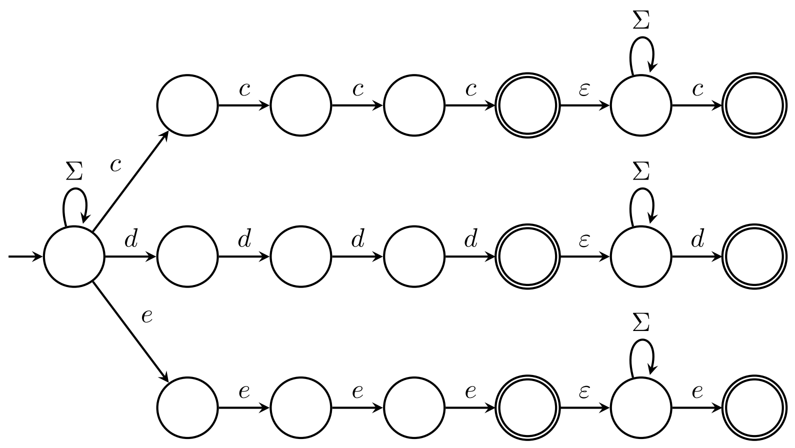
CS 181 Spring 2020 Homework Week 2 Solutions

1. An NFA for L_5 is shown on the following diagram:



We break up our computation into two phases. In the first phase, we are trying to identify a sequence $xxxx$ for some $x \in \Sigma$. So from our starting state we take a nondeterministic transition into two worlds: one in which we guess that the first character is the first x in the sequence $xxxx$, and the other in which we stay at the starting state. In this way we will eventually guess correctly and identify the first x in the sequence. For example if the input was $cdeeeede \in L_5$, then one possible pathway is that we stay at the starting state after seeing both c and d, and then transition using e into a new state. At this new state we transition forward on the same character until we've counted 4 (and if not we implicitly go to a reject state and stay there). If this is the end of the sequence we accept and are done. Otherwise we begin the next phase: we ε-transition into a temporary state where for every character we nondeterministically either remain there or transition into an accepting state upon seeing a x. Continuing the prior example, after the fourth e we ε-transition to the temporary state, loop once on d, and then transition on e to the accepting state and finish. Thus this last phase allows us to "eat up" the rest of the input and then use the final x symbol to transition into an accepting state.

2. (a) Notice that $ab \in L_1$. However, ab cannot be represented as a concatenation xy such that $x, y \in L_1$. Indeed, both x and y cannot be empty, and $x = a,\ y = b$ does not work either because $\#(a, x) \neq \#(b, x)$. Thus, $L_1 \neq L_{11}$.

   (b) Let w be any string in L_2. Notice that $\varepsilon \in L_2$. Hence, $w = w\varepsilon \in L_2 \circ L_2$, so $L_2 \subseteq L_2 \circ L_2$. On the other hand, if $w \in L_2 \circ L_2$, then by definition there are strings $x, y \in L_2$ such that $w = xy$ where $\#(a, x) = \#(b, x)$ and $\#(a, y) = \#(b, y)$. Since

$$\#(a, w) = \#(a, x) + \#(a, y) \quad \text{and} \quad \#(b, w) = \#(b, x) + \#(b, y),$$

   we notice that $\#(a, w) = \#(b, w)$, so $w \in L_2$. Thus, $L_2 \circ L_2 \subseteq L_2$. Finally, having the two inclusions, we conclude $L_2 = L_2 \circ L_2$.

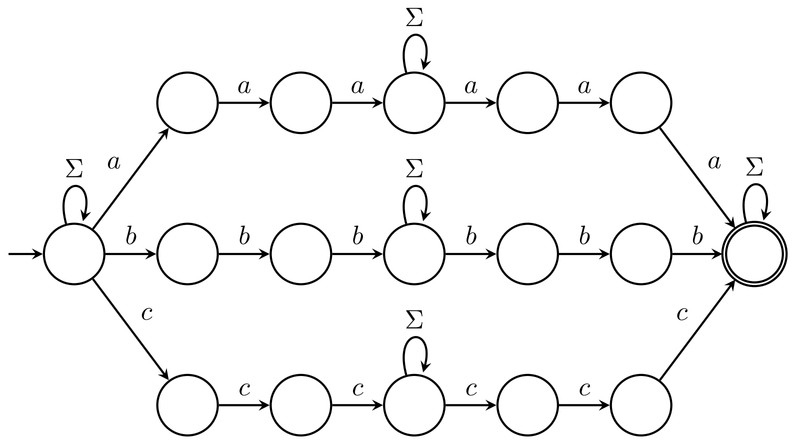
3. (a) A k-ary tree will have *at most* k children per node; intuitively we would expect fewer leaves with fewer children. It is natural to consider the extreme example of only one child per node. This is

just a line of nodes, and so there will only be one leaf at the end. Since all trees must have at least one leaf, one leaf is indeed the minimum possible number of leaves in a k-ary tree.

(b) The natural idea for the maximum number of leaves is to have every parent have the maximum number of children (i.e. k children per internal node). Because each next level of a tree should have k children for each leaf in the previous level, we might expect the form of the solution to be something like k^n.

If $n = 0$ then the tree is a single node which is a leaf, so we have $1 = k^0$ leaves. Now suppose that any maximum-leaf tree of height n edges has k^n leaves. Consider an arbitrary maximum-leaf tree T of height $n + 1$ edges, and remove all the leaves so that it is now a tree T' of height n edges. T' must also be a maximum-leaf tree; otherwise we could add another leaf to T' and hence add another leaf to T, contradicting the fact that T is a maximum-leaf tree. Since T' is a maximum-leaf tree of height n edges, by inductive hypothesis it has k^n leaves. Now if we re-add the leaf nodes that we removed from T back into T', we can count how many leaves are in T. Note that the parents of the leaves of T are the leaves of T', and that each of these parents must have a maximal number of children in order to maximize the number of leaves. Hence each of the k^n leaves of T' have k children in T, and so the number of leaves are $k \cdot k^n = k^{n+1}$.

4. An NFA for L_4 is the following automaton:



Our first observation is that for string to be accepted, it must contain two non-overlapping copies of xxx for some $x \in \Sigma$. Indeed, to reach the only accepting state, we have to use one of the three disjoint branches, and each branch contains two subpaths which have only 3 consecutive transitions on the same symbol. Thus, we accept only words in L_4.

On the other hand, if some word w is in L_4, it can be represented as $w_1xxxw_2xxxw_3$ where $x \in \Sigma$, and w_i are some (possibly empty) words. Then there is the following computational path:

(i) Use the loop on Σ in the initial state while reading w_1,

(ii) Use three transitions on x,

(iii) Use the loop on Σ in the corresponding middle state while reading w_2,

(iv) Use three transitions on x again,

(v) Use the loop on Σ in the rightmost state while reading w_3.

Since we can read the whole word and end up in the accepting state, this NFA accepts w. Thus, it accepts *exactly* words in L_4.