# Solutions to Test 1

**Problem 1.** Why do we need to study automata? Provide two main reasons.

**Solution to Problem 1.**

- To help develop a general understanding of which general problems are solvable and which are not.
- To understand how program are compiled.

**Problem 2–4.** Let us consider the automaton that has two states: $s$ (student is struggling) and $c$ (student is doing well); $s$ is the starting state, $c$ is the final state. The only two symbols are $h$ (help provided) and $n$ (no help provided).

- From $s$, $n$ lead back to $s$, and $h$ leads to $c$.
- From $c$, any symbol leads back to $c$.

**Problem 2.** Trace, step-by-step, how this finite automaton will check that the word $hnh$ belongs to this language. Use the tracing to find the parts $x$, $y$, and $z$ of the word $nhn$ corresponding to the Pumping Lemma. Check that the "pumped" word $xyyz$ will also be accepted by this automaton.

**Solution to Problem 2.** Let us first trace how the automaton will accept the word $hnh$:

- we start in the starting state $s$;
- we read the first symbol $h$ and move to $c$;
- we read $n$ and stay in $c$;
- we read $h$ and stay in $c$.

We have read all the letters of the word, we are in the final state, so the word is accepted.

Let us now trace how the automaton will accept the word $nhn$:

$$\begin{array}{ccccccccc} | & & n & | & h & | & n & | \\ \underline{s} & & & \underline{s} & & c & & c \end{array}$$

In this derivation, the first pair of repeating states is the pair of the $s$ states: So:

- $x$ is what is before the first repetition, i.e., $x = \Lambda$;
- $y$ is what is in between the repetitions, i.e., $y = n$; and
- $z$ is what is after the second repetition, i.e., $z = hn$.

By repeating the part between the two repetitions we get the derivation of the word $xyyz = nnhn$:

$$\begin{array}{ccccccccccc} | & & n & | & n & | & h & | & n & | \\ s & & & s & & s & & c & & c \end{array}$$

**Problem 3.** Write down the tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$ corresponding to this automaton:

- $Q$ is the set of all the states,
- $\Sigma$ is the alphabet, i.e., the set of all the symbols that this automaton can encounter;
- $\delta : Q \times \Sigma \to Q$ is the function that describes, for each state $q$ and for each symbol $s$, the state $\delta(q, s)$ to which the automaton that was originally in the state $q$ moves when it sees the symbol $s$ (you do not need to describe all possible transitions this way, just describe two of them);
- $q_0$ is the staring state, and
- $F$ is the set of all final states.

**Solution to Problem 3.** Here, $Q = \{s, c\}$, $\Sigma = \{h, n\}$, $q_0 = s$, $F = \{c\}$, and the function $\delta$ is described by the following table:

| | $s$ | $c$ |
|---|---|---|
| $h$ | $c$ | $c$ |
| $n$ | $s$ | $c$ |

**Problem 4.** Use a general algorithm that we had in class to generate a context-free grammar corresponding to this automaton. Show how this grammar will generate the word $hnh$.

**Solution to Problem 4.** The corresponding grammar has variables $S$ and $C$ corresponding to the states of the automaton. The variable $S$ corresponding to the starting state $s$ is the starting variable. We have the following rules:

$$S \to hC;$$

$$S \to nS;$$

$$C \to hC;$$

$$C \to nC;$$

$$C \to \varepsilon.$$

The corresponding derivation is:

$$\underline{S} \to h\underline{C} \to hn\underline{C} \to hnh\underline{C} \to hnh.$$

