CSc 473: Homework Assignment 4

Assigned: Tuesday October 9 2012
Due: 2:00 PM, Tuesday October 23 2012

Clear, neat and concise solutions are required in order to receive full credit so revise your work carefully before submission, and consider how your work is presented. If you cannot solve a particular problem, state this clearly in your write-up, and write down only what you know to be correct. For involved proofs, first outline the argument and the delve into the details.

1. (10 pts) For each of the languages below indicate whether or not it is Context Free. In each case, prove your answer: if you believe L_i is a CFL, give a CFG or a PDA for it and explain why the construction works; if you believe L_j is not a CFL, then prove this using the Pumping Lemma.

   (a) $L_1 = \{a^n b^n a^n b^n | n \geq 0\}$

   answer:

   L_1 is not a CFL.

   Assume L_1 is a CFL. Then it has pumping length p.

   Consider the string $w = a^p b^p a^p b^p$. Then $w \in L_1$.

   Since $|w| \geq p$, we can write $w = uvxyz$ s.t. $|vxy| \leq p$, $|uv| > 0$, and $\forall i \geq 0, uv^i xy^i z \in L_1$.

   Consider such a division $w = uvxyz$.

   Since $|vxy| \leq p$, either vxy is entirely inside one of the p-letter regions, or it extends across two regions.

   If vxy is entirely inside one of the p-letter regions then we are just pumping one of the regions and keeping the rest of the blocks same. So vxy has to be spread across multiple regions. Now v cannot have multiple alphabets in it neither can y. Otherwise, the alphabets will be jumbled.

   So v and y can be part of only two p bloack regions and the two blocks has to be adjacent. So only two regions can be pumped and so at least one region will have more than p alphabets and atleast two regions will have exactly p alphabets. Therefore we have a contradiction, and L_1 is not a CFL.

   (b) $L_2 = \{w | w \in \{a, b\}^*$ and twice the number of a's is equal to three times the number of b's$\}$

   answer:

   L_2 is a CFL, since we can construct the following CFG for it:

$$
\begin{aligned}
S &\to \epsilon \\
S &\to bSbSaSaSa \\
S &\to bSaSbSaSa \\
S &\to aSbSbSaSa \\
S &\to bSaSaSbSa \\
S &\to aSbSaSbSa \\
S &\to aSaSbSbSa \\
S &\to bSaSaSaSb \\
S &\to aSbSaSaSb \\
S &\to aSaSbSaSb \\
S &\to aSaSaSbSb
\end{aligned}
$$

Since there are only two characters in the alphabet of L_2, given any string $w \in L_2$, we can find some division $w = xyz$ such that $y \in L_2$ and $xz \in L_2$. Therefore the above grammar, which produces all permutations of 3 a's and 2 b's, produces the grammar, since all strings of L_2 can be parsed by the language according to the above property, and clearly all strings produced by the grammar will be in the language.

(c) $L_3 = \{a^n ba^{2n} ba^{3n} | n \geq 0\}$

answer:

L_3 is not a CFL.

Assume L_3 is a CFL. Then it has pumping length p.

Consider the string $w = a^p ba^{2p} ba^{3p}$. Then $w \in L_3$.

Since $|w| \geq p$, we can write $w = uvxyz$ such that $|vxy| \leq p$, $|vy| > 0$, and $\forall i \geq 0, uv^i xy^i z \in L_3$.

Consider such a division $w = uvxyz$. Then since $|vxy| \leq p$,

vxy cannot entirely be in a^p , a^{2p} or a^{3p} because then pumping it up will change the count of only one block and will not affect the other two blocks.

Similarly vxy cannot be spread over all three blocks of a^p , a^{2p} or a^{3p} because a^{2p} is of size more than p. So there exists at least one block that vxy is not contained in. While pumping up we increase the count of at least one block while keeping the the count of at least one block the same.

Therefore we have a contradiction, so L_3 is not a CFL.

(d) $L_5 = \{w\#u|$ where $u, w \in \{a, b\}^*$, and w is a substring of $u\}$

answer:

L_5 is not a CFL.

