Do problem zero and **six** of problems 1 through 9. If you write down solutions for more that six problems, clearly indicate those that you want graded. Note that problems can be worth 12, 15 or 20 points: you can attempt between 84 and 108 points depending on the problems that you choose. This exam will be graded on a scale of 100 points.

0. (**3 points**) If you have read and understood the instructions in the previous paragraph, write

   I have read the instructions and understand that I am supposed to solve six of the nine problems; that I can thereby attempt between 84 and 108 total points; and that the exam is graded on a scale of 100.

   as your answer to this problem.

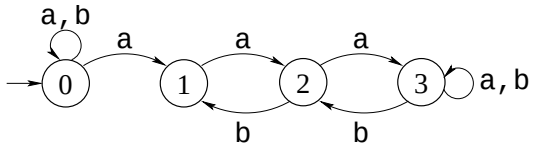
   Solution: I have read the instructions and understand that I am supposed to solve six of the nine problems; that I can thereby attempt between 84 and 108 total points; and that the exam is graded on a scale of 100.

1. (**12 points**) Let $\#\mathsf{a}(w)$ be the number of a's in w and $\#\mathsf{b}(w)$ be the number of b's in w. Draw the transition diagram for an NFA that recognizes the language A_1 defined below:

$$A_1 \quad = \quad \{w \in \{\mathsf{a},\mathsf{b}\}^* \mid \exists x, y, z.\ (w = xyz) \wedge (\#\mathsf{a}(y) \geq \#\mathsf{b}(y) + 3)\}$$

   In other words, the A_1 contains those strings that contain some substring (possibly the entire string itself) that has three more a's than b's.

   Solution:



   It's also possible to turn this into a DFA, just add an arc from state 1 back to state 0 labeled "b", and change the label on the self-arc for state 0 from "a,b" to "b". Because every DFA is an NFA, this is also an acceptable answer.

2. (**20 points**) Let Σ be a finite alphabet. For $x, y \in \Sigma^*$ with $|x| = |y|$, define the *distance* between x and y as the number of symbols for which x and y differ. For those who like formulas:

$$\begin{array}{rcll} dist(\epsilon, \epsilon) & = & 0 & \\ dist(x \cdot c, y \cdot c) & = & dist(x, y), & x, y \in \Sigma^*;\ c \in \Sigma \\ dist(x \cdot c, y \cdot d) & = & dist(x, y) + 1, & x, y \in \Sigma^*;\ c, d \in \Sigma;\ c \neq d \end{array}$$

   Let A be a language. Define

$$\begin{array}{rcl} threeStrikes(A) & = & \{x \mid \exists y \in A\ (|y| = |x|) \wedge (dist(x, y) < 3)\} \\ notBad(A) & = & \{x \mid \exists y \in A.\ (|y| = |x|) \wedge (dist(x, y) \leq |x|/3)\} \end{array}$$

   (a) (**10 points**) Show that if A is a regular language, then $threeStrikes(A)$ is also regular.
   Hint: My solution has five sentences.

Solution: Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes A. Define $N_{3s} = (Q \times \{0,1,2\}, \Sigma, \delta_{3s}, (q_0, 0), F \times \{0,2,2\})$ where

$$\begin{array}{rcl} \delta_{3s}((q,n),c) & = & \{(\delta(q,c),n)\} \cup \{(q', n+1) \mid \exists c' \in \Sigma.\ q' = \delta(q,c')\}, \quad n < 2 \\ \delta_{3s}((q,3),c) & = & \{(\delta(q,c),3)\} \end{array}$$

N_{3s} uses the second component of its state to count the number of mismatch symbols, and it uses its non-determinism to "guess" where and what these mismatch symbols are. $L(N_{3s}) = threeStrikes(A)$, and $L(N_{3s})$ is regular. Thus, $threeStrikes(A)$ is regular as required.

(b) **(2 points)** Show a language A_1 such that A_1 and $notBad(A_1)$ are both regular.

Solution: Let $A_1 = \Sigma^*$. Clearly, $notBad(A_1) = \Sigma^*$ which is regular.

