number of 1s is odd). If found, replace it by *a*, go to the beginning of the tape and scan for the first 1. If not found, *reject* (number of 1s is too small). If found, replace by *b*, and goto 1.

2. Go to the beginning of the tape and scan for the first 1. If found, *reject* (number of 1s is too big). If not found *accept* (now tape contains twice as many *a*'s as *b*'s).

3.8(c)[5] { w | w does not contain twice as many 0s as 1s}.

Like 3.8(b), but replace *accept* with *reject* and *reject* with *accept*.

11.[10] Problems 3.15(b) and 3.16(b) from page 189 of the Sipser's textbook.

Solution:

3.15(b)[5] Show that the collection of decidable languages is closed under concatenation.

Let L_1 be decided by a Turing Machine T_1, and let L_2 be decided by a Turing Machine T_2. We have to show that L_1L_2 is decided by some Turing Machine T_{12}.

The problem we have here is that in an arbitrary string w that would be an input to T_{12} we do not know which part should be analysed by T_1 and which by T_2. For $w = a_1a_2...a_n$ it may happen that for some $k<m<n$, both $a_1a_2...a_k \in L_1$ and $a_1a_2...a_m \in L_1$, but $a_{k+1}a_{k+2}...a_n \notin L_2$ and $a_{m+1}a_{m+2}...a_n \in L_2$. So, we have to consider all cases.

Suppose the input $w = a_1a_2...a_n$. We give w to the machine T_1, and T_1 scans w for the first prefix of w that belongs to L_1. If no such a prefix exist, *reject*. Suppose $a_1a_2...a_k \in L_1$ (so T_1 is in its state accept state, which is not the accept state of T_{12}). Then we give $a_{k+1}a_{k+2}...a_n$ to the machine T_2, and T_2 checks if $a_{k+1}a_{k+2}...a_n \in L_2$. If T_2 ends with *accept*, then *accept*. If T_2 ends with *reject*, then we go back to the machine T_1. It starts with its last state it was in before switching to T_2, read a_{k+1} and scan the rest of w for the next prefix of w that belongs to L_1. And next repeat the entire process until either T_2 ends with *accept*, or the entire w was used. *Reject* in the latter case.

Hence T_{12} decides L_1L_2.

3.16(b)[5] Show that the collection of Turing-recognizable languages is closed under concatenation.

We cannot use the solution for decidable languages because of the following problem. Consider the case $w = a_1a_2...a_n$ and for some $k<m<n$, both $a_1a_2...a_k \in L_1$ and $a_1a_2...a_m \in L_1$, but $a_{k+1}a_{k+2}...a_n \notin L_2$ and $a_{m+1}a_{m+2}...a_n \in L_2$. But now languages are only recognizable so T_2 may loop on $a_{k+1}a_{k+2}...a_n$.

We will apply the idea from the construction for transformation of multitape Turing Machines into single tape Turing machines. Let $w = a_1a_2...a_n$. We can divide $w=x_iy_i$, where $x_0 = \varepsilon$, $x_i = a_1a_2...a_i$, $y_i = a_{i+1}a_2...a_n$, for i=0,1,...,n. Suppose we had a TM with infinite number of tapes. Then we can put the string $x_i\$y_i$, on (i+2)th tape (i=0,...,m), and verify *simultaneously* if $x_i \in L_1$ and $y_i \in L_1$, for all i. If for at least one i, $x_i \in L_1$ and $y_i \in L_1$, *accept*. For all i, that either T_1 or T_2 rejects, then *reject*. Note that this new machine may loop if $w \notin L_1L_2$. But we do not have Turing machines with infinite number of tapes. Nevertheless we can still apply the construction from simulation of multitape Turing machine into single tape. Define $w_i = x_i\$y_i$, i=0,1,...,n. Now we start from replacing the input w by $\#w_0\#w_1...w_n\#$. We can now simulate the process described above on single tape.

The same construction can be used for 3.15(b).

12.[4] Use the definitions of primitive recursive functions add(x,y) and mult(x,y) presented in class (see Lecture Notes on the website) to prove that $3 + 4 = 7$ and $2 \times 3 = 6$.

