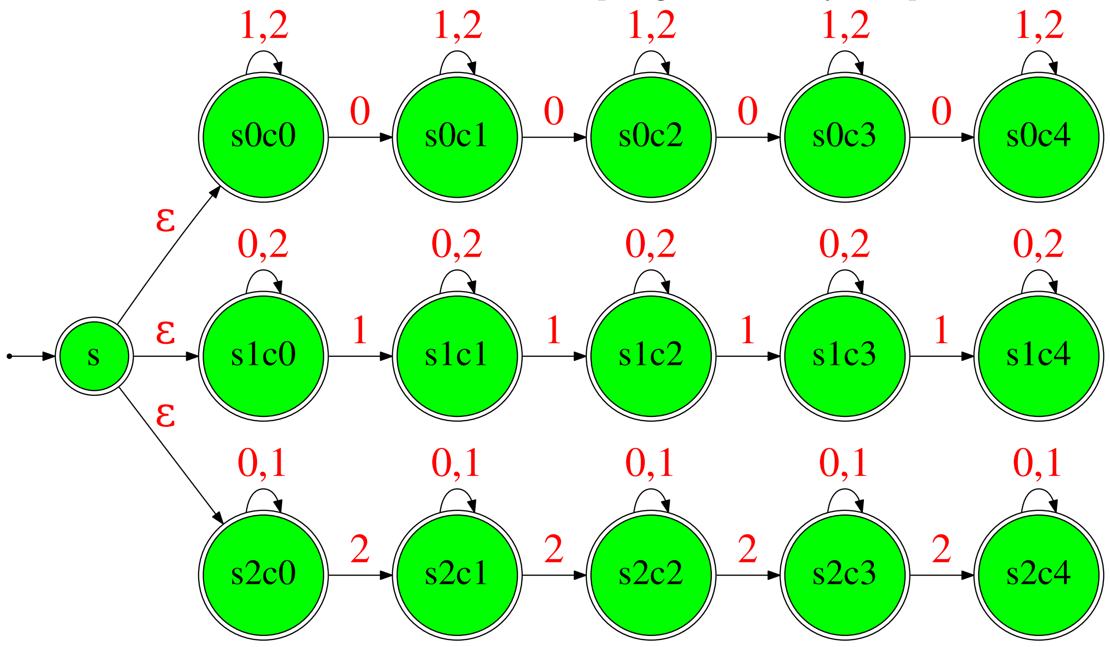
7 (100 PTS.) Draw me a sheep.

For each of the following languages in **7.A.–7.C.**, draw an NFA that accepts them. Your automata should have a small number of states. Provide a short explanation of your solution, if needed.

7.A. All strings in $\{0, 1, 2\}^*$ such that at least one of the symbols 0, 1, or 2 occurs at most 4 times. (Example: 1200201220210 is in the language, since 1 occurs 4 times.)