Assume L_5 is a CFL. Then it has pumping length p.

Consider the string $w = 0^p 1^p \# 0^p 1^p$. Then $w \in L_5$, and since $|w| > p$, it can be pumped.

Consider a division $w = uvxyz$ that meets the three criteria of the pumping lemma. If v or y contains the # character, then $uv^0 xy^0 z \notin L_5$. So neither v nor y contains the # character.

Case I. v and y are both in the first portion of the string (before the #). Let $i = 2$. Then the length of the first portion of the string is larger than the length of the second portion of the string(since $|vy| > 0$) , so the first portion cannot be a substring of the second. So $uv^2 xy^2 z \notin L_5$.

Case II. v and y are both in the second portion of the string (after the #). Let $i = 0$. Then the length of the second portion of the string is less than the length of the first portion of the string (since $|vy| > 0$) so the first portion cannot be a substring of the second. So $uv^0 xy^0 z \notin L_5$.

Case III. v is in the first portion of the string, and y is in the second. Then since $|vxy| \leq p$, v is contained within the region of 1's in the first portion of the string, and y is contained within the region of 0's in the second portion of the string, and since $|vy| > 0$, at least one is non-empty.

If v is non-empty, choose $i = 2$, and we have $uv^2 xy^2 z = 0^p 1^{p+j} \# 0^{p+k} 1^p$, where $j > 0$ and $k \geq 0$, so the first portion cannot be a substring of the second (since $p + j > p$).

If y is non-empty, choose $i = 0$, and we have $uv^2 xy^2 z = 0^p 1^{p-j} \# 0^{p-k} 1^p$, where $j \geq 0$ and $k > 0$, so the first portion cannot be a substring of the second (since $p > p - k$).

Then in all cases, we have a contradiction. So L_5 is not a CFL.

2. (10 pts) For each of the languages below indicate whether or not it is Context Free. In each case, prove your answer: if you believe L_i is a CFL, give a CFG or a PDA for it and explain why the construction works; if you believe L_j is not a CFL, then prove this using the Pumping Lemma.

(a) $L_6 = \{a^i b^j c^k | i, j, k \geq 0$ and $j = max(i, k)\}$

answer:

L_6 is not a CFL.

Proof by contadiction. Assume L_6 is a CFL. Then it has pumping length p.

Consider the string $w = a^p b^p c^p$. Then $w \in L_6$.

Since $|w| \geq p$, we can divide it into $w = uvxyz$ in a way that meets the three criteria of the pumping lemma. Consider such a division.

vxy cannot be spread over all three regions because the b block has size p. vxy cannot be contained entirely in a ,b or c block because then we can pump up and j will not be max of (i, k). So vxy has to be spread over either a and b block or it is spread over b and c block. Also v and y can contain only one kind of alphabets otherwise we will be getting jumbled strings.

If v is in a block and y is in b block then we can pump down if y is not empty or we can pump up if y is empty. Both cases we get a string which is not in L_6.

If v is in b block and y is in c block then we can pump down if v is not empty or we can pump up if v is empty. Both cases we get a string which is not in L_6.

Therefore we have a contradiction, so L_6 is not a CFL.

(b) $L_8 = \{www|w \in \{a, b\}^*\}$

answer:

L_8 is not a CFL. Proof by contradiction:

Assume L_8 is a CFL. Then it has pumping length p.

Consider the string $w = ba^pba^pba^p$. Then $w \in L_8$, and since

$|w| \geq p$, w is pumpable.

A division $w = uvxyz$ with $|vxy| \leq p$. So vxy cannot contain more than 1 b. Also v and y cannot contain b's because then if we pump down we will b's that cannot be matched.

So v and y should be contained entirely in some a block. So if we pump up we have at least two blocks of a's which are of different length.

(c) $L_9 = \{a^ib^ja^ib^j|i, j \geq 1\}$

answer:

L_9 is not a CFL. Proof by contradiction:

Assume that L_9 is a CFL. Then it has pumping length p.