Solution:
add(3,4) = s(add(3,3)) = s(s(add(3,2))) = s(s(s(add(3,1)))) = s(s(s(s(add(3,0))))) = s(s(s(s(3))))= s(s(s(4))) = s(s(5)) = s(6) = 7,
mult(2,3) = add(mult(2,2),2) = add(add(mult(2,1),2),2) = add(add(add(mult(2,0),2),2),2) = add(add(add(0,2),2),2) = add(add(2,2),2) = add(4,2) = 6.

13.[4] Show that $f(x) = ax^2 + bx + c$ is primitive recursive.

Solution:

We defined primitive recurive functions add(x,y) and mult(x,y) in class (see Lecture Notes 8). Hence:

f(x) = add(add(mult(a,mult(x,x)),mult(b,x)),c),

so it is primitive recursive as a composition of primitive recursive functions.

14.[5] Show that the function f(x,y) = if $x \neq y$ then x else 0 is primitive recursive.

Solution:

Note that $x \dot{-} y$ = subtr(x,y) equals 0 if $x \leq y$ and $y \dot{-} x$ = subtr(y,x) equals 0 if $y \leq x$.
Hence $(x \dot{-} y) + (y \dot{-} x)$ equals 0 if x=y and is greater or equal 1 if $x \neq y$.
We can now define equal(x,y) = $1 \dot{-} ((x \dot{-} y) + (y \dot{-} x))$ = if x=y then 1 else 0.
Let notequal(x,y) = $1 \dot{-}$ equal(x,y) = if $x \neq y$ then 1 else 0.

We can now define: f(x,y) = mult(x,notequal(x,y)).

15.[6] For each g(x,y) below, compute h(y) = μy(g(x,y)), and determine its domain

a.[3] g(x,y) = xy - 5

Solution:
Note that $g(x,0) = x \times 0 - 5 = 0 \dot{-} 5 = 0$ all x. Hence h(x) = 0 and the domain is {0,1,2,...}, all natural numbers.

b.[3] $g(x,y) = 2^x + y - 7$

Solution:

h(0) = minimal y such that $2^0 + y - 7 = 1 + y - 7 = y \dot{-} 6 = 0$,
h(1) = minimal y such that $2^1 + y - 7 = 2 + y - 7 = y \dot{-} 5 = 0$,
h(2) = minimal y such that $2^2 + y - 7 = 4 + y - 7 = y \dot{-} 3 = 0$,

h(3) = minimal y such that $2^3 + y - 7 = 8 + y - 7 = y + 1$ = undefined as y+1>0 for all natural y, if $x \geq 3$, then $g(x,y) = 2^x + y - 7 > 0$ for all natural y, so h(x) is undefined.

Hence the domain of h(x) is {0,1,2}.

Bonus questions

16.[6] Design a Turing machine that computes the function f(x,y) = 2x + 3y.

Solution.

The question does not specify the number of tapes. Two tape results the simpler solution than single tape. The input is $0^x\#0^y$ so we first just copy twice 0^x on the second tape and then copy 0^y three times immediately after 0^{2x}. Now we have 0^{2x+3y} on the second tape. The outcome is now on the second tape. It may be enough, but if the definition requires output on the first tape, we just copy it.

Both single and multitape solutions are accepted, both graphical and tabular. A single tape sample solution, detailed and rather lengthy is below. In the solution below we assume the input: 0^x10^y (you may replace 1 by # if you want).

Input of the machine is 0^x10^y, the output string is $0^{(2x+3y)}$. During a computation, we separate the input and output with a blank segment. Each time we erase one 0 to the left of the 1 in the input, we add two 0's at the output. Each time we erase one 0 to the right of the separator 1, we add three 0's at the output string. Following is the formal definition.

The Turing Machine $TM = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

$Q = \{q_0,q_1,q_2,q_3,...q_{21},q_{reject}\}$
$\Sigma = \{0,1\}$
$\Gamma=\{0,1,B\}$ B for blank
$q_{accept}= q_{21}$

Where δ is defined as follow:

	0	1	B
0			B,R,1
1	B,R,2	B,R,12	B,R,21
2	0,R,2	1,R,3	
3	0,R,3		B,R,4
4	0,R,4		0,R,5

5			0,L,6
6	0,L,6		B,L,7
7	0,L,7	1,L,8	
8	0,L,9		B,R,11
9	0,L,0		B,R,10
10	B,R,1		
11		1,R,12	
12	B,R,13		
13	0,R,13		B,R,14
14	0,R,14		0,R,15
15			0,R,16
16			0,L,17
17	0,L,17		B,L,18
18	0,L,19		B,R,21
19	0.L,19		B,R,20
20	0,L,13		
21			

