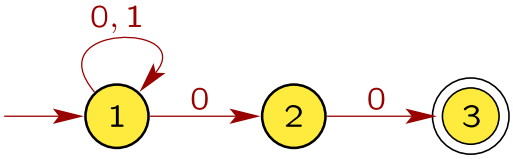
CS 341: Foundations of Computer Science II
Prof. Marvin Nakayama

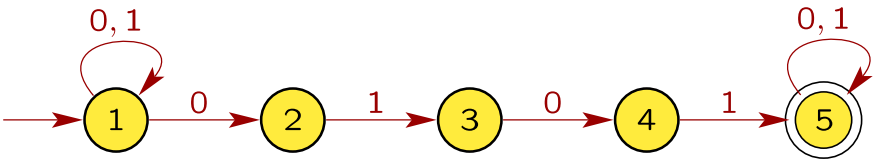
Homework 3 Solutions

1. Give NFAs with the specified number of states recognizing each of the following languages. In all cases, the alphabet is $\Sigma = \{0, 1\}$.

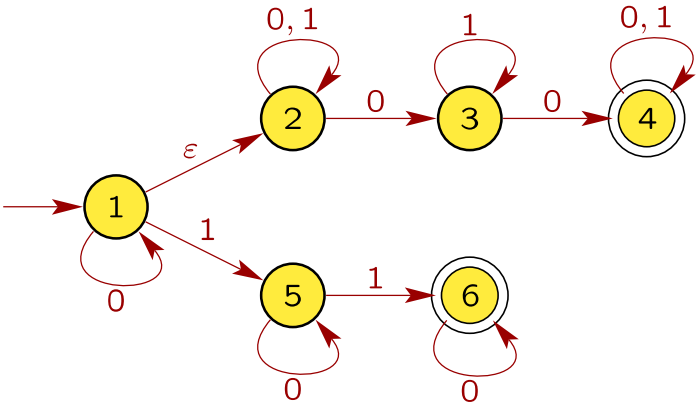
   (a) The language $\{ w \in \Sigma^* \mid w \text{ ends with } 00 \}$ with three states.



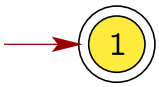
   (b) The language $\{ w \in \Sigma^* \mid w \text{ contains the substring } 0101, \text{ i.e., } w = x0101y \text{ for some } x, y \in \Sigma^* \}$ with five states.



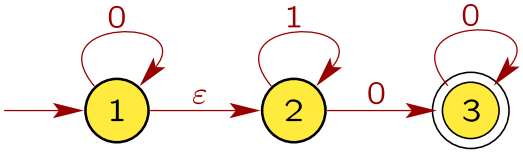
   (c) The language $\{ w \in \Sigma^* \mid w \text{ contains at least two } 0\text{s, or exactly two } 1\text{s} \}$ with six states.



   (d) The language $\{\varepsilon\}$ with one state.

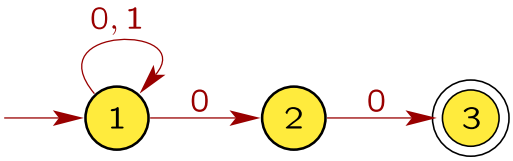


(c) The language $0^*1^*0^*0$ with three states.

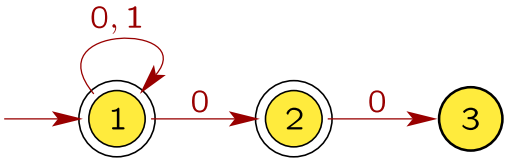


2. (a) Show by giving an example that, if M is an NFA that recognizes language C, swapping the accept and non-accept states in M doesn't necessarily yield a new NFA that recognizes $\overline{C}$.

Answer:
The NFA M below recognizes the language $C = \{ w \in \Sigma^* \mid w \text{ ends with } 00 \}$, where $\Sigma = \{0, 1\}$.