Consider the string $w = a^pb^pa^pb^p$. Then $w \in L_9$, and w can be pumped since $|w| \geq p$.

Consider a division $w = uvxyz$ that meets the three criteria of the pumping lemma. Since $|vy| > 0$, either v or y is non-empty. Without loss of generality, assume that v is non-empty (the following argument also holds if y is non-empty). Then if v contains a's and b's, uv^2xy^2z will contain more than 4 regions of a's and b's, so it is not in L_9. If v contains just a's or just b's, then let $i = 0$. Then whichever region v is in will contain fewer than p characters in uv^0xy^0z, and since $|vxy| \leq p$, it is not possible that y could fall in the other region containing the same character (since the two regions are separated by p characters). So the region that v is in will contain fewer characters than the corresponding region of the same character, so $uv^0xy^0z \notin L_9$. Then we have a contradiction. So L_9 is not a CFG.

(d) $L_{10} = \{w_1\#w_2\#...\#w_k|k \geq 0$, each $w_i \in \{a, b\}^*$, and for some $i \neq j$, $w_i = w_j\}$.

answer:

L_{10} is not a CFL. Proof by contradiction:

Assume that L_{10} is a CFL. Then it has pumping length p.

Consider the string $w = a^pb^p\#a^pb^p$. Then $w \in L_{10}$, and since $|w| \geq p$, it can be pumped.

Consider a division $w = uvxyz$ that meets the three criteria of the pumping lemma. Since $|vy| > 0$, either v or y is non-empty. Neither v nor y can contain the # character in w, because then uv^0xy^0z would not contain a #, and would not be in L_{10} (since $k = 1 \not> 1$).

Then there are three options: either v and y are both in the first block of w (before the #), both are in the second block (after the #), or v is in the first block and y is in the second block. We will consider all cases.

I. v and y are both in the first block of w. Then $uv^0xy^0z \notin L_{10}$, since either v or y is non-zero, and therefore the first block of uv^0xy^0z will not match the second.

II. v and y are both in the second block of w. Then $uv^0xy^0z \notin L_{10}$, since either v or y is non-zero, and therefore the second block of uv^0xy^0z will not match the first.

III. v is in the first block and y is in the second. Then $v = b^j$ and $y = a^k$, where $j > 0$ or $k > 0$ (since $|vxy| \leq p$). Then $uv^0xy^0z = a^p b^{p-j} \# a^{p-k} b^p \notin L_{10}$, since $j > 0$ or $k > 0$, and therefore the second block does not match the first.

Therefore in all cases we have a contradiction, so L_{10} is not a CFL.

3. (10 pts) Draw a transition diagram for a deterministic one-tape Turing machine that accepts the language $L_1 = \{a^i b^j c^k k | i, j > 1 \text{ and } k = min(i, j)\}$ Also give a brief explanation of why your Turing machine accepts precisely the given language. A formal proof is not required, but a well-worded argument is going to help if your machine is complicated.

answer:

See attached diagram.

The diagram skips the reject state.

First, the machine turns the first a into an s. (Stage 0) For each a read, the machine turns a c into a 1. If there are more a's than c's, the machine writes y's beyond the end of the string for each additional a. This is performed by stages $ca1$ and $ca2$, which move from the block of a's to the block of c's and turn the first c into a 1 or the first space into a y, stages $a1$ and $a2$, which go back to the first unmarked a, and stage wa, which marks each a by turning it into a d.

Once the a's are consumed, the machine moves on to the b's, shuttling back and forth between c's and b's. For each b, it moves to a character in the c block and does the following: If a c already has a corresponding a (is marked with a 1), the machine writes an x over the character. If a c has no corresponding a, the machine writes a 2 over the character. And if there are more b's than c's, the machine writes z's beyond the end of the string, one for each additional character and this will not be done if a y is already present in the end of the string. States $cb1, b1$ and wb perform these tasks in the same manner as the a states above.

After marking the last b, the machine is at the beginning of what was the c block. The *tally* states check to make sure that the c block has been entirely replaced by x's, and that there are at least two x's. If so, it accepts. If it sees any 1's or 2's, then either the a block or b block was shorter than the c block, so it rejects.