****

In all the tables related to the Turing Machine, a row:

	a	b	c
i	d,R,j	e,L,k	

means:

$\delta(q_i,a) = (q_j,d, R)$
$\delta(q_i,b) = (q_k,e, L)$
$\delta(q_i,c) = (q_{reject},c, L)$, actually only is q_{reject} important here.

17.[5] Ackermann's function is a function from Nat×Nat to Nat, where Nat={0,1,2,...} defined by

$A(0,y)=y+1$
$A(x,0)=A(x-1,1),$

A(x,y+1)=A(x-1,A(x,y)).

The following theorem can be proven:
Theorem.
Let f :Nat→Nat be any primitive recursive function. Then there exists some n ∈ Nat such that
f(i) < A(n,i),
for all i = n, n+1, n+2,.... .

Using the above theorem prove that Ackermann's function is not primitive recursive.

Solution:

Consider the function:
g(i) = A(i,i).
If A(i,j) were primitive recursive, then so would g(i). But then, according to the above theorem, there exists n such that
g(i) < A(n,i),
for all i. If we now pick i=n, we get a contradiction:
g(n)=A(n,n)<A(n,n),
proving that A cannot be primitive recursive.

18.[5] Read Lecture Notes 9. Write a RAM program that accepts the language over the input alphabet {0,1} consisting of all string with the same number of 0's and 1's (formally, $\{ x \mid \#_0(x) = \#_1(x) \}$). Write a pseudo-code first and show precisely which parts of pseudo-code correspond to which part of RAM code.

Solution:

The program read each input symbol into register 1 and in register 2 keeps track of the difference d between the number of 0's and 1's seen so far. When the endmarker $ is encountered, the program checks that the difference is zero, and if so prints 1 and halts.

The pseudo-code below contains the essential details of the algorithm.

```
begin
   d=0;    read(x);
   while x ≠ $ do
         if x ≠ 0 then begin
            d=d+1
            read(x)
             end;
         else begin
            d=d-1
```

```
            read(x)
        end;
if d==0 then write(1) else write(0)
end
```

****

```
                LOAD =0                 d := 0
                STORE 2
                READ 1                  read(x)
                LOAD 1
                SUB =1                  x:=x-1
                STORE 1
while:          LOAD 1
                JUMP$ endwhile          while x ≠ $ do
                JZERO case0             if x ≠ 0 then
                LOAD 2
                ADD =1                  d:=d+1
                STORE 2
                READ 1                  read(x)
                JUMP while              end of if x ≠ 0 do
case0:          LOAD 2
                SUB=1                   d:=d-1
                STORE 2
                READ 1                  read(x)
                LOAD 1
                SUB =2                  x:=x-2
                STORE 1
                JUMP while              end of while x ≠ $ do
endwhile:       LOAD 2
                JZERO yes               if d ≠ 0
                WRITE =0
                HALT
yes:            WRITE =1                if d = 0
                HALT
```

****

19.[5] Prove that ambiguity problem for context-free grammars is undecidable.
Hint. Use the proof that the test if an intersection of context-free languages is the empty set is undecidable.

Solution:

Consider the proof of Theorem from page 9 of Lecture Notes 9. The grammars G_x and G_y are unambiguous (as they have only one variable). Define $G_{xy} = (\{S_x,S_y,S_{xy}\}, \Sigma', S_{xy}, P_{xy})$, where

$G_{xy} = \{ S_{xy} \rightarrow S_x, S_{xy} \rightarrow S_y \} \cup P_x \cup P_x$

Note that G_{xy} is unambiguous if and only if $L(G_x) \cap L(G_y) = \emptyset$ if and only if lists A and B do not have the Post property. Hence, suppose that UNAMB(G) gives TRUE if G is unambiguous and False if it is not. Then POST(A,B) = ¬UNAMB(G_{xy}), where G_{xy} is constructed from the lists A and B, gives a solution to the Post Correspondence Problem, which is undecidable.