CSC236 Winter 2017
Assignment #3: Algorithm Analysis, Formal Languages — Sample Solution
Due April 5th, by 9:00 pm

The aim of this assignment is to give you some practice with topics related to formal languages. You will also design an algorithm, and do a complete analysis of it.

For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks will be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions. You should clearly cite any sources or people you consult, other than the course notes, lecture materials, and tutorial exercises.

Your assignment must be typed to produce a PDF document **a3.pdf** (hand-written submissions are not acceptable). You may *neatly* hand draw and scan your FSA diagrams to include them in your PDF submission. You may work on the assignment in groups of 1, 2, or 3, and submit a single assignment for the entire group on MarkUs.

1. Give pseudocode for an algorithm to solve the following problem: given a sequence of numbers, determine if the sequence is the postorder traversal of some binary search tree. Then, give a complete analysis of the complexity and correctness of your algorithm.

   There are no restrictions on your algorithm (other than that it solves the problem), however there is a Divide & Conquer approach that should make your analysis fairly straightforward.

   You are encouraged to use proofs by induction, as we have in the course, to show your results. Other proof techniques are acceptable, but less likely to get partial marks if incorrect.

   Your analysis should be clearly laid out for the reader - correct proofs that are difficult to follow will not receive full marks.

   (Other solutions are acceptable if they solve the problem, and have a correct analysis.)

```
isBSTPostorder(A, b, e):
    # Precondition: A is an array of numbers, b <= e, b and e are valid indices in A
    if b == e or b == e + 1:
        return true
    left = b
    while left < e and A[left] <= A[e]:
        left += 1
    right = left
    while right < e and A[right] >= A[e]:
        right += 1
    return right == e and isBSTPostorder(A, b, left - 1) and isBSTPostorder(A, left, e - 1)
    # Postcondition: returns true iff A[b..e] is a postorder traversal of some BST
```

   Complexity: Let $n = e - b + 1$.

   The worst case for $isBSTPostorder$ is when $right = e$ and either $left = right$ or $left = b$ on each recursive call. These two cases are symmetric — assume without loss of generality the case that $right = e$ and $left = b$. In this case, the sequence represents the postorder traversal of a BST where every node (including the root) has only a right child.

The first while loop iterates 0 times, as $left$ is never incremented. The second while loop iterates $e - b = n - 1$ times. The first recursive call has input of size 0, and second recursive call has input of size $n - 1$.

We can express the worst case running time $T(n)$ of the algorithm as follows:

$$T(n) = \begin{cases} c & \text{if } n = 1\text{, for some } c \in \mathbb{N} \\ T(n-1) + f(n) & \text{if } n > 1\text{, where } f(n) \in \Theta(n) \end{cases}$$

To simplify the unwinding, let $c = 1$ and $f(n) = n$. Note that these assumptions will simplify the unwinding, without affecting the bound.

$T(n) = T(n-1) + n = T(n-2) + (n-1) + n = \dots$

After the ith substitution, $T(n) = T(n-i) + \Sigma_{j=n-i+1}^{n} j$.

So $T(n) = T(1) + \Sigma_{j=2}^{n} j = 1 + \frac{(n+1)n}{2} - 1 = \frac{(n+1)n}{2}$.

So $T(n) \in \Theta(n^2)$. (Note that different choices of c or $f(n)$ would lead to the same bound.)

Correctness: Here, we need to show correctness of an algorithm with both recursive calls and loops. We can prove these separately, and then conclude that the algorithm is correct. The preconditions and postconditions for the algorithm are indicated with the pseudocode.

Here, we will (1) determine pre- and post-conditions and loop invariants for the while loops, (2) assume the loops are correct and prove the algorithm is correct from that assumption, and (3) prove the loops are correct. Then, we can conclude that the algorithm is correct.

(1) Let $L1$ be the first loop in the algorithm, and $L2$ the second. For $L1$, the precondition is that $left = b$, and the postcondition is that $left \leq e$ and $A[b..left-1] \leq A[e]$. The loop invariant is the same as the postcondition. For $L2$, the precondition is that $right = left$, and the postcondition is that $right \leq e$ and $A[left..right-1] \geq A[e]$. The loop invariant is the same as the postcondition.

(2) We now assume the loop invariants are correct, and show the algorithm is correct. That is, show that the algorithm terminates and satisfies the postconditions for all inputs that satisfy the preconditions, assuming the loops are correct.

Proof, by induction. Let $n \geq 0$, and assume $H(n)$: the algorithm terminates and satisfies the postconditions on inputs of size i, $\forall i \in \mathbb{N}$, $1 \leq i < n$ that satisfy the preconditions.

