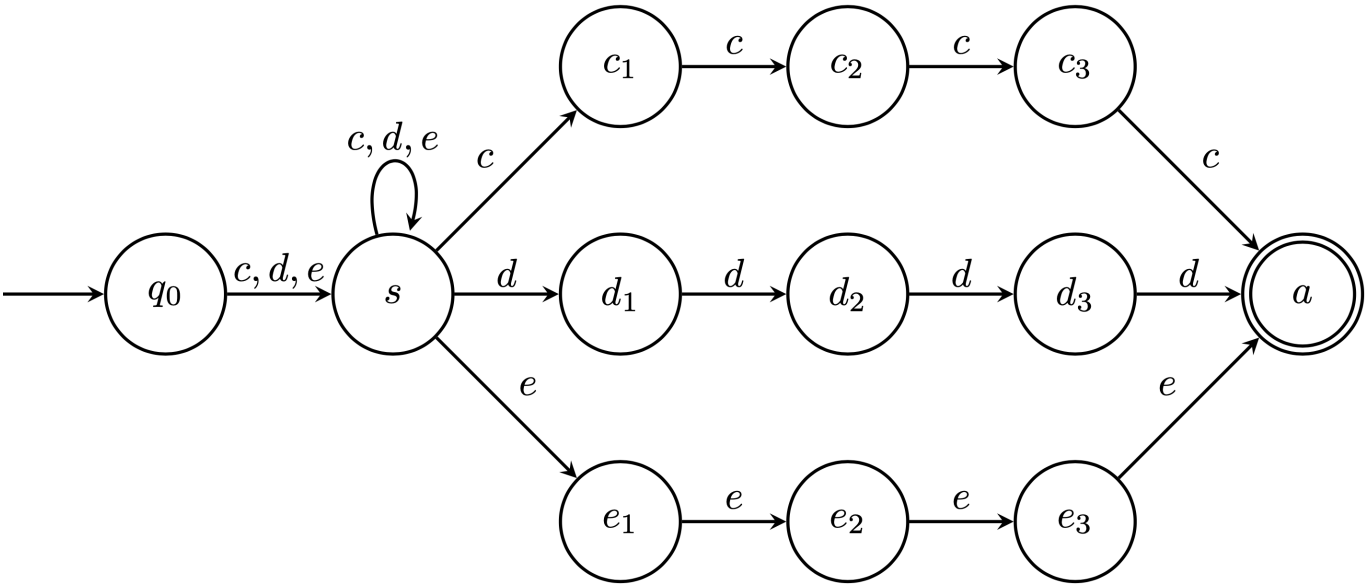
Dustin Newman

CS 181, Discussion 1C

Campbell, Mathur

Homework Two

1



The NFA above accepts the language L_5. We arrive at the start state of q_0 and, at any symbol, we transition to state s. Note that we have no ϵ transitions and thus reject the empty string. Once at s, we have satisfied our condition that $y \in \Sigma^+$. y can be one of two things: one character or greater. If y is just one character, then we follow the transition to the appropriate state for the following input symbol (i.e. if the input symbol after y is "c" then we transition to c_1). If y is greater than one character, however, we can loop on s until y is terminated. This uses the "guess and check" view of NFAs, which will "guess" that we are now done with y and ready to process substring $xxxx$ and then "check" with the appropriate following states (i.e. one of $c_1 \ldots c_3$, $d_1 \ldots d_3, e_1 \ldots e_3$).

Once we have verified through those states that we have the substring $xxxx$, we are in the sole accepting state a. If there is any other character remaining, then our current thread of computation terminates because a has no outgoing edges for any symbol in the alphabet.

2

a

$L_1 \neq L_{11}$ because there exists some string which is in L_1 which cannot be in L_{11}, namely $w = \text{“ab”}$. w can only be in L_{11} in one of two ways: either from the combination of "a" from L_1 and "b" from L_1 or from the combination of "ab" from L_1 and ϵ from L_1.

Neither "a" nor "b" can be L_1 because neither string contains an equal number of a's and b's i.e. $\#(a, \text{“}a\text{”}) \neq \#(b, \text{“}a\text{”})$ and $\#(a, \text{“}b\text{”}) \neq \#(b, \text{“}b\text{”})$.

While "ab" can be in L_1 (as it contains the same number of a's and b's), the empty string ϵ cannot, since L_1 is defined to be any string w such that w is in Σ^+. However, Σ^+ cannot contain the empty string and thus there is no w which can be the empty string.

$\therefore L_1 \neq L_{11}$

b

$L_2 = L_{22}$ because there exists no string $w \in L_2$ which is not also in L_{22} and vice versa. I will prove this by induction, where the base case is $w = \epsilon \in \Sigma^*$.

When $w = \epsilon$, $w \in L_2$ as L_2 is defined as all strings within Σ^* (where $\#(a, w) = \#(b, w)$). If we take the concatenation of w across both L_2, then we derive $w \circ w = \epsilon\epsilon = \epsilon \in L_{22}$.

The inductive step is based on the observation that, if $w \neq \epsilon$, it must be some string consisting of "a" and "b" in equal frequency. Let's take "ab" for example. "ab" $\in L_2$ and thus it is also in L_{22} as the result of concatenation with the empty string: $\text{“}ab\text{”} \circ \epsilon = \text{“}ab\text{”} \in L_{22}$. This same logic applies to every other w: we can simply take the $w \in L_2$ and concatenate with ϵ to have the $w \in L_{22}$, because $w \circ \epsilon = w$. We can summarize this as:

$$L_2 = \{w \in \Sigma^* \mid \#(a, w) = \#(b, w)\}$$
$$L_{22} = L_2 \circ L_2 = \{w \in \Sigma^* \mid \#(a, w) = \#(b, w)\} \circ \epsilon = L_2$$

3

a

The fewest number of leaf nodes in T is **one**. This is formed by having one child for every node. The definition of a "k-ary" tree is that each node has *at most* k children, not exactly k children. Thus, a tree can be a k-ary tree with one child per node.

In terms of n, the fewest number of leaf nodes is $n + 1 - n$. For all $n + 1$ nodes in the longest path from the root to the leaf, we create an edge between each pair of nodes. For each edge we create of the form (u, v), u is no longer a leaf, as it has a directed outgoing edge. Because there are n such pairs, we remove n nodes from the set of leaf nodes. The last remaining v is the sole leaf node in the path.

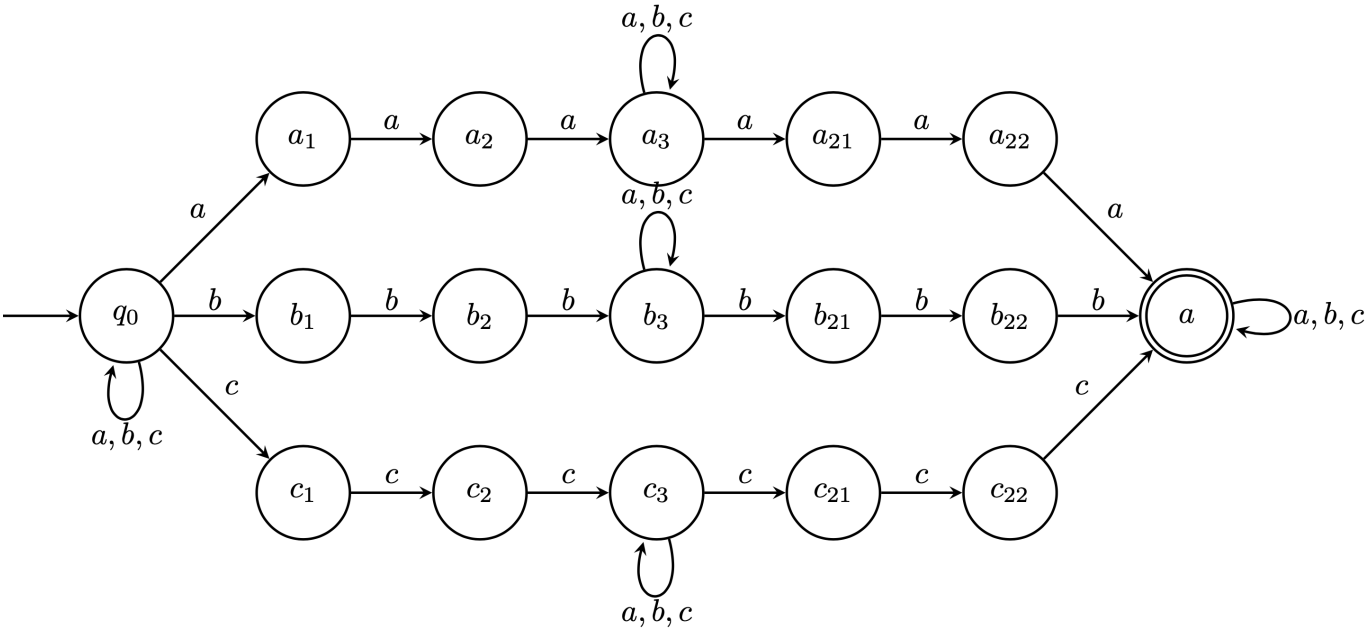
b

The largest number of leaf nodes possible is k^n. For a proof by induction, we consider the base case of $n := 1$. Because we consider the largest number of leaf nodes possible, the root node has all k children possible. Because $n := 1$, there is one edge on the longest path from the root to any leaf. Since there are k children, each having an edge from the root node and no outgoing edges, there are k leaf nodes. Therefore, our base case is proven as $k^n = k^1 = k$.

For our inductive step, we assume that there are k^n leaf nodes when the height of the tree is n. We need to prove that there are k^{n+1} leaf nodes when the height is $n + 1$.

First, take the tree n nodes high. For all k^n leaf nodes at height n, we create children. Since each node must have k children, there are k times as many nodes at height $n + 1$ than at height n. Since there were k^n nodes at height n, we now have $k^n \times k$ nodes at height $n + 1$. $k^n \times k = k^{n+1}$. Therefore there are k^{n+1} nodes at height $n + 1$.

4



We use the "guess and check" view of NFA to describe this machine. We remain at the initial state q_0 until we guess that we are on the beginning of the first substring consisting of three identical, consecutive symbols. When we do correctly guess, we move to the corresponding state x_1 (i.e. if we read an "a," we move to a_1; if we read a "b," we move to b_1, etc.). From there, we use the property of NFA which allow us to define only the relevant outgoing edges. This allows us to block any false starts or incorrect guesses (i.e. "kill" the current thread of computation). Once we read our first substring, we are in the x_3 state (where $x := a \mid b \mid c$). We once again use our "guess and check" view, staying on x_3 until we guess that we are on the beginning of the second substring. Once we have guessed, we once again block on any false starts. Once we check that the second substring is complete, we arrive at our final, accepting state a, where we loop forever, as there are no positional requirements on L_4 as there were in L_5 (from Problem One).

I chose to use an NFA for this problem because the nondeterministic options (multiple edges with the same label) allowed me to more tersely describe the machine. All I had to show above was that at least one thread correctly reaches a, not that we always reach a for all threads. If I had chosen a DFA, I would have had to have edges from each $x_1 \ldots x_3$ back to $q0$ for all symbols $\neq x$ (e.g. if we read a "b" on a_2, we have to move back to q_0).