4. (10 pts) Draw a transition diagram for a deterministic one-tape Turing machine that accepts the language $L_2 = \{a^m b^n a^m b^n | m, n > 1\}$ Also give a brief explanation of why your Turing machine accepts precisely the given language. A formal proof is not required, but a well-worded argument is going to help if your machine is complicated.

answer:

See attached diagram.

This diagram omits the reject state.

The machine shuttles back and forth between the two a blocks and marks all the a's in both blocks when they are matched. When the 2nd block has less a's the machine halts. Once all a's from the first block, and their corresponding a's in the second block have been marked, wa will be invoked on the first b in the first b block. The process repeats as above. Once all b's are marked, the machine is at the beginning of what was the second a block, and the *tallyx* states move left to right making sure there are at least two marked and no unmarked a's in the a block, and the *tallyy* states do the same for the second b block, to the end of the string.

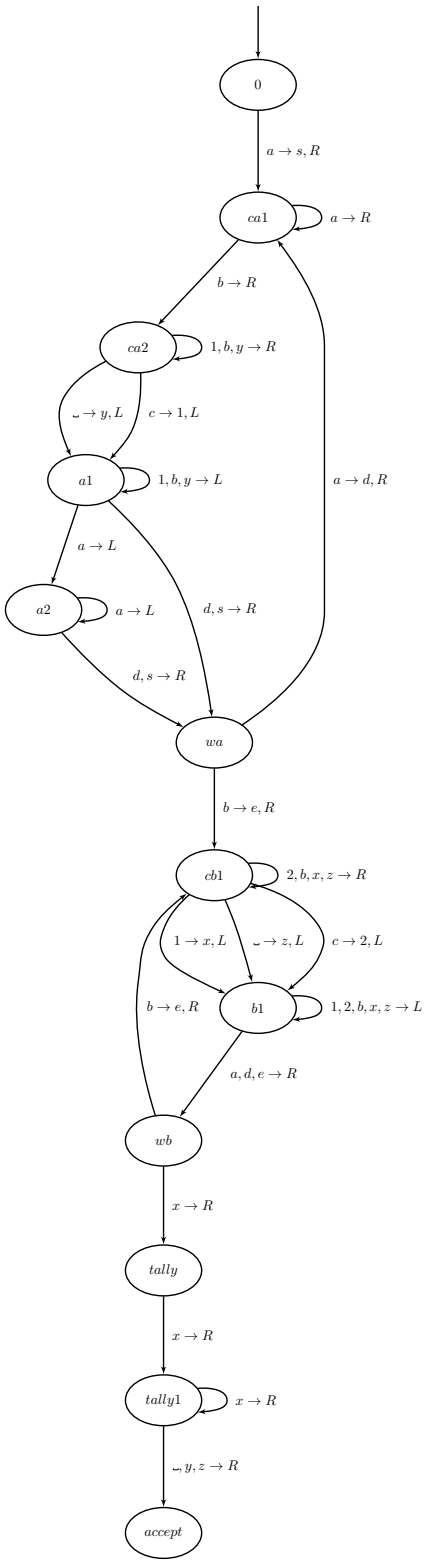

Figure 1: A Turing machine for problem 3.

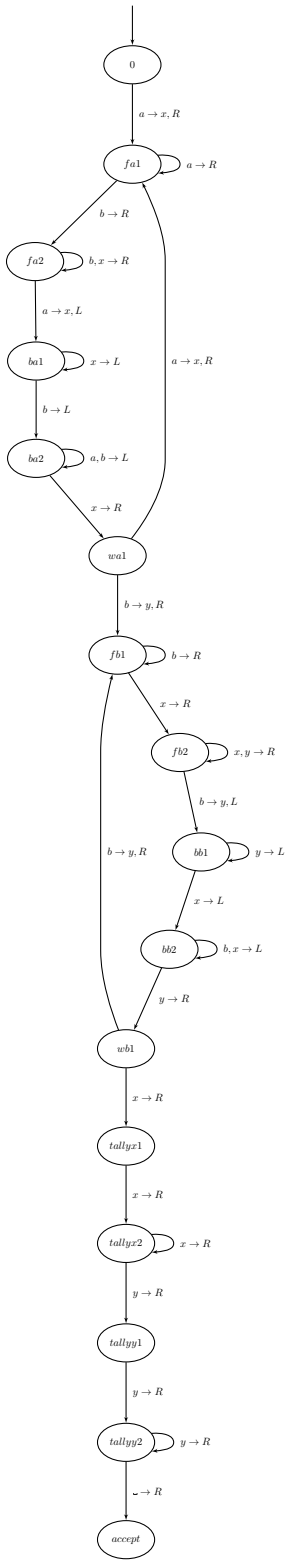

Figure 2: A Turing machine for problem 4.

5. (10 pts) Draw a transition diagram for a deterministic one-tape Turing machine that accepts the language $L_2 = \{a^m b^2 n a^m c^3 n | n > 1\}$ Also give a brief explanation of why your Turing machine accepts precisely the given language. A formal proof is not required, but a well-worded argument is going to help if your machine is complicated.

   answer:

   See attached diagram.

   This diagram omits the reject state.

   The machine shuttles back and forth between the two a blocks, marking a's with x's until all a's from the first block are consumed.

   Once the a's are consumed, wa will be on the first b in the b block. The machine marks two successive b's with y's , then moves to the c block . Once in the c block, the first 3 unmarked c's are marked with z's. If we haven't reached the end of the string, we will return to the first unmarked b, and the loop will continue. If we have reached the end of the string, the *tallyz* states ensure there are at least 6 marked c's and no unmarked c's. Then the *tallyx* states ensure there are at least 2 marked a's in the second block and no unmarked a's, and *checkbs* ensures there are no unmarked b's.

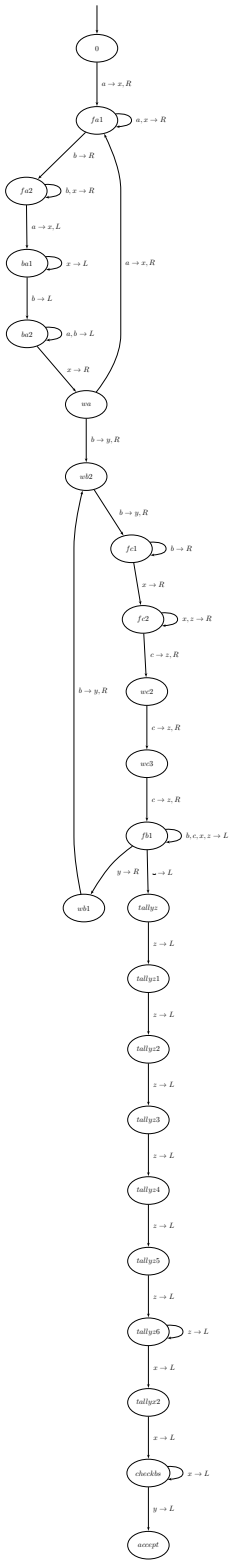

Figure 3: A Turing machine for problem 5.

6. (10 pts) For each of the languages below give an implementation-level description of Turing machine that decides it. (Hint: see examples of implementation-level descriptions of Turing machines in our textbook.)

   (a) $L_1 = \{w | w \in \{a, b\}^*$ and contains an equal number of a's and b's$\}$

   answer:

   On input w

   (1) Scan from left to right until you reach the first unmarked a or $\sqcup$.
   If you stop on an a, mark it, and return to the start of the tape.
   Otherwise, return to the start of the tape and go to (3).

   (2) Scan the tape from left to right until you reach the first unmarked b or $\sqcup$.
   If b, mark it, return to the start of the tape, and go to (1).
   Otherwise, reject.

   (3) Scan from left to right until the first unmarked b or $\sqcup$.
   If b, reject. If $\sqcup$, accept.