Swapping the accept and non-accept states of M gives the following NFA M':



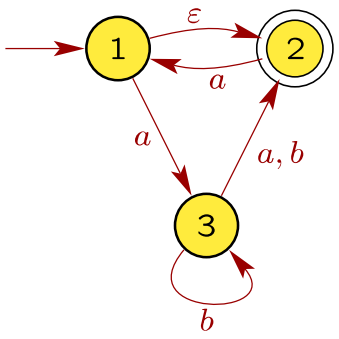
Note that M' accepts the string $100 \notin \overline{C} = \{ w \mid w \text{ does not end with } 00 \}$, so M' does not recognize the language $\overline{C}$.

(b) Is the class of languages recognized by NFAs closed under complement? Explain your answer.

Answer:
The class of languages recognized by NFAs is closed under complement, which we can prove as follows. Suppose that C is a language recognized by some NFA M, i.e., $C = L(M)$. Since every NFA has an equivalent DFA (Theorem 1.39), there is a DFA D such that $L(D) = L(M) = C$. By problem 3 on Homework 2, we then know there is another DFA $\overline{D}$ that recognizes the language $\overline{L(D)}$. Since

every DFA is also an NFA, this then shows that there is an NFA, in particular $\overline{D}$, that recognizes the language $\overline{C} = \overline{L(D)}$. Thus, the class of languages recognized by NFAs is closed under complement.

3. Use the construction given in Theorem 1.39 to convert the following NFA N into an equivalent DFA.



Answer: Let NFA $N = (Q, \Sigma, \delta, 1, F)$, where $Q = \{1, 2, 3\}$, $\Sigma = \{a, b\}$, 1 is the start state, $F = \{2\}$, and the transition function δ as in the diagram of N. To construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ that is equivalent to NFA N, first we compute the ε-closure of every subset of $Q = \{1, 2, 3\}$.

Set $R \subseteq Q$	ε-closure $E(R)$
$\emptyset$	$\emptyset$
$\{1\}$	$\{1, 2\}$
$\{2\}$	$\{2\}$
$\{3\}$	$\{3\}$
$\{1, 2\}$	$\{1, 2\}$
$\{1, 3\}$	$\{1, 2, 3\}$
$\{2, 3\}$	$\{2, 3\}$
$\{1, 2, 3\}$	$\{1, 2, 3\}$

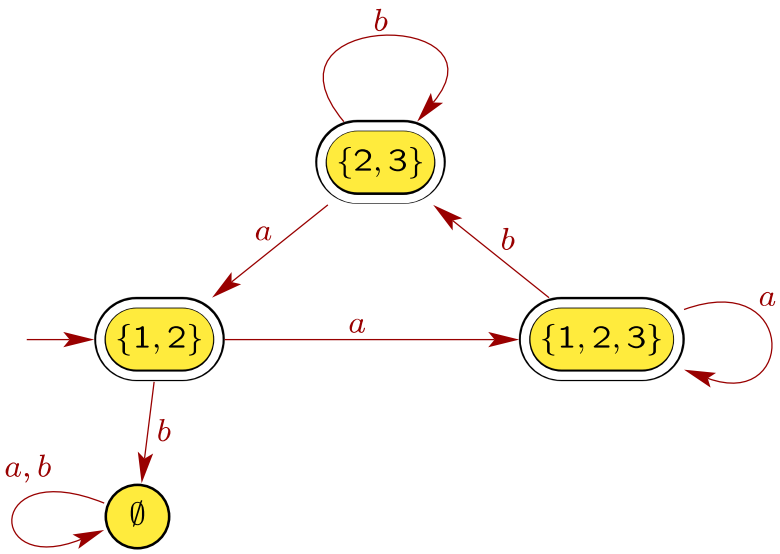
Then define $Q' = \mathcal{P}(Q)$, so

$$Q' = \{\ \emptyset,\ \{1\},\ \{2\},\ \{3\},\ \{1,2\},\ \{1,3\},\ \{2,3\},\ \{1,2,3\}\ \}.$$

The start state of M is then $E(\{1\}) = \{1, 2\}$. The set of accept states of M is

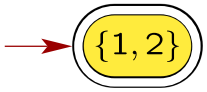
$$F' = \{\ \{2\},\ \{1,2\},\ \{2,3\},\ \{1,2,3\}\ \}.$$