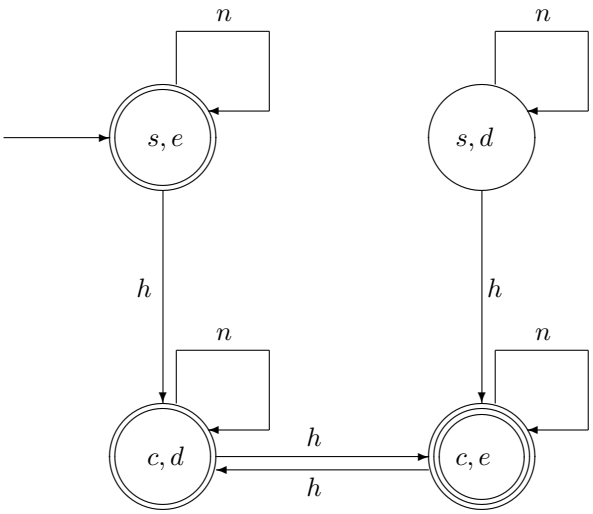
**Problem 5.** Let $A_1$ be the automaton described in Problem 2. Let $A_2$ be an automaton that accepts all the strings that contains even number (0, 2, 4, etc.) of helps $h$s. This automaton has two states: the starting state $e$ which is also final, and the odd state $d$. The transitions are as follows:

- from the start state, $n$ lead back to the start state, while $h$ leads to the odd state $d$;
- from the odd state, $h$ leads to the start state, while $n$ leads back to the odd state.

Use the algorithm that we had in class to describe the following two new automata:

- the automaton that recognizes the union $A_1 \cup A_2$ of the two corresponding languages, and
- the automaton that recognizes the intersection of the languages $A_1$ and $A_2$.

**Solution to Problem 5.**

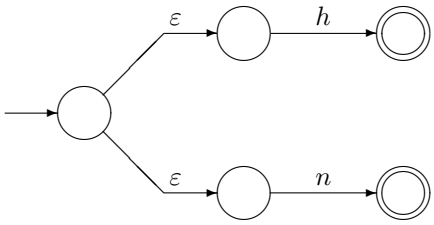


**Problem 6.** Use the general algorithm that we learned in class to design a non-deterministic finite automaton that recognizes the language $(h \cup n)(h \cup n)^*$:

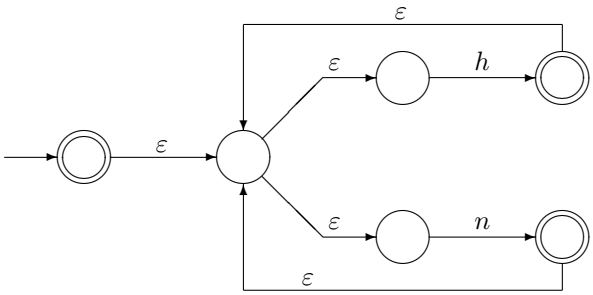
- first, describe the automata for recognizing $h$ and $n$;
- then, combine them into the automata for recognizing the union $h \cup n$, and the Kleene star $(h \cup n)^*$;
- finally, combine the automata for $h \cup n$ and $(h \cup n)^*$ into an automaton for recognizing the desired composition of the two languages.

**Solution to Problem 6.** We start with the standard non-deterministic automata for recognizing the words $h$ and $n$:

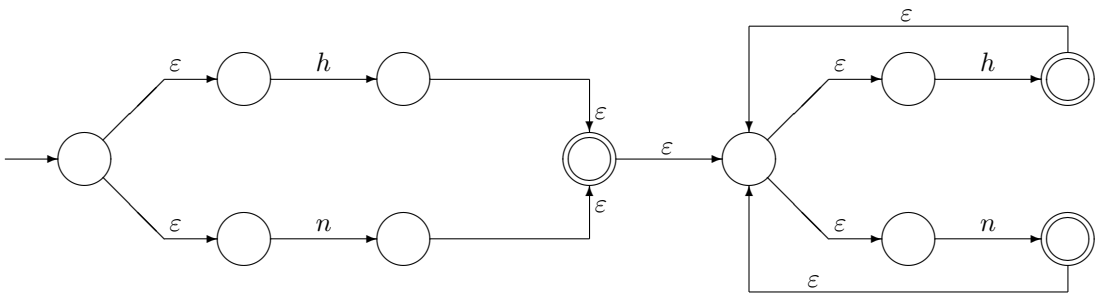

Then, we use the general algorithm for the union to design a non-deterministic automaton for recognizing the language $h \cup n$:




Now, we apply a standard algorithm for the Kleene star, and we get the following non-deterministic automaton for $(h \cup n)^*$:

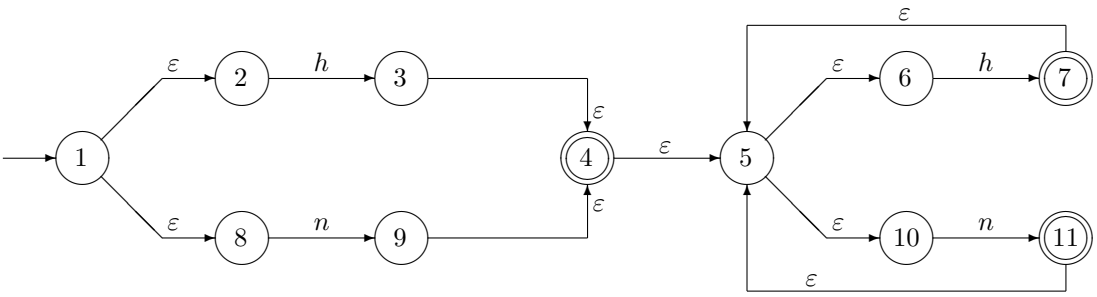



Now, we use the algorithm for concatenation for combine them:

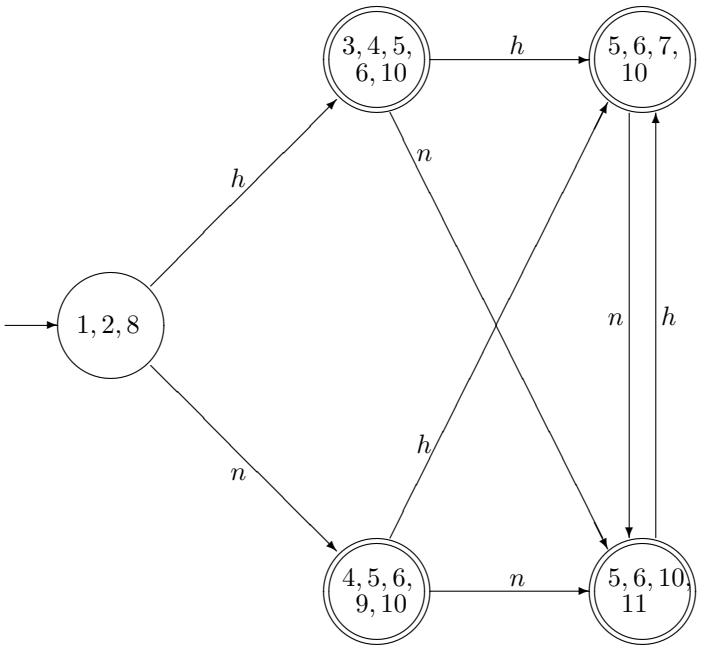

ε
h
ε
ε
n
ε
ε
ε
ε
h
ε
ε
n
ε

**Problem 7.** Use the general algorithm to transform the resulting non-deterministic finite automaton into a deterministic one.

**Solution to Problem 7.** First, we enumerate the states:

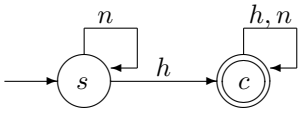



Then, we get the following deterministic automaton:



**Problem 8–9.** Use a general algorithm to transform the finite automaton from Problem 2 into the corresponding regular expression.