Case: $b = e$ or $b = e + 1$, i.e. $n = 0$ or $n = 1$. In this case, the first if condition is true, and the algorithm terminates and returns true. A sequence of length 0 is a valid postorder traversal of an empty BST. A sequence of length 1 is a valid postorder traversal of a BST that contains only a single node.

Case: $b < e$, i.e. $n > 1$. In this case, the first if condition is false. We assume the loops are correct (i.e. satisfy postconditions and terminate), so after both loops have executed, $A[b..left-1] \leq A[e]$ and $A[left..right-1] \geq A[e]$.

By the BST property and the definition of postorder, $A[b..e]$ is a postorder traversal of a BST iff $A[b..e] =$ [values in left subtree] + [values in right subtree] + [root], where all values in the left subtree are smaller than the root, all values in the right subtree are larger than the root, and the values for each subtree are also a postorder traversal of a BST.

If $right < e$, then there exist some values in A that could not be included in either a left subtree (ie. $A[b..left-1]$) or a right subtree (ie. $A[left..right-1]$), and so $A[b..e]$ is not a postorder traversal of any BST. The algorithm will terminate immediately, and return false.

If $right == e$, then $A[b..e-1]$ can be divided such that $A[b..left-1] \leq A[e]$, and $A[left..e-1] \geq A[e]$. Here, $A[b..e]$ is a postorder traversal of a BST if both $A[b..left-1]$ and $A[left..e-1]$ are postorder traversals of some BST. Each recursive call returns true iff the given range in A is a postorder traversal of some BST (by $H(n)$, since $left \leq e$ and $right \leq e$). By $H(n)$, the recursive calls also terminate. Then, the algorithm terminates returns true if both recursive calls return true, and false otherwise.

Therefore, in all cases, the algorithm terminates and satisfies the postcondition on inputs that satisfy the precondition.

(3) It remains to be shown that the loop invariants are correct, by induction, for each loop.

Proof for $L1$: Let $k \geq 0$, and let LI_k refer to the loop invariant after iteration k.

Assume $H(k)$: LI_k holds. i.e. After iteration k, $left_k \leq e$ and $A[b..left_k - 1] \leq A[e]$. Show $H(k) \leftarrow C(k)$: after iteration $k+1$, LI_{k+1} holds.

Case: There is no $k+1$th iteration, so $left_{k+1} = left_k$, and $LI_{k+1} = LI_k$, so by $H(k)$, LI_{k+1} holds.

Case: There is a $k+1$th iteration. Then, since this iteration exists, $left_k < e$, so $left_{k+1} = left_k + 1 \leq e$. Since this iteration exists, $A[left_k] \leq A[e]$. $A[b..left_k - 1] \leq A[e]$ (by $H(k)$), and $left_k = left_{k+1} - 1$, so $A[b..left_{k+1} - 1] \leq A[e]$.

Base Case: Let $k = 0$. Then $left_0 = b$. $b \leq e$ by the algorithm's precondition, and the second part of the invariant is vacuously true, since $A[b..b-1]$ is empty.

Therefore, in all cases, the loop invariant holds.

It remains to be shown that the loop terminates. Let $E_k = e - left_k + 1$ for the kth iteration. Since $e, left_k \in \mathbb{N}$, and, by the loop invariant, $left_k \leq e$, then $E_k \in \mathbb{N}$. $E_{k+1} = e - left_{k+1} + 1 = e - (left_k + 1) + 1 = e - left_k < E_k$. Thus, by well-ordering, the loop terminates.

Proof for $L2$: Let $k \geq 0$, and let LI_k refer to the loop invariant after iteration k.

Assume $H(k)$: LI_k holds. i.e. After iteration k, $right_k \leq e$ and $A[left..right_k - 1] \geq A[e]$. Show $H(k) \leftarrow C(k)$: after iteration $k+1$, LI_{k+1} holds.

Case: There is no $k+1$th iteration, so $left_{k+1} = left_k$, and $LI_{k+1} = LI_k$, so by $H(k)$, LI_{k+1} holds.