We define the transitions in the DFA M as in the following diagram:



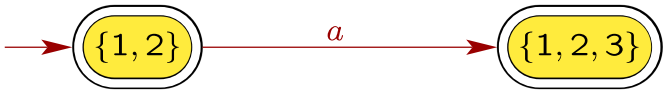
Note that we left out some of the states (e.g., $\{1\}$) in $\mathcal{P}(Q)$ from our diagram of the DFA M since they are not accessible from the start state $\{1, 2\}$. Also, we had to add an arc from state $\emptyset$ to itself labelled with "a, b" so that this state has an arc leaving it corresponding to each symbol in the alphabet Σ, which is a requirement for any DFA.

The algorithm given in the notes and textbook will always correctly construct an equivalent DFA from a given NFA, but we don't always have to go through all the steps of the algorithm to obtain an equivalent DFA. For example, on this problem, we begin by figuring out what states the NFA can be in without reading any symbols. In this case, this is $E(\{1\}) = \{1, 2\}$ since 1 is the starting state of the NFA, and the NFA can jump from 1 to 2 without reading any symbols by taking the ε-transition. Thus, we first create a DFA state corresponding to the set $\{1, 2\}$:

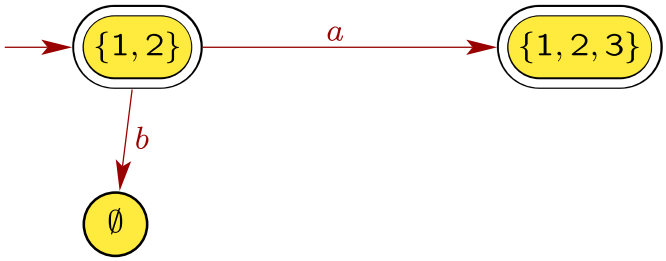


The state $\{1, 2\}$ is the start state of the DFA since this is where the NFA can be without reading any symbols. The state $\{1, 2\}$ is also an accepting state for the DFA since it contains 2, which is accepting for the NFA.

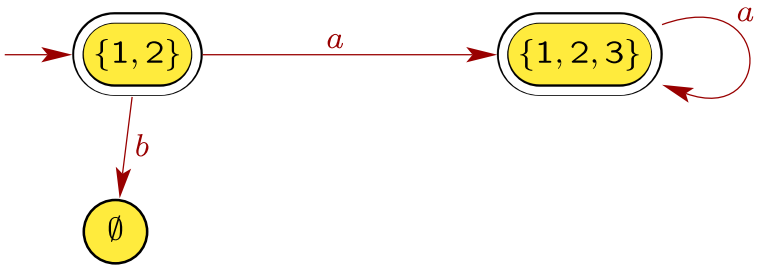
Now for DFA state $\{1, 2\}$, determine where the NFA can go on an a from each NFA state within this DFA state, and where the NFA can go on a b from each NFA state within this DFA state. On an a, the NFA can go from state 1 to state 3; also, the NFA can go from state 2 to 1, and then it also can go further from 1 to 2 on the ε. So from NFA states 1 and 2 on an a, the NFA can end up in states 1, 2, and 3, so draw a transition in the DFA from state $\{1, 2\}$ to a new state $\{1, 2, 3\}$, which is an accepting state since it contains $2 \in F$:



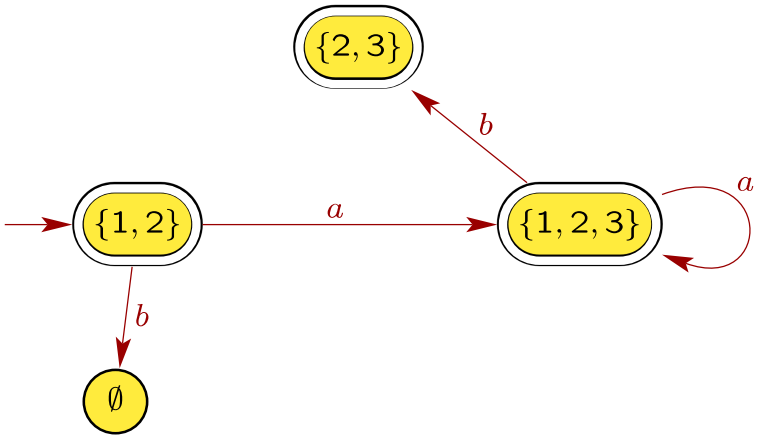
Similarly, to determine where the DFA moves on b from DFA state $\{1, 2\}$, determine all the possibilities of where the NFA can go from NFA states 1 and 2 on b. From state 1, the NFA can't go anywhere on a b; also, the NFA can't go anywhere from state 2 on b. Thus, the NFA can't go anywhere from states 2 and 3 on a b, so we add a b-edge in the DFA from state $\{1, 2\}$ to a new DFA state $\emptyset$, which is not accepting since it contains no accept states of the NFA:



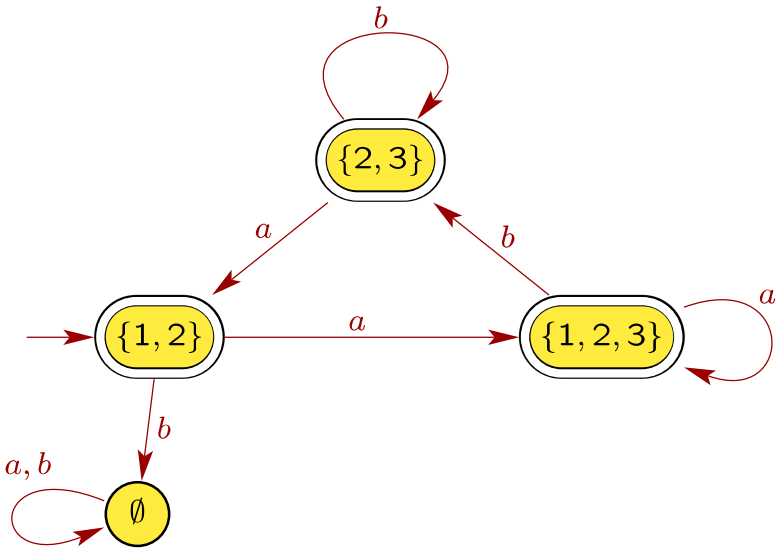
Now every time we add a new DFA state, we have to determine all the possibilities of where the NFA can go on an a from each NFA state within that DFA state, and where the NFA can go on a b from each NFA state within that DFA state. For DFA state $\{1, 2, 3\}$, we next determine where the NFA can go on an a from each of the NFA states 1, 2 and 3. From NFA state 1, the NFA on an a can go to NFA state 3; from NFA state 2, the NFA on an a can go to NFA state 1, and then it can also further jump to 2 on ε; from NFA state 3, the NFA on an a can go to NFA state 2. Thus, if the NFA is in states 1, 2 and 3, it can go on an a to states 1, 2 and 3, so we add to the DFA an a-edge from $\{1, 2, 3\}$ to $\{1, 2, 3\}$.



Now we determine where the b-edge from DFA state $\{1, 2, 3\}$ goes to. To do this, we examine what happens to the NFA from states 1, 2 and 3 on a b. If the NFA is in state 1, then there is nowhere to go on a b; if the NFA is in state 2, then there is nowhere to go on a b; if the NFA is in state 3, then the NFA can go to 2 or 3 on b. Hence, if the NFA is in states 1, 2 and 3, the NFA on b can end in states 2 and 3. Thus, in the DFA, draw an edge from state $\{1, 2, 3\}$ to a new state $\{2, 3\}$, which is accepting since it contains $2 \in F$:



Now do the same for DFA states $\{2, 3\}$ and $\emptyset$. If any new DFA states arise, then we need to determine the a and b transitions out of those states as well. We stop once every DFA state has an a-transition and a b-transition out of it. Accepting states in the DFA are any DFA states that contain at least one accepting NFA state. We eventually end up with the DFA below as before:



For the DFA state $\emptyset$, there are no versions of the NFA currently active, i.e., all "threads" have "crashed," so the NFA cannot proceed and the input string will not be accepted. However, according to the definition of a DFA, each state must have edges leaving it corresponding to each symbol in the alphabet Σ. Thus, we add a loop from the DFA state $\emptyset$ back to itself labeled with Σ, which in our case is a, b.