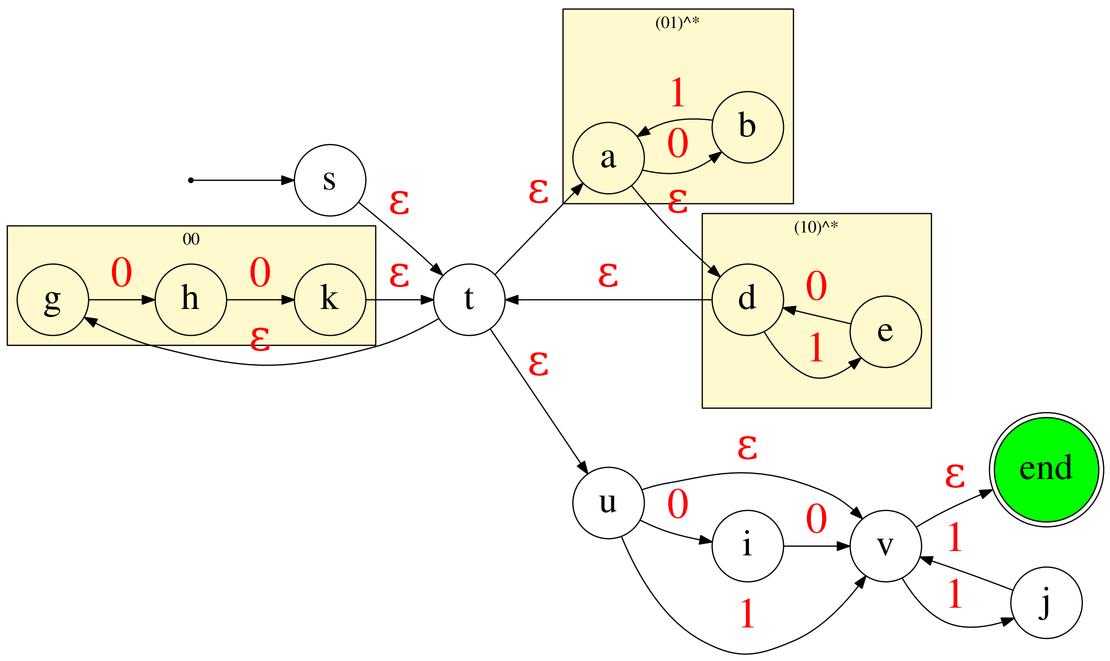
Solution:

Explanation: Building an automata that counts the number of, say, 0s, and reject if the number exceeds 4 is easy. Next build such automatas for 1 and 2, and observe that all you need to do is ε transition into one of these three. Thus resulting in the following automata. All states in this automata are accepting. It is a very acceptable automata.



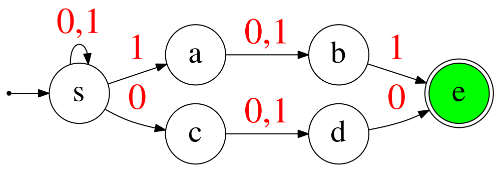
7.B. $\left((01)^*(10)^* + 00\right)^* \cdot (1 + 00 + \varepsilon) \cdot (11)^*$.

Solution:



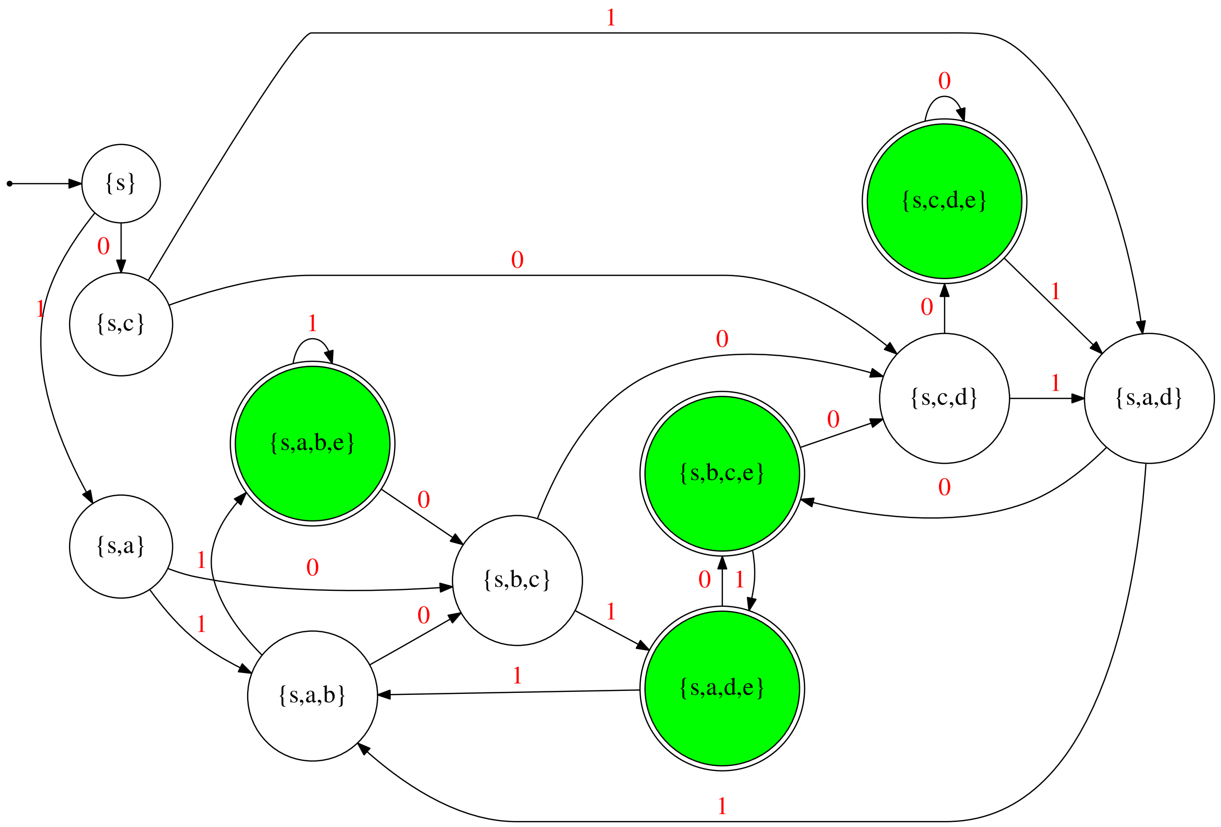
7.C. All strings in $\{0, 1\}^*$ such that the last symbol is the same as the third last symbol. (Example: 1100101 is in the language, since the last and the third last symbol are 1.)