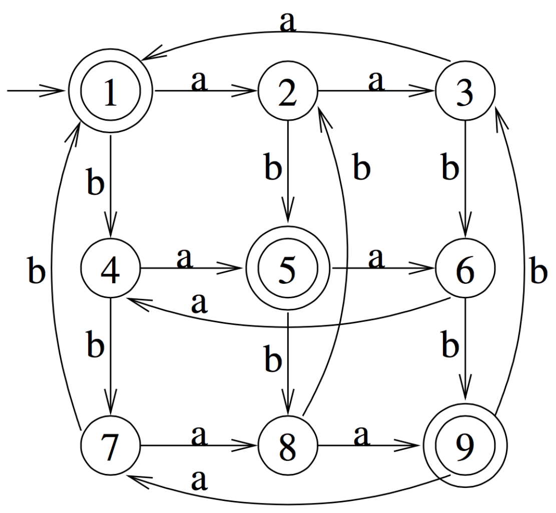
Case: There is a $k+1$th iteration. Then, since this iteration exists, $right_k < e$, so $right_{k+1} = right_k + 1 \leq e$. Since this iteration exists, $A[right_k] \geq A[e]$. $A[left..right_k - 1] \geq A[e]$ (by $H(k)$), and $right_k = right_{k+1} - 1$, so $A[left..right_{k+1} - 1] \geq A[e]$.

Base Case: Let $k = 0$. Then $left_0 = b$. $b \leq e$ by the algorithm's precondition, and the second part of the invariant is vacuously true, since $A[b..b-1]$ is empty.

Therefore, in all cases, the loop invariant holds.

It remains to be shown that the loop terminates. Let $E_k = e - right_k + 1$ for the kth iteration. Since $e, right_k \in \mathbb{N}$, and, by the loop invariant, $right_k \leq e$, then $E_k \in \mathbb{N}$. $E_{k+1} = e - right_{k+1} + 1 = e - (right_k + 1) + 1 = e - right_k < E_k$. Thus, by well-ordering, the loop terminates.

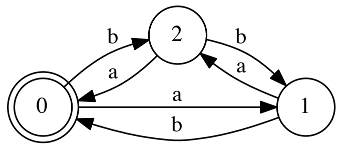
2. Consider the following DFA:



(a) Give a one-sentence description of the language accepted by this DFA.

All strings of a's and b's where the difference between the number of a's and the number of b's is a multiple of 3.

(b) Give another DFA that is equivalent to the one above but uses as few states as possible.

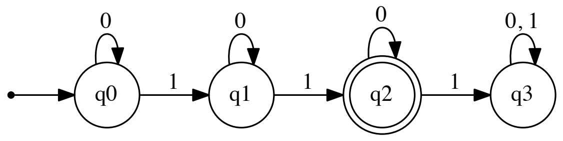


(c) Justify that your DFA accepts the same language as the one above. (You do not have to write a formal proof.)

The second DFA keeps track of the number of a's minus the number of b's, modulo 3, making use of the facts that $-1 \mod 3 = 2$ and $-2 \mod 3 = 1$.

3. For each language below, give the transition diagram for a DFA A that accepts the language.

(a) $L = \{s \in \{0,1\}^* : s \text{ has exactly two 1's}\}$



$$\delta^*(q_0, s) = \begin{cases} q_0 & \text{if } s \text{ has no 1's} \\ q_1 & \text{if } s \text{ has exactly one 1} \\ q_2 & \text{if } s \text{ has exactly two 1's} \\ q_3 & \text{if } s \text{ has more than two 1's} \end{cases}$$

Proof: Let $n \geq 0$ and assume $H(n)$: the state invariant holds for all strings in $\{0,1\}^n$.
Let $s \in \{0,1\}^{n+1}$. Since $n \geq 0$, s can be written as $t \cdot c$, where $t \in \{0,1\}^n$ and $c \in \{0,1\}$.
We now show the state invariant holds for all possible cases of $\delta^*(q_0, t)$ and of c.
Case: $\delta^*(q_0, t) = q_0$. Then, by $H(n)$, t has no 1's. ie. $t = 0^n$.
Subcase: $c = 0$. From the DFA, $\delta(q_0, 0) = q_0$. s contains no 1's because t is all 0's and c is a 0, as required.
Subcase: $c = 1$. From the DFA, $\delta(q_0, 1) = q_1$. s contains exactly one 1 because t is all 0's and c is a 1, as required.
Case: $\delta^*(q_0, t) = q_1$. Then, by $H(n)$, t has exactly one 1.
Subcase: $c = 0$. From the DFA, $\delta(q_1, 0) = q_1$. s contains exactly one 1 because t contains exactly one 1 and c is a 0, as required.
Subcase: $c = 1$. From the DFA, $\delta(q_1, 1) = q_2$. s contains exactly two 1's because t contains exactly one 1 and c is a 1, as required.
Case: $\delta^*(q_0, t) = q_2$. Then, by $H(n)$, t has exactly two 1's.
Subcase: $c = 0$. From the DFA, $\delta(q_2, 0) = q_2$. s contains exactly two 1's because t contains exactly two 1's and c is a 0, as required.

Subcase: $c = 1$. From the DFA, $\delta(q_2, 1) = q_3$. s contains more than two 1's because t contains exactly two 1's and c is a 1, as required.

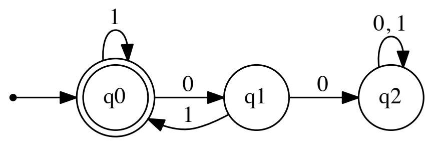
Case: $\delta^*(q_0, t) = q_3$. Then, by $H(n)$, t has more than two 1's. (Note that q_3 is a dead state, but we include it in our state invariant for a complete proof.)

In either case for c, $\delta(q_3, c) = q_3$, since if t contains more than two 1's, then s contains more than two 1's.

Base Case: Let $n = 0$. $\delta^*(q_0, \epsilon) = q_0$, and ϵ has zero 1's.

Thus, in all cases, the state invariant holds.

(b) $L = \{s \in \{0, 1\}^* : \text{each 0 in } s \text{ is followed by one or more 1's}\}$



$$\delta^*(q_0, s) = \begin{cases} q_0 & \text{if } s \text{ has every 0 followed by at least one 1} \\ q_1 & \text{if } s \text{ ends with a 0 and has every 0 except the last followed by at least one 1} \\ q_2 & \text{if } s \text{ has one or more 0's followed by a 0} \end{cases}$$

Proof: Let $n \geq 0$ and assume $H(n)$: the state invariant holds for all strings in $\{0, 1\}^n$.

Let $s \in \{0, 1\}^{n+1}$. Since $n \geq 0$, s can be written as $t \cdot c$, where $t \in \{0, 1\}^n$ and $c \in \{0, 1\}$.

We now show the state invariant holds for all possible cases of $\delta^*(q_0, t)$ and of c.

Case: $\delta^*(q_0, t) = q_0$. Then, by $H(n)$, every 0 in t is followed by at least one 1.

Subcase: $c = 0$. From the DFA, $\delta(q_0, 0) = q_1$. s ends with a 0, because c is 0, and every other 0 is followed by at least one 1, because every other 0 comes from t. So s is as required.

Subcase: $c = 1$. From the DFA, $\delta(q_0, 1) = q_0$. Every 0 in s is followed by at least one 1, as required, because every 0 in s comes from t.

Case: $\delta^*(q_0, t) = q_1$. Then, by $H(n)$, t ends with a 0, and every 0 in t except the last is followed by at least one 1.

Subcase: $c = 0$. From the DFA, $\delta(q_0, 0) = q_2$. Then s has a 0 followed by another 0, because 0 is the last symbol of t, and c is a 0, as required.

Subcase: $c = 1$. From the DFA, $\delta(q_0, 1) = q_0$. Then every 0 in s is followed by at least one 1, as required, because every 0 in t except the last is followed by at least one 1, and c is a 1 which follows the last 0 in t in s.

Case: $\delta^*(q_0, t) = q_2$. Then, by $H(n)$, t has one or more 0's followed by a 0. (Note that q_2 is a dead state, but we include it in our state invariant for a complete proof.)

In either case for c, $\delta(q_2, c) = q_2$, since if t contains one or more 0's followed by a 0, then s does too.

Base Case: Let $n = 0$. $\delta^*(q_0, \epsilon) = q_0$, and the state invariant is vacuously true, as there are no 0's in ϵ.

Thus, in all cases, the state invariant holds.

For one of the languages above, find and prove a state invariant by induction over $|s|$.

4. Give a regular expression that generates each language below.

(a) All binary strings with a double symbol (00 or 11) somewhere

$(0 + 1)^*(00 + 11)(0 + 1)^*$ (other solutions exist)

Proof (by double inclusion):

Let $L = \{s \in \{0,1\}^* : s \text{ has a double symbol somewhere}\}//$. Let $R = (0+1)^*(00+11)(0+1)^*$.

Show $L \subseteq L(R)$: Let s be an arbitrary string in L. Then s has a double symbol somewhere, and $s = xyz$ where x and z are some binary string, and y is a double symbol. i.e. $y = 00$ or $y = 11$. Thus, $x, z \in L((0+1)^*)$ and $y \in L(00+11)$. So $s \in L((0+1)^*(00+11)(0+1)^*) = L(R)$.

Show $L(R) \subseteq L$: Let s be an arbitrary string in $L(R)$. Then $s = tuv$ such that $t, v \in L((0+1)^*)$ and $u \in L(00+11)$. Thus u is either 00 or 11, both of which are double symbols. So $s \in L$.

Therefore, $L = L(R)$.

(b) All binary strings without a double symbol anywhere

$(0+\epsilon)(10)^*(1+\epsilon)$ (other solutions exist)

Proof (by double inclusion):

Let $L = \{s \in \{0,1\}^* : s \text{ has no double symbol anywhere}\}$

Let $R = (0+\epsilon)(10)^*(1+\epsilon)$.

Show $L \subseteq L(R)$: Let s be an arbitrary string in L. Then s has no double symbol anywhere, and so s must be made up of alternating symbols. Let $s = tuv$. If s begins with 0, then $t = 0$, and if s begins with 1, then $t = \epsilon$. If s ends with 1, then $v = 1$, and if s ends with 0, then $v = \epsilon$. u is a binary string of alternating 0's and 1's, and by the choice of t and v, then u begins with 1 and ends with 0. Thus, $t \in L(0+\epsilon)$, $v \in L(1+\epsilon)$, and $u \in L(10)^*$, and so $s \in L((0+\epsilon)(10)^*(1+\epsilon)) = L(R)$.

Show $L(R) \subseteq L$: Let s be an arbitrary string in $L(R)$. Then $s = xyz$ s.t. $x \in L(0+\epsilon)$, $y \in L((10)^*)$, and $z \in L(1+\epsilon)$. If $y = \epsilon$, then $s \in \{\epsilon, 0, 1, 01\} \subset L$. If $y \neq \epsilon$, then y is some number concatenations of 10, and so y begins with 1, ends with 0, and has no double symbol. x is 0 or ϵ, and z is 1 or ϵ, so s has no double symbol anywhere ie. $s \in L$.

Thus, $L = L(R)$.

For one of the languages above, prove your regular expression is correct.

5. For every language $L \subseteq \Sigma^*$, define $\text{BEG}(L) = \{x \in \Sigma^* : \exists y \in \Sigma^*, xy \in L\}$. (Intuitively, BEG($L$) is the set of all strings that are the beginning of some string in L.) For example, $\text{BEG}(\{a, ab, bab\}) = \{\epsilon, a, ab, b, ba, bab\}$.

Prove that the class of regular languages is closed under the BEG operation. That is, for every regular expression R, there exists a regular expression R_B such that $L(R_B) = \text{BEG}(L(R))$.

We prove the statement "$\exists R_B, L(R_B) = \text{BEG}(L(R))$" by structural induction on the regular expression R.

Base Cases:

- $\text{BEG}(L(\emptyset)) = \emptyset = L(\emptyset)$.
- $\text{BEG}(L(\epsilon)) = \{\epsilon\} = L(\epsilon)$.
- $\text{BEG}(L(a)) = \{\epsilon, a\} = L(\epsilon + a)$ for all $a \in \Sigma$.

Inductive Hypothesis: Assume S and T are arbitrary regular expressions and S_B and T_B are regular expressions that satisfy $L(S_B) = \text{BEG}(L(S))$, $L(T_B) = \text{BEG}(L(T))$.

Cases:

- If $R = S + T$, then $\text{BEG}(L(R)) = \text{BEG}(L(S) \cup L(T)) = \text{BEG}(L(S)) \cup \text{BEG}(L(T))$ because $\exists y, xy \in L(S) \cup L(T) \iff (\exists y, xy \in L(S)) \vee (\exists y, xy \in L(T))$.
  Hence, $L(S_B + T_B) = \text{BEG}(L(S+T))$ so we can let $R_B = S_B + T_B$.
- If $R = ST$, then $\text{BEG}(L(R)) = \{x \in \Sigma^* : \exists y \in \Sigma^*, \exists z_1 \in L(S), \exists z_2 \in L(T), xy = z_1 z_2\}$. But then, either x is some initial portion of z_1 (in which case $x \in L(S_B)$) or x consists of z_1 followed by some initial portion of z_2 (in which case, $x \in L(ST_B)$).
  Hence, $L(S_B + ST_B) = \text{BEG}(L(ST))$ so we can let $R_B = S_B + ST_B$.

- If $R = S^*$, then every string in $\mathrm{BEG}(L(S^*))$ can be completed to become a string made up of some number of strings from $L(S)$ concatenated to each other. Every such string must consist of some number of strings from $L(S)$ followed by some initial portion of a string from $L(S)$.
  Hence, $L(S^* S_B) = \mathrm{BEG}(L(S))$ so we can let $R_B = S^* S_B$.

By structural induction, for all regular expressions R, $\exists R_B, L(R_B) = \mathrm{BEG}(L(R))$, i.e., regular languages are closed under the BEG operation.