   If you ever see a character other than an a,b, or $\sqcup$, reject.

   (b) $L_2 = \{w | w \in \{a, b, c\}^*$ and contains an equal number of a's, b's and c's$\}$

   answer:

   On input w

   (1) Scan from left to right until you reach the first unmarked a or $\sqcup$.
   If you stop on an a, mark it, and return to the start of the tape.
   Otherwise, return to the start of the tape and go to (4).

   (2) Scan the tape from left to right until you reach the first unmarked b or $\sqcup$.
   If b, mark it, and return to the start of the tape.
   Otherwise, reject.

   (3) Scan the tape from left to right until you reach the first unmarked c or $\sqcup$.
   If c, mark it, return to the start of the tape, and go to (1).
   Otherwise, reject.

   (4) Scan from left to right until the first unmarked b, c or $\sqcup$.
   If b or c, reject. If $\sqcup$, accept.

   If you ever see a character other than an a,b,c, or $\sqcup$, reject.

7. (10 pts) Give an implementation-level description of a Turing machine with input alphabet $\{0, 1\}$ that on input u halts with v written on its tape, where if x is the decimal value for u, then the decimal values of v is $x+1$. (In other words, the Turing machine adds 1 to its input and the input is in binary.)

   answer:

   On input u

   (1) Shift u right by one and prefix with \$.

   (2) Scan the tape left to right until you reach $\sqcup$.

   (3) Shift tape head left once.
   If 0, replace with 1 and go to (4).
   If 1, replace with 0 and go to (3).
   If \$, replace with 1 and accept.

   (4) Scan right until $\sqcup$.

   (5) Scan right to left shifting the string left by one until you overwrite \$, then accept.

   If you ever see a character other than a 0,1,\$ or $\sqcup$, reject.

8. (10 pts) Give an implementation-level description of a Turing machine for language $L = \{a^p | p \text{ is a prime number}\}$.

answer:

Let our Turing machine be a two tape machine where each tape is infinite in one direction (to the right). The first tape is the work tape. The work tape has the input (u) on it. The second tape is the counting tape. It keeps track of which divisors have been checked, and it begins empty.

We define the following subroutine to be used in the main routine:
Check whether the divisor matches the input.

Assumption: the tape heads both begin and end over the first a on their respective tapes, and the number of characters on the counting tape should never be greater than the number of characters on the work tape.

1. If the counting tape and work tape both see an a, mark each a, and go right once on each tape. Return to the beginning of step 1.
   If the counting tape and work tape both see a $\sqcup$, ACCEPT.
   If the work tape sees an a while the counting tape sees a $\sqcup$, go left once on each tape.
2. For each tape, if there is a marked a, unmark it and go left once. Return to the beginning of step 2.
   For each tape, if there is a \$, go right once.
   Both tape heads should be over their respective first a, END subroutine.

The main routine:

(1) On the work tape with input u, shift u right by one and prefix with \$.

(2) On the counting tape write \$, go right once, write a, go right once, write a, go left until \$, go right once.

(3) If work tape is over a $\sqcup$, REJECT.

(4) On the work tape go right once, if $\sqcup$, REJECT. Else, go left once.

(5) Call subroutine: check whether the divisor matches the input.

(6) If both counting tape and work tape see an a, mark each a, go right once on each tape, and go to (6).
   If $\sqcup$ on both tapes, REJECT.
   If $\sqcup$ on the counting tape and an a on the work tape, move the counting head left once and go to (7).
   If $\sqcup$ on the work tape and an a on the counting tape, go to (8).

(7) On the counting tape: if a marked a, unmark it, go left once, and go to (7).
   If \$, go right once and go to (6).

(8) On the counting tape: go right until $\sqcup$. Write an a. Go left once.

(9) On the counting tape: if an unmarked a, go left once and go to (9).
   If a marked a, unmark it, go left once and go to (9). If \$, go right once.

(10) On the work tape: go left once.

(11) On the work tape: if a marked a, unmark it, go left once and go to (11).
   If \$, go right once and go to (5).

If you ever see a character other than a a,\$ or $\sqcup$, REJECT.

9. (10 pts) For each of the languages below give an implementation-level description of Turing machine that decides it. (Hint: see examples of implementation-level descriptions of Turing machines in our textbook.)

   The following solutions are over a single-tape Turing machine which is singly infinite to the right.

   (a) $L_1 = \{a^i b^j a^k | i, j, k \geq 1 \text{ and } k = i \cdot j\}$

   answer:

   On input u

   (1) Shift input right prefixing with a \$.

   (2) Scan left until \$ and move right once.

   (3) Check that the input has the form a's followed by b's followed by a's followed by a $\sqcup$:
   If not an a, REJECT. Else, scan right until it sees a b.
   If it doesn't see a b before $\sqcup$, REJECT.
   Otherwise, if it sees a b continue scanning right until it sees another a.
   If it doesn't see another a before $\sqcup$, REJECT.
   If if sees another a continue scanning over a's until it sees a $\sqcup$. If it sees a b, REJECT.

   (4) Scan the tape from right to left until \$ and move right once.

   (5) Scan the tape from left to right until the first unmarked a or unmarked b.

   a. If an unmarked a, mark it and move right until the first unmarked b or $\sqcup$.
   If $\sqcup$, REJECT. Else, continue scanning right until the first unmarked a or $\sqcup$.
   If $\sqcup$, REJECT. Else, mark the a. Go to (4).

   b. If an unmarked b, mark it. Move right once.
   If an unmarked b, scan right to left unmarking any a's seen until \$. Move right once.
   Otherwise, go to (6).

   (6) Scan right to left until \$; move right once.

   (7) Scan left to right until the first unmarked character or $\sqcup$.
   If $\sqcup$, ACCEPT. Else, REJECT.

   If you ever see a character other than a (marked or unmarked) a,b,\$ or $\sqcup$, reject.

   (b) $L_2 = \{ww^R | w \in \{a, b\}^*\}$

   answer:

   On input u

   (1) If $\sqcup$, ACCEPT.

   (2) If a, mark it. Scan left to right until $\sqcup$ or the first marked character. Shift left once.
   - If a, mark it and move left once. If a marked character, ACCEPT.
   Otherwise, scan right to left until the first marked character.
   Move right once and go to (2).
   - Otherwise, REJECT.

   (3) If b, mark it. Scan right to left until $\sqcup$ or the first marked character. Shift left once.
   - If b, mark it and move left once. If a marked character, ACCEPT.
   Otherwise, scan right to left until the first marked character.
   Move right once and go to (2).
   - Otherwise, REJECT.

   If you ever see a character other than a $\sqcup$, a or b (marked or unmarked), REJECT.

(c) $L_3 = \overline{L_2}$

answer:

On input u

(1) If $\sqcup$, REJECT.

(2) If a, mark it. Scan left to right until $\sqcup$ or the first marked character. Shift left once.
- If a, mark it and move left once. If a marked character, REJECT.
  Otherwise, scan right to left until the first marked character.
  Move right once and go to (2).
- Otherwise, ACCEPT.

(3) If b, mark it. Scan right to left until $\sqcup$ or the first marked character. Shift left once.
- If b, mark it and move left once. If a marked character, REJECT.
  Otherwise, scan right to left until the first marked character.
  Move right once and go to (2).
- Otherwise, ACCEPT.

If you ever see a character other than a $\sqcup$, a or b (marked or unmarked), REJECT.

(d) $L_4 = \{ww | w \in \{a, b\}^*$ and w contains $abba$ as a substring$\}$

answer:

On input u

(1) Shift u right by one and prefix with \$.

(2) Scan the tape right to left until you reach \$ and move right.

(3) Scan right until an a. If you reach $\sqcup$, REJECT.

(4) Move right once.

(5) If b, move right once. If a, go to (4).

(6) If b, move right once. If a, go to (4).

(7) If a, capitalize it and move left. If b, go to (3).

(8) (Try to match all the character up to and including the capitalized character with the characters following the capitalized character.)
Scan left until \$, scan right until which ever comes first: an unmarked character or capital.
If capital, go to (9). If a, mark it, scan right until first unmarked character after capital or $\sqcup$.
If a, mark it, move left and go to (8). If $\sqcup$, REJECT. Else, go to (10). If b, mark it, scan right until first unmarked character after capital or $\sqcup$.
If b, mark it, move left and go to (8). If $\sqcup$, REJECT. Else, go to (10).

(9) Scan left until \$, scan right until first unmarked character or $\sqcup$. If $\sqcup$, ACCEPT. Else, REJECT.

(10) Scan right to $\sqcup$. Scan left to \$, replacing all marked characters with unmarked ones except the capital. Scan right to the capital. Remove the capitalization. Move right. Capitalize that character. Go to (8).

If you ever see a character other than a a,b,\$ or $\sqcup$ (marked or unmarked), REJECT.

10. (10 pts) Consider a Turing-like machine TMJ that has a jump-to-first-cell move instead of move left. Argue whether this machine is equivalent to the standard Turing machine or not. If not, what is the class of languages recognized by the machine? (Note: An argument that a machine is equivalent to a Turing machine includes two parts. An argument that a machine is not equivalent involves a counterexample and characterization of language class.)

    Consider a Turing-like machine TMJ that has a jump-to-first-cell move instead of move left. Argue whether this machine is equivalent to the standard Turing machine or not. If not, what is the class of languages recognized by the machine? (Note: An argument that a machine is equivalent to a Turing machine includes two parts. An argument that a machine is not equivalent involves a counterexample and characterization of language class.)

    answer:

    Prove that a TMJ is equivalent to a TM.

    $\Rightarrow$) We can simulate a TMJ with a Turing machine (TM).

    A TMJ has a single tape, infinite in one direction (right), you can move a single space right or jump to the beginning of the tape.

    Let our TM have a single tape, infinite in one direction (right). When the TMJ moves right, move right on the TM. When the TMJ jumps to the beginning of the tape, have the TM scan to the left until it reaches the beginning of the tape (which can be marked to be recognized as such).

    Thus, we can simulate a TMJ with a Turing machine.

    $\Leftarrow$) We can simulate a Turing machine (TM) with a TMJ.

    Any Turing machine can be simulated by a singly infinite, single tape Turing machine. A Turing machine can move one space right or left.

    Let our TMJ have a single tape, infinite in one direction (right).
    When the TM moves right, move right on the TMJ.
    When the TM moves left, mark the current character. Jump to the beginning of the tape and shift the tape content right except that you replace the marked character with a marked version of the previously seen character and then follow it with an unmarked version of the marked character. When you have finished shifting the content, jump to the beginning of the tape and scan until the marked character. This is the character to the left of the original position. Thus, we can simulate moving left.

    This process will continually shift the string right. However, we can undo the shift right if necessary as follows. When shifting the tape content to the right for a move left operation prefix it with a \$.

    1. Mark where the tape head is and jump to the beginning of the tape.
    2. Scan right for the first \$ or $\sqcup$. If $\sqcup$, skip the rest of the steps, else continue to step 3.
    3. Scan right for the first non=\$ character or $\sqcup$.
       Overwrite the non-\$ character with a \$. If no non-\$ character, go to step 5.
    4. Jump to the beginning and scan until the first \$.
       Overwrite the \$ with the character just overwritten, and go to step 1.
    5. Jump to the beginning of the tape and scan right until the first \$.
       Overwrite each \$ with $\sqcup$ and move right until $\sqcup$.
    6. Jump to the beginning of the tape and scan right until the character marked in step 1 and unmark the character.

    We can simulate all the behavior of a TM with a TMJ. Thus, we can simulate a TM with a TMJ.

    Since we can simulate a TMJ with a TM and a TM with a TMJ, they are equivalent.

Extra Credit: Pick a Turing machine simulator of your choice and implement a sorting algorithm. That is, the input should be a coma-separated list of decimal numbers and the output should be a coma-separated sorted list of these numbers.