Solution:



7.D. Use the power-set construction (also called subset construction) to convert your NFA from **7.C.** to a DFA. You may omit unreachable states.

Solution:



8 (100 PTS.) Fun with parity.

Given $L \subseteq \{0,1\}^*$, define $\mathit{even}_0(L)$ to be the set of all strings in $\{0,1\}^*$ that can be obtained by taking a string in L and inserting an even number of 0's (anywhere in the string). Similarly, define $\mathit{odd}_0(L)$ to be the set of all strings x in $\{0,1\}^*$ that can be obtained by taking a string in L and inserting an odd number of 0's.

(Example: if $01101 \in L$, then $01010000100 \in \mathit{even}_0(L)$.)

(Another example: if L is 1^*, then $\mathit{even}_0(L)$ can be described by the regular expression $(1^*01^*0)^*1^*$.)

You will prove that if $L \subseteq \{0,1\}^*$ is regular, then $\mathit{even}_0(L)$ and $\mathit{odd}_0(L)$ are regular. Specifically, given a regular expression r, you will describe a recursive algorithm to construct regular expressions for $\mathit{even}_0(L(r))$ and $\mathit{odd}_0(L(r))$.

8.A. For each of the base cases of regular expressions $\emptyset$, ε, 0, and 1, give regular expressions for $\mathit{even}_0(L(r))$ and $\mathit{odd}_0(L(r))$.

Solution:

$\mathit{even}_0(L(\emptyset)) = \emptyset$

$\mathit{even}_0(L(\varepsilon)) = (00)^*$

$\mathit{even}_0(L(0)) = 0(00)^*$

$\mathit{even}_0(L(1)) = (00)^*1(00)^* + (00)^*010(00)^*$

$\mathit{odd}_0(L(\emptyset)) = \emptyset$

$odd_0(L(\varepsilon)) = 0(00)^*$
$odd_0(L(0)) = 00(00)^*$
$odd_0(L(1)) = (00)^*01(00)^* + (00)^*10(00)^*$

8.B. Given regular expressions for $e_j = even_0(L(r_j))$ and $o_j = odd_0(L(r_j))$ for $j \in \{1, 2\}$, give regular expressions for

(i) $L_i = even_0(L(r_1 + r_2))$

Solution: $e_1 + e_2$.
Every word in L_i is a modification of a word in e_1 or a word in e_2.