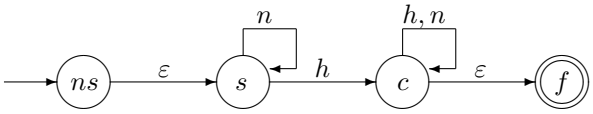
**Solution to Problem 8–9.** We start with the described automaton:




According to the general algorithm, first we add a new start state $ns$ and a new final state $f$, and we add jumps:

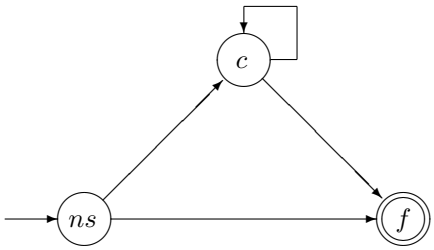
- from the new start state $ns$ to the old start state, and
- from each old final state to the new final state $f$.

As a result, we get the following automaton.




Then, we need to eliminate the two intermediate states $s$ and $c$ one by one. We can start with eliminating $s$ or with eliminating $c$. Let us show what happens in both cases.

**First version, when we first eliminate the state $s$.** First, we draw all possible arrows:



Now, to find expressions to place at all these arrows, we will use the general formula

$$R'_{i,j} = R_{i,j} \cup (R_{i,k} R^*_{k,k} R_{k,j}),$$

where $k$ is the state that we are eliminating, i.e., in this case, the state $k = s$.

By applying this formula, and by using simplification formulas described in the lecture, we get the following results:

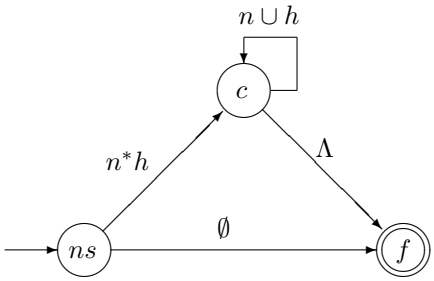
$$R'_{ns,c} = R_{ns,c} \cup (R_{ns,s} R^*_{s,s} R_{s,c}) = \emptyset \cup (\Lambda n^* h) = \emptyset \cup n^* h = n^* h;$$

$$R'_{ns,f} = R_{ns,f} \cup (R_{ns,s} R^*_{s,s} R_{s,f}) = \emptyset \cup (\Lambda n^* \emptyset) = \emptyset \cup \emptyset = \emptyset;$$

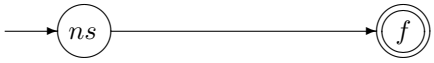
$$R'_{c,c} = R_{c,c} \cup (R_{c,s} R^*_{s,s} R_{s,c}) = h \cup n \cup (\emptyset \ldots) = h \cup n \cup \emptyset = h \cup n;$$

$$R'_{c,f} = R_{c,f} \cup (R_{c,s} R^*_{s,s} R_{s,f}) = \Lambda \cup (\emptyset \ldots) = \Lambda \cup \emptyset = \Lambda.$$

Thus, the 3-state a-automaton takes the following form:




Now, all that remains to do is to go from here to the 2-state a-automaton by eliminating the remaining state $c$:




The final expression is the corresponding expression for $R'_{ns,f}$:

$$R'_{ns,f} = R_{ns,f} \cup (R_{ns,c} R^*_{c,c} R_{c,f}) =$$

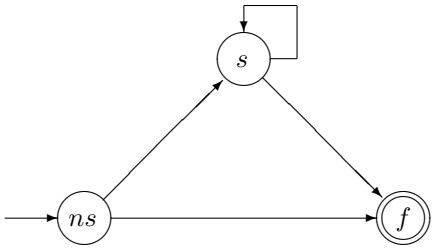
$$\emptyset \cup (n^* h (n \cup h)^* \Lambda) = \emptyset \cup (n^* h (h \cup n)^*) =$$

$$n^* h (h \cup n)^*.$$

The formula on the previous line is a regular expression corresponding to the original automaton.

**First version – answer:** $n^* h (h \cup n)^*$.

**Second version, when we first eliminate the state $c$.** First, we draw all possible arrows:




Now, to find expressions to place at all these