4. Give regular expressions that generate each of the following languages. In all cases, the alphabet is $\Sigma = \{a, b\}$.

(a) The language $\{ w \in \Sigma^* \mid |w| \text{ is odd} \}$.

Answer: $(a \cup b)((a \cup b)(a \cup b))^*$

(b) The language $\{ w \in \Sigma^* \mid w \text{ has an odd number of } a\text{'s} \}$.

Answer: $b^*a(ab^*a \cup b)^*$

(c) The language $\{ w \mid w \text{ contains at least two } a\text{'s, or exactly two } b\text{'s} \}$.

Answer: $b^*ab^*a(a \cup b)^* \ \cup \ a^*ba^*ba^*$

(d) The language $\{ w \in \Sigma^* \mid w \text{ ends in a double letter} \}$. (A string contains a *double letter* if it contains aa or bb as a substring.)

Answer: $(a \cup b)^*(aa \cup bb)$

(e) The language $\{ w \in \Sigma^* \mid w \text{ does not end in a double letter} \}$.

Answer: $\varepsilon \cup a \cup b \cup (a \cup b)^*(ab \cup ba)$

(f) The language $\{ w \in \Sigma^* \mid w \text{ contains exactly one double letter} \}$. For example, $baaba$ has exactly one double letter, but $baaaba$ has two double letters.

Answer: $(\varepsilon \cup b)(ab)^*aa(ba)^*(\varepsilon \cup b) \ \cup \ (\varepsilon \cup a)(ba)^*bb(ab)^*(\varepsilon \cup a)$

5. Suppose we define a restricted version of the Java programming language in which variable names must satisfy all of the following conditions:

- A variable name can only use Roman letters (i.e., `a`, `b`, ..., `z`, `A`, `B`, ..., `Z`) or Arabic numerals (i.e., `0`, `1`, `2`, ..., `9`); i.e., underscore and dollar sign are not allowed.
- A variable name must start with a Roman letter: `a`, `b`, ..., `z`, `A`, `B`, ..., `Z`
- The length of a variable name must be no greater than 8.
- A variable name cannot be a keyword (e.g., if). The set of keywords is finite.

Let L be the set of all valid variable names in our restricted version of Java.

(a) Let L_0 be the set of strings satisfying the first 3 conditions above; i.e., we do not require the last condition. Give a regular expression for L_0.

Answer: To simplify the regular expression, we define

$$\begin{aligned} \Sigma_1 &= \{\texttt{a}, \texttt{b}, \ldots, \texttt{z}, \texttt{A}, \texttt{B}, \ldots, \texttt{Z}\} \\ \Sigma_2 &= \{\texttt{0}, \texttt{1}, \texttt{2}, \ldots, \texttt{9}\}. \end{aligned}$$

Then a regular expression for L_0 is

$$\Sigma_1 \underbrace{(\Sigma_1 \cup \Sigma_2 \cup \varepsilon) \cdots (\Sigma_1 \cup \Sigma_2 \cup \varepsilon)}_{7 \text{ times}}.$$

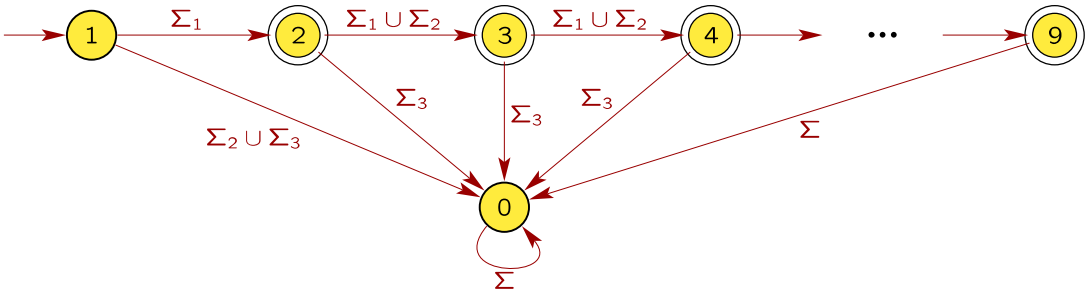
Note that by including the ε in each of the last parts, we can generate strings that have length strictly less than 8.

(b) Prove that L has a regular expression, where L is the set of strings satisfying all four conditions.

Answer: We proved in Homework 1, problem 4(b), that L is finite. Thus, L is regular, so it has a regular expression. Although the problem didn't ask for it, we can write a regular expression for L by listing all of the strings in L and putting a $\cup$ in between each pair of consecutive strings. This works because L is finite.

(c) Give a DFA for the language L_0 in part (a), where the alphabet Σ is the set of all printable characters on a computer keyboard (no control characters), except for parentheses to avoid confusion.

Answer: Define Σ_1 and Σ_2 as in part (a), and let $\Sigma_3 = \Sigma - (\Sigma_1 \cup \Sigma_2)$ be all of the other characters on a computer keyboard except for parentheses. Then a DFA for L_0 is as follows:



6. Define L to be the set of strings that represent numbers in a modified version of Java. The goal in this problem is to define a regular expression and an NFA for L. To precisely define L, let the set of *digits* be $\Sigma_1 = \{\,$`0`, `1`, `2`, ..., `9`$\,\}$, and define the set of *signs* to be $\Sigma_2 = \{\,$`+`, `-`$\,\}$. Then $L = L_1 \cup L_2 \cup L_3$, where

   - L_1 is the set of all strings that are decimal integer numbers. Specifically, L_1 consists of strings that start with an optional sign, followed by one or more digits. Examples of strings in L_1 are "`02`", "`+9`", and "`-241`".
   - L_2 is the set of all strings that are floating-point numbers that are not in exponential notation. Specifically, L_2 consists of strings that start with an optional sign, followed by zero or more digits, followed by a decimal point, and end with zero or more digits, where there must be at least one digit in the string. Examples of strings in L_2 are "`13.231`", "`-28.`" and "`.124`". All strings in L_2 have exactly one decimal point.
   - L_3 is the set of all strings that are floating-point numbers in exponential notation. Specifically, L_3 consists of strings that start with a string from L_1 or L_2, followed by "`E`" or "`e`", and end with a string from L_1. Examples of strings in L_3 are "`-80.1E-083`", "`+8.E5`" and "`1e+31`".

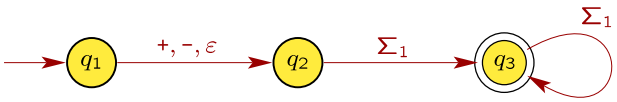
   Assume that there is no limit on the number of digits in a string in L. Also, we do not allow for the suffixes `L`, `l`, `F`, `f`, `D`, `d`, at the end of numbers to denote types (long integers, floats, and doubles). Define Σ as the alphabet of all printable characters on a computer keyboard (no control characters), except for parentheses to avoid confusion.

(a) Give a regular expression for L_1. Also, give an NFA and a DFA for L_1 over the alphabet Σ.

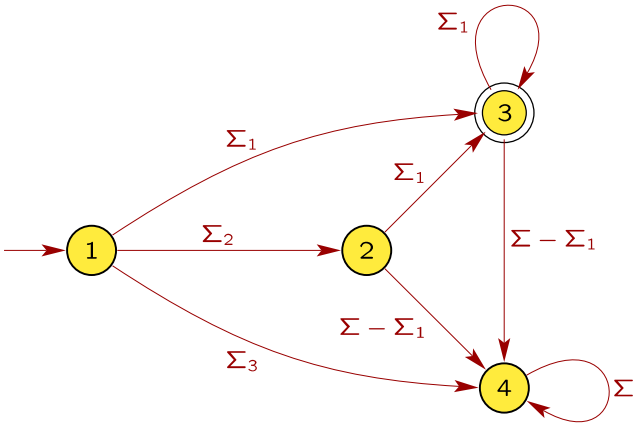
Answer: A regular expression for L_1 is

$$R_1 = (\,+ \cup - \cup \varepsilon\,)\,\Sigma_1\,\Sigma_1^*$$

where $\Sigma_1 = \{\,0,\ 1,\ 2,\ \ldots,\ 9\,\}$ as previously defined. An NFA for L_1 is



Define $\Sigma_3 = \Sigma - (\Sigma_1 \cup \Sigma_2)$ with $\Sigma_2 = \{\,-,\ +\,\}$, as before. Then a DFA for L_1 is



Notice that the DFA is more complicated than the NFA.

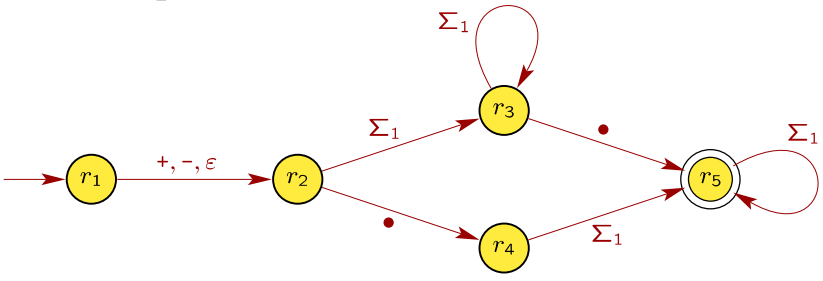
(b) Give a regular expression for L_2. Also, give an NFA for L_2 over the alphabet Σ.

Answer: A regular expression for L_2 is

$$R_2 = (\,+ \cup - \cup \varepsilon\,)(\Sigma_1\,\Sigma_1^*\,.\,\Sigma_1^* \ \cup\ .\,\Sigma_1\,\Sigma_1^*)$$

Note that the regular expression $(\,+ \cup - \cup \varepsilon\,)\,\Sigma_1^*\,.\,\Sigma_1^*$ is not correct since it can generate the strings ".", "+." and "-.", which are not valid floating-point numbers.

An NFA for L_2 is

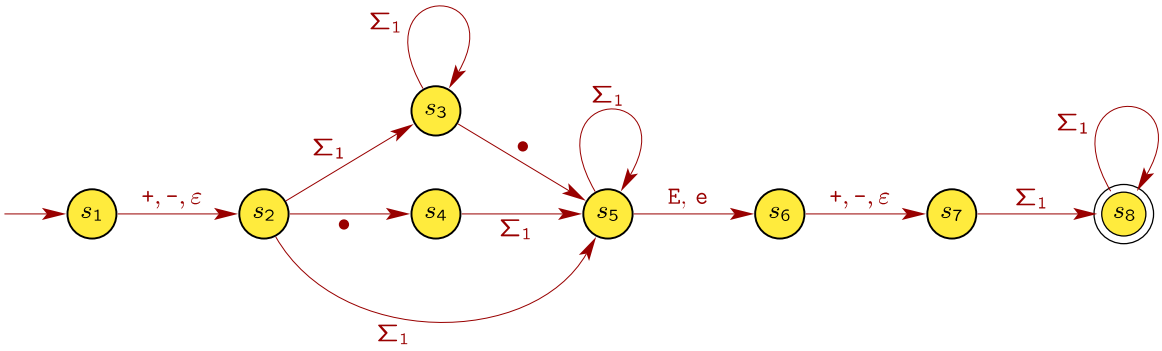


(c) Give a regular expression for L_3. Also, give an NFA for L_3 over the alphabet Σ.