(ii) $L_{ii} = odd_0(L(r_1 + r_2))$

Solution: $o_1 + o_2$.
Every word in L_{ii} is a modification of a word in o_1 or a word in o_2.

(iii) $L_{iii} = even_0(L(r_1 r_2))$

Solution: $e_1e_2 + o_1o_2$
Let $w \in L_{iii}$. It had risen out of two words $x_1 \in L(r_1)$ and $x_2 \in L(r_2)$. As such, break w, after x_1 is completed. That is $w = w_1w_2$, where w_i is created by inserting 0s to x_i, for $i = 1, 2$. Let $\Delta_i = \#_0(w_i) - \#_0(x_i)$, for $i = 1, 2$, and let $\Delta = \Delta_1 + \Delta_2$. By definition Δ is even.

Now, either Δ_1 is even, but then Δ_2 is even, which means that $w_1 \in e_1$ and $w_2 \in e_2$. Or, Δ_1 is odd, but then Δ_2 is odd, which implies that $w_1 \in o_1$ and $w_2 \in o_2$.

(iv) $odd_0(L(r_1 r_2))$

Solution: $e_1o_2 + o_1e_2$.
This follows by applying the same logic as above.

(v) $even_0(L(r_1^*))$

Solution:

$$E = (00)^* + e_1^*\Big(e_1^* o_1 e_1^* o_1 e_1^*\Big)^* = (00)^* + e_1^*\Big(o_1 e_1^* o_1 e_1^*\Big)^*.$$

Consider a string $w \in E$, and assume that it is the result of adding 0s to a string $x = x_1 \ldots x_m$, where $x_i \in L(r_1)$. Break w into substrings $w = w_1 \ldots w_m$, where w_i corresponds to x_i, for all i.

The string w_i is ***odd*** if $\#_0(w_i) - \#_1(x_i)$ is odd. Clearly, the number of strings in $w_1, \ldots, w_m$ that are odd is even. Every two consecutive odd words might be separated by a run of even words (i.e., a string in e_1^*). There might also be a prefix or a suffix of even words. And we can repeat this construction as many times as we want. The only case not covered by this logic is when all words are even, which is why we added e_1^* in the beginning of the expression.

The $(00)^*$ corresponds to $\varepsilon \in r_1^*$. Note that $\varepsilon \in r_1^*$, even if $L(r_1) = \emptyset$. So we have to handle this exceptional case separately.

(vi) $odd_0(L(r_1^*))$

Solution: $0(00)^* + Eo_1E$

Repeat the same argument as above, and observe that there must be at least one odd word. This odd word is prefixed by a word that has even number of zeroes added (and is thus in $L(E)$), and similarly it has a suffix with the same property.

Give brief justification of correctness for each of the above.

8.C. (10 PTS.) Using the above, describe (shortly) a recursive algorithm that given a regular expression r, outputs a regular expression for $even_0(L(r))$ (similarly describe the algorithm for computing $odd_0(L(r))$).

Solution:

We have two functions **even**(r) and **odd**(r).

even(r): The algorithm inspects r. If r is one of the base cases studied in (A), it returns the expression as specified in your answer to part (A). Otherwise, r must be a compound expression. There are several possibilities, and it returns the answer using the answers from part (B):

```
• if $r = r_1 + r_2$ then return even$(r_1)$ | " + " | even$(r_2)$.
  // Here | used to designate string concatenation
• if $r = r_1 r_2$ then return even$(r_1)$ | even$(r_2)$ | " + " | odd$(r_1)$ | odd$(r_2)$ .
• if $r = (r_1)^*$ then
  $e_1 \leftarrow$ even$(r_1)$
  $o_1 \leftarrow$ odd$(r_1)$
  // Abusing notations, we use the math expression, but you know how to
  // convert it into a string, right?
  return $(00)^* + (e_1)^*(o_1(e_1)^*o_1(e_1)^*)^*$.
```

The algorithm **odd** works in a similar fashion, and we omit the details.

9 (100 PTS.) "+1".

Let binary(i) denote the binary representation of a positive integer i. (Note that the string binary(i) must start with a 1.)

Given a language $L \subseteq \{0,1\}^*$, define $\mathrm{INC}(L) = \{\mathrm{binary}(i+1) \mid \mathrm{binary}(i) \in L\}$. For the time being assume that L does not contain any string of 1^*.

(Example: for $L = \{100, 101011, 111\}$, we have $\mathrm{INC}(L) = \{101, 101100, 1000\}$.)

9.A. (30 PTS.) Given a DFA $M = (Q, \Sigma, \delta, s, A)$ for L, describe **informally** (in a few sentences) how to construct an NFA M_w for INC(L).

Solution:

Let us start with the easier case where $|\mathrm{binary}(i)| = |\mathrm{binary}(i+1)|$. In this case, binary$(i)$ can be written as $w01^i$, and binary$(i+1) = w10^i$. So, all we have to do is to guess where the suffix 01^i begins, and then we should just do the following – read the guessed 1 that moves us to flipped automata, where 0 is interpreted as being 1 (for the original automata). Notice that in this flipped automata, there are transitions with 0 – no other transitions are possible.

The original automata would be in the bottom layer, and the new flipped automata would be in the top layer. The only accept state would be in the top layer, which would be a copy of the original accept state.

9.B. (30 PTS.) Given a DFA $M = (Q, \Sigma, \delta, s, A)$ for L, describe **formally** how to construct an NFA M' for $\mathrm{INC}(L)$.

Solution:

We first duplicate the current DFA into two copies, the second copy being this 1^* "tail" detected. As such, the new set of states is

$$Q' = Q \times \{b, t\}.$$

All the states (q, b) behave in the same way as before, except that they might decide to jump to the second layer when they read a 1.

$$\forall((q,\ell) \in Q' \qquad \delta_2\Big(\big(q,\ell\big), c\Big) = \begin{cases} \{(\delta(q,0), b)\} & c = 0, \ell = b \\ \Big\{\big(\delta(q,1), b\big)\Big\} \cup \Big\{\big(\delta(q,0), t\big)\Big\} & c = 1, \ell = b \\ \{(\delta(q,1), t)\} & c = 0, \ell = t \\ \emptyset & c = 1, \ell = t. \end{cases}$$

Confusingly, in the top level we flip the behavior (i.e., if you get 0 you behave like you got a 1), and keep only the transitions of 0. (Remember: For NFA you do not need to specify transition for all input characters.)

The set of accepting states is:

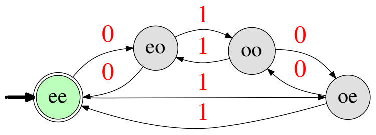
$$A' = \{(q,t) \mid q \in A\}.$$

The start state of the new NFA is (s, b).

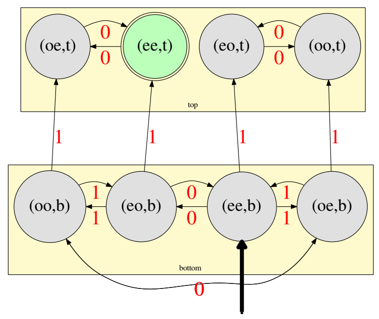
The resulting NFA is

$$N = (Q', \Sigma, \delta_2, (s,b), A')$$

As an example, consider the current DFA:



And the resulting NFA:



9.C. (30 PTS.) Prove formally the correctness of your construction from (**9.B.**). That is, prove that $\mathrm{INC}(L) = L(M')$.