Note that $\emptyset$, any finite language, and many other languages would work as well. For those who don't think it's obvious that $notBad(\Sigma^*) = \Sigma^*$, here's the proof. Let y be any string in Σ^*. Note that $dist(y, y) = 0 \leq |y|/3$. Thus $y \in notBad(\Sigma^*)$ as claimed.

(c) **(2 points)** Show another language A_2 such that A_2 is regular and $notBad(A_2)$ is not regular.

Solution: Let $\Sigma = \{a, b\}$ and $A_2 = a^*$.

(d) **(6 points)** Prove that for your choice of A_2, $notBad(A_2)$ is not regular.

Solution:

Let $p > 0$ be a proposed pumping lemma constant for $notBad(A_2)$.

Let $w = b^p a^{2p}$. $w \in notBad(A_2)$ because $a^{3p} \in A_2$ and $dist(w, a^{3p}) = p = |w|/3$.

Let x, y, and z be any strings such that $w = xyz$, $|xy| \leq p$ and $|y| > 1$.

Clearly, $y = b^k$, for some $k > 0$. Thus, $xy^2z = b^{p+k}a^{2p}$. The only string in A_2 that is comparable to $b^{p+k}a^{2p}$ is a^{3p+k}, and $dist(b^{p+k}a^{2p}, a^{3p+k}) = p + k > (3p+k)/3$. Thus, $xy^2z \notin notBad(A_2)$.

$notBad A_2$ does not satisfy the conditions of the pumping lemma for regular languages. Therefore, it is not regular.

3. (**12 points**) Let

$$\begin{array}{rcl} B_1 & = & \{x \mid \exists w \in \{\mathsf{a}, \mathsf{b}\}^*.\ x = w\#w^{\mathcal{R}}\} \\ B_2 & = & \{x \mid \exists w \in \{\mathsf{a}, \mathsf{b}\}^*.\ x = w\#w^{\mathcal{R}}\#w\} \end{array}$$

(a) **(6 points)** Give a CFG for B_1.

Solution:

$$S_0 \quad \rightarrow \quad \mathsf{a}S\mathsf{a} \mid \mathsf{b}S\mathsf{b} \mid \#$$

(b) **(6 points)** Prove that B_2 is not a CFL.

Solution:

Let $p > 0$ be a proposed pumping lemma constant for B_2.

Let $s = a^p\#a^p\#a^p$. Clearly, $s \in B_2$ – no justification of this claim is needed, but for those who are in doubt, let $w = \mathsf{a}^p$. $w \in \{\mathsf{a}, \mathsf{b}\}^*$; $w^{\mathcal{R}} = w$; and $s = w\#w^{\mathcal{R}}\#w$.

Let u, v, x, y, and z be any strings with $uvxyz = s$, $|vy| > 0$, and $|vxy| \leq p$.

Because $|vxy| \leq p$, it overlaps at most two of the w or $w^{\mathcal{R}}$ substrings of s. Thus, uv^2xy^2z changes the number of a's in one or two of these substrings but not in all three (or none). This shows that $uv^2xy^2z \notin B_2$.

B_2 does not satisfy the conditions of the pumping lemma for CFLs. Therefore, it is not context-free.

4. (**15 points**) As in question 3, let

$$B_2 \quad = \quad \{x \mid \exists w \in \{\mathsf{a}, \mathsf{b}\}^*.\ x = w\#w^{\mathcal{R}}\#w\}$$

Show that $\overline{B_2}$ is a CFL.

Hint: Consider the proof that $\overline{a^n b^n c^n}$ is a CFL (e.g, HW 6, Q1.e).