Answer: A regular expression for L_3 is

$$R_3 = (R_1 \cup R_2)\,(\mathtt{E} \cup \mathtt{e})\,R_1$$

where R_1 and R_2 are defined in the previous parts. An NFA for L_3 is

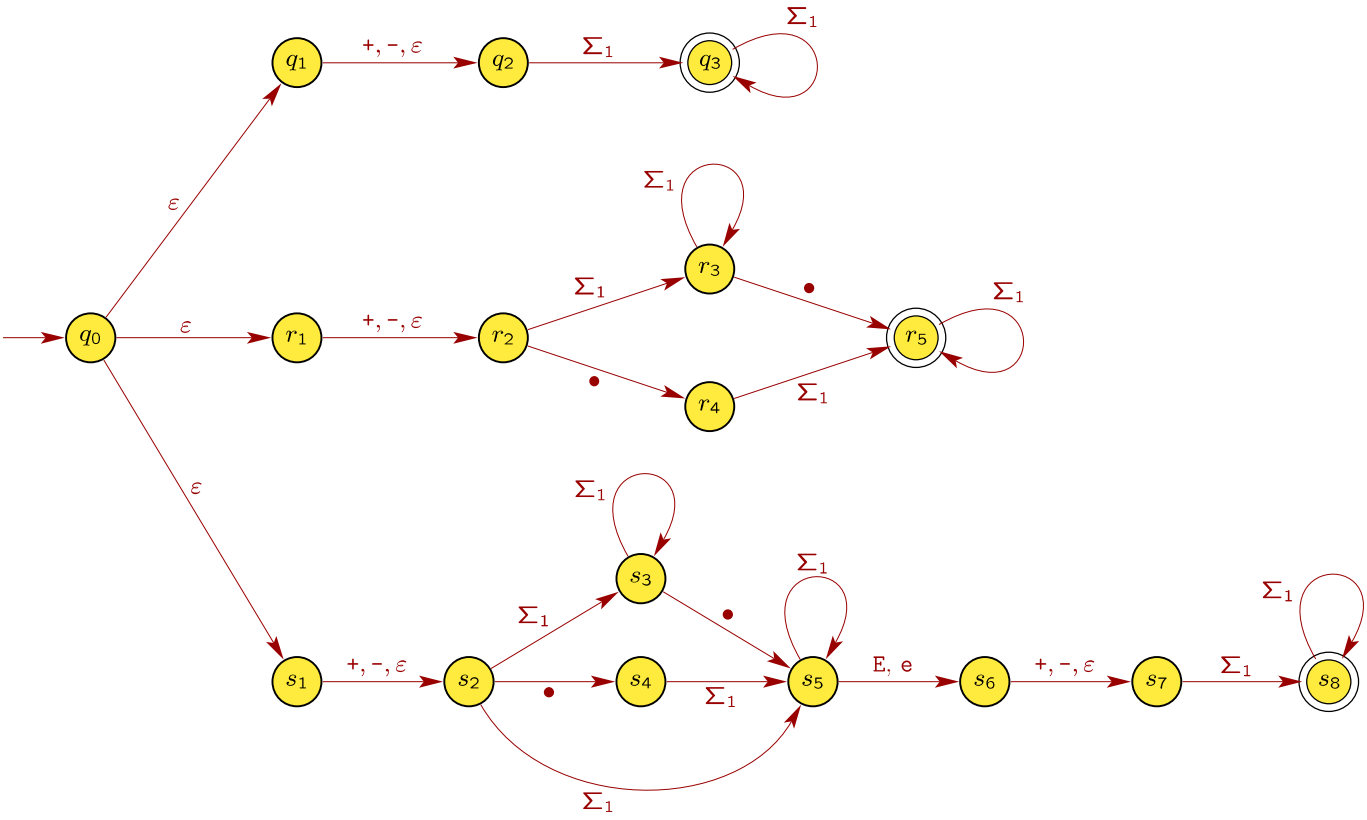


(d) Give a regular expression for the language L. Also, give an NFA for L over the alphabet Σ.

Answer: Note that $L = L_1 \cup L_2 \cup L_3$, so a regular expression for L is

$$R_4 = R_1 \cup R_2 \cup R_3$$

We can construct an NFA for L by taking the union of the NFA's for L_1, L_2 and L_3, as follows:



A simpler NFA for L is to take the NFA for L_3 and make s_3 and s_5 also accepting states in addition to s_8.