Solution:

Lemma 3.1. *Let M be a DFA, and let N be the resulting NFA constructed above. We have that*

$$L(N) = \{ \mathit{binary}(i+1) \mid \mathit{binary}(i) \in L \}.$$

Proof: We have to prove inclusion in both directions.

increment$(L(M)) \subseteq L(N)$: If $w = \mathrm{binary}(i) \in L$, then (by assumption), we can write $w = p01^j$ for some j (that can be zero). As such, let

$$\begin{aligned} q_0 &= s, \\ q_1 &= \delta^*(q_0, p), \\ q_2 &= \delta(q_1, 0), \\ q_3 &= \delta^*(q_2, 1^j). \end{aligned}$$

Clearly, $q_3 \in A$, as $w \in L(M)$. It is easy to verify that we have

$$\begin{aligned} q'_0 &= (s, b), \\ q'_1 &= (q_1, b) \in \delta_1^*\big((s,b), p\big) \\ q'_2 &= (q_2, t) \in \delta_1\big(q'_1, 1\big) \\ q'_3 &= (q_3, t) \in \delta_1^*(q'_2, 0^j). \end{aligned}$$

Since $q_3 \in A$. it follows that $q'_3 \in A'$, and $\mathrm{binary}(i+1) = p10^j \in L(N)$.

$L(N) \subseteq$ increment$(L(M))$: We also need to prove that for any $w \in L(N)$, with $w = \mathrm{binary}(i)$, we have that $\mathrm{binary}(i-1) \in L(M)$.

The proof is similar to the above. Let $w = p10^j$. Let $k = |p|$. Consider an execution of the NFA N that accepts w. It must be of the form

$$\begin{aligned} q'_0 &= (s, b), \\ q'_k &= (q_k, b) \in \delta_1^*(q_0, p), \\ q'_{k+1} &= (q_{k+1}, t) \in \delta_1(q'_k, 1), \\ q'_{|w|} &= (q_{|w|}, t) \in \delta_1^*(q'_{k+1}, 0^j), \end{aligned}$$

By assumption on L, $p \neq \epsilon$, and it is now straightforward, to show that $p01^j \in L(M)$. Indeed, we have

$$\begin{aligned} q_0 &= s, \\ q_k &= \delta^*(q_0, p), \\ q_{k+1} &= \delta(q_k, 0), \\ q_{|w|} &= \delta^*(q_{k+1}, 1^j), \end{aligned} \qquad \blacksquare$$

We conclude that $\mathrm{binary}(i-1) = \delta^*(s, p01^j) = q_{|w|} \in A$.

9.D. (10 PTS.) Describe formally how to modify the construction of M' from above, to handle that general case (without the above assumption) that L might also contain strings of the form 1^*. You do not need to provide a proof of correctness of the new automata.

Solution:

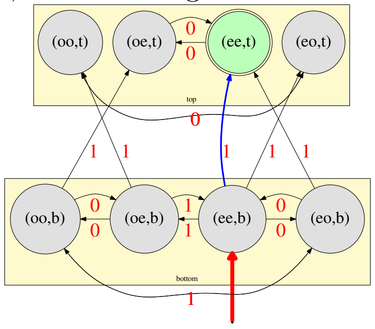
We use here the notations from (**9.B.**). The case that the word in L is of the form 1^* corresponds to the case that $|\text{binary}(i+1)| = |\text{binary}(i)| + 1$. Specifically, $\text{binary}(i) = 1^j$, and $\text{binary}(i+1) = 10^j$. But for this, all we need to do is to transition from (s,b) to (s,t) on the letter 1. We modify the transition function to be

$$\forall (q,\ell) \in Q' \qquad \delta_1\Big((q,\ell),c\Big) = \begin{cases} \delta_2\Big((q,\ell),c\Big) \cup \{(s,t)\} & c = 1, \ell = b, q = s \\ \delta_2\Big((q,\ell),c\Big) & \text{otherwise.} \end{cases}$$

The resulting NFA is

$$N = (Q', \Sigma, \delta_1, (s,b), A')$$

For the example above, the resulting NFA is:


(oo,t)
(oe,t)
(ee,t)
(eo,t)
top
(oo,b)
(oe,b)
(ee,b)
(eo,b)
bottom
0
1