Solution: Note that

$$\begin{array}{rcl}\overline{B_2} & = & \{w \mid \nexists x, y, x \in \{\mathsf{a}, \mathsf{b}\}^*.\ w = x\#y\#z\} \\ & \cup & \{w \mid \exists x, y, z \in \{a, b\}^*.\ (w = x\#y\#z) \wedge (x \neq y^{\mathcal{R}})\} \\ & \cup & \{w \mid \exists x, y, z \in \{a, b\}^*.\ (w = x\#y\#z) \wedge (y^{\mathcal{R}} \neq z)\}\end{array}$$

Now note that

$\{w \mid \nexists x, y, x \in \{\mathsf{a}, \mathsf{b}\}^*.\ w = x\#y\#z\}$ is regular, and therefore a CFL.

$\{w \mid \exists x, y, z \in \{a, b\}^*.\ (w = x\#y\#z) \wedge (x \neq y^{\mathcal{R}})\}$ is recognized by a PDA that does the following:

0. Pushes an end-of-stack marker, \$ onto the stack.
1. Pushes each a or b that it sees onto the stack until it encounters the first #. This is the x string.
2. Pops the x string off the stack while reading the y string. The PDA uses its finite state to record if it finds a pair of symbols that don't match or if $|x| \neq |y|$.
3. After the second #, the PDA verifies that the rest of the string is in $\{a, b\}^*$.
4. The PDA accepts if it saw that $x \neq y$ in step 2 and the string has two #'s.

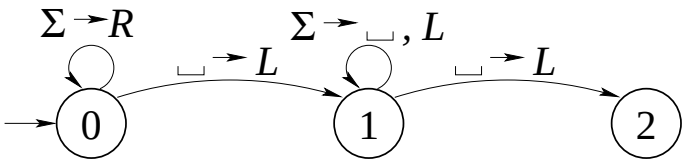
Thus, $\{w \mid \exists x, y, z \in \{a, b\}^*.\ (w = x\#y\#z) \wedge (x \neq y^{\mathcal{R}})\}$ is a CFL.

$\{w \mid \exists x, y, z \in \{a, b\}^*.\ (w = x\#y\#z) \wedge (y^{\mathcal{R}}) \neq z\}$ is a CFL by an argument similar to the one for the previous case.

Thus, B_2 is the union of three CFLs. The CFLs are closed under union; therefore, B_2 is a CFL.

5. (**12 points**) Draw the transition diagram for a Turing machine that erases its tape and then continues from state q_1 with the tape head at the left end of the tape. My solution has three states: q_0 (the initial state); q_1 (the state the TM enters after erasing its tape and moving back to the left end); and q_2 (one more state to get the work done).

Solution:



Note that the space symbol is not in Σ. I'm taking advantage of Sipser's defintion for TM operation that says that if the head is at the leftmost square and the transition says to move the head to the left, the head just stays at the leftmost square (but performs the specified write and state transition).

6. (**15 points**) Let

$$\begin{array}{rcl}A_1 & = & \{[M] \mid \text{There is some non-empty string } w \text{ such that } [M] \text{ reads every symbol of } w \text{ when run with input } w.\ \} \\ A_2 & = & \{[M] \mid \text{There is some non-empty string } w \text{ such that } [M] \text{ does not read every symbol of } w \text{ when run with input } w.\ \}\end{array}$$

(a) (**3 points**) Is $A_1 = \overline{A_2}$? Give a short justification for your answer.

Solution: No. A machine could read every symbol of some strings but not all.

The statements above are a sufficient answer. Here's a more detailed explanation for those who are still in doubt. For example, let M be a machine that recognizes $(\mathsf{a} \cup \mathsf{b})^*\mathsf{b}(\mathsf{a} \cup \mathsf{b})^*$ (i.e. any string in $\{\mathsf{a}, \mathsf{b}\}^*$ with at least one b. Let M scan its tape to the right until it encounters a b at which point it accepts. If it encounters a $\square$, then it rejects. Note that M reads every symbol in the string aa but only the first symbol of bb. Thus, $M \in A_1$ and $M \in A_2$. This shows that $A_1 \neq \overline{A_2}$.

(b) (**6 points**) Is A_1 decidable? Justify your answer.

Solution: A_1 is decidable. Let w be any string consisting of exactly one symbol from the input alphabet of M. M must make at least one move when run with input c (because the start, accept, and reject states are distinct by the definition of a TM). Thus, M reads every symbol of w when run with input w. In other words, A_1 holds for any TM. All a decider has to do is check to make sure that its input is a valid TM description.

(c) (**6 points**) Is A_2 decidable? Justify your answer.

Solution: A_2 is not decidable. I'll reduce A_{TM} to A_2. I'll do this by reducing A_{tm} to $A_{bounded}$. and reducing $A_{bounded}$ to A_2 where

$$A_{bounded} = \{[M]\#w \mid \text{TM } M \text{ uses a bounded amount of space when run with input } w. \}$$

First, I'll reduce A_{TM} to $A_{bounded}$. Let $M_{A_{TM}}$ be a TM that does the following when run with input $[M]\#w$:

```
if([M]#w ∈ A_bounded) /* reduction to A_bounded */
   accept; /* M uses unbounded tape on input w... */
           /* ∴ it must be looping */
else {
   Simulate M running with input w, recording each configuration encountered.
   if(M accepts) reject;
   else if(M rejects) accept;
   else if(M repeats a configuration) /* M is looping */
      accept;
}
```

Now, I'll reduce $A_{bounded}$ to A_2. Let $[M]$ be a TM description and w be an input string to w. Construct the TM description for M' that does the following when run on input x:

1. Write w on its input tape followed by $\square'$. M' then moves its head back to the leftmost tape square. $\square'$ is a symbol that is not in the tape alphabet of M, the orginal TM. Note that M' does not erase its input first; thus there will be some suffix of x left unread on the tape if $|x| > |w| + 1$.
2. M' now runs M except that if it ever encounters a square with a $\square'$, M' does the following:
   2.a Let q be the state M' is in when it encounters the $\square'$ symbol.
   2.b M' writes a $\square$ on the tape; moves one square to the right; and enters state q' (a different q' for each q). Note that there is always at most one $\square'$ symbol on the tape.
   2.c M' writes a $\square'$ on the tape; moves one square to the left and returns to state q.
   2.d M' continues as M.

Clearly, all of these operations are Turing computable.
Consider the case where M is bounded on input w. Let n be the number of tape squares M touches when run with input w. Then, if M' is run with any string of with length greater than n, M' will not read its entire input. Thus, if M is bounded on input w, $M' \in A_2$. Conversely, if M is unbounded on input w, M' will eventually overwrite any input string, and $M' \notin A_2$. This shows that $M' \in A_2$ is bounded iff M is bounded. I've reduce $A_{bounded}$ to A_2, and transitively have reduced A_{TM} to A_2. This shows that A_2 is undecidable. overwrite every symbol of any input string. In this case,

7. (**15 points**) Let

$$A_{42} = \{[M] \mid [M] \text{ describes a TM that accepts at least 42 strings. }\}$$

(a) (**8 points**) Prove that A_{42} is not Turing-decidable.

Solution:

(b) (**7 points**) Prove that A_{42} is Turing-recognizable.

Solution:

8. (**20 points**) A one-counter automaton (OCA) is a 6-tuple $(Q, \Sigma, \delta, q_0, q_f, q_r)$. The symbols $\vdash$ and $\dashv$ are left and right endmarkers; if the input string is w, the OCA's tape will be $\vdash\ w\ \dashv$.

   As the name suggests, the OCA has a counter that can hold any integer. The OCA starts in state q_0 with the counter set to zero and the read-head at the leftmost tape square (the one with the $\vdash$). At each step, the OCA makes a move depending on its current state, the tape symbol currently under the read head, and whether or not the value of the counter is equal to zero. Based on this information, the OCA transitions to a new state; moves its tape head one square to the left or the right; and the counter is either incremented, decremented or left unchanged. If the OCA ever reaches state q_a it accepts, and if it reaches q_r it immediately rejects.

   (a) (**5 points**) Describe an OCA that recognizes the language:

$$B_1 \quad = \quad \{x \mid \exists w \in \{\mathsf{a}, \mathsf{b}\}^*.\ x = w\#w^{\mathcal{R}}\}$$

   You don't need to into lots of detail. My solution has nine sentences.

   Solution: The OCA reads the first symbol of the string before the #. square to the right of the # symbol. If it reaches the right end-marker without seeing a #, it rejects. Otherwise, it performs the following loop:

   Store the current symbol using the finite state. Move the head to the left until the # is reached, and increment the counter with each move. Now, decrement the counter to find the corresponding symbol in the string before the #. If the symbol from the right side was $\dashv$ and the current symbol is $\vdash$ accept. Otherwise, if the symbols differ, reject. Otherwise, move one square to the left, store that symbol in the current state using the counter as before to find the corresponding position in the right substring, check that symbol, and continue.

   (b) (**5 points**) Describe an OCA that recognizes the language:

$$B_2 \quad = \quad \{x \mid \exists w \in \{\mathsf{a}, \mathsf{b}\}^*.\ x = w\#w^{\mathcal{R}}\#w\}$$

   You don't need to into lots of detail. My solution two sentences.

   Solution: Use the same idea as above to check the first $w\#w^{\mathcal{R}}$ part. Then, the OCA moves over past the first # and checks the $w^{\mathcal{R}}w$ part.

   (c) (**10 points**) Prove that the language emptiness problem for OCAs is Turing-undecidable.

   Once again, you don't need lots of detail. Just describe the key points of the reduction so that it's clear that you know how to solve the problem. My solution has seven sentences (using computational histories) or nine (using PCP).

   Solution 1: Computational histories approach

   Let M be a TM and w be an input string. M accepts w iff there exists a computational history,

$$\#config_0\#config_1^{\mathcal{R}}\#\cdots\#config_m\#$$

   that shows M accepting w (where $config_m$ is reversed if m is odd that). We can make the strings for each of the $config_i$'s the same length by appending spaces to the shorter ones to make them all as long as the longest.

   The solution to $ww^{\mathcal{R}}w$ above can be generalized to create a OCA that recognizes $ww^{\mathcal{R}}ww^{\mathcal{R}}\ldots w$. Likewise, the OCA can check that each successive configuration is the correct successor to the previous one – this is just a slight variation of checking that the strings are each other's reverse; the OCA has to take special care of the part of the configuration that encodes the head position and state. Finally, the OCA can use its finite state to verify that the first configuration is the correct one and that the last configuration is accepting.

Solution 2: PCP approach

Here's a reduction from PCP. Let $(a_1, b_1), (a_2, b_2), \ldots (a_k, b_k)$ be an instance of PCP. If there is a solution, it can be written as:

$$a_{i_1}\texttt{,}\, a_{i_2}\texttt{,}\, \cdots\texttt{,}\, a_{i_m} \# b_{i_m}^{\mathcal{R}}\texttt{,}\, \cdots b_{i_2}^{\mathcal{R}}\texttt{,}\, b_{i_1}^{\mathcal{R}}$$

where , and # are symbols that are not in the alphabet for the a_i's and b_i's. A OCA can verify that $a_{i_1} a_{i_2} \cdots a_{i_m} = (b_{i_m}^{\mathcal{R}} \cdots b_{i_2}^{\mathcal{R}} b_{i_1}^{\mathcal{R}})^{\mathcal{R}}$ by essentially the same way that it can verify that a string is of the form $ww^{\mathcal{R}}$; it just ignores the commas. The OCA can then perform a second phase where it again does a $w\#w^{\mathcal{R}}$ check, but this time it reads an a_{i_j} string and determines the set of PCP pairs that it could come from. There are 2^k possible sets, so the OCA can remember this in its finite. It then checks that the corresponding b_{i_j} string is a valid partner for the a_{i_j} string. And so on.

A string will pass both tests iff it is a solution to the PCP problem. Thus, this OCA recognizes a non-empty language iff the PCP problem is solvable.

9. (**20 points**) Let A be a regular language with $|A| = \infty$. Prove that there exists a language B with $B \subseteq A$ such that B is not Turing-recognizable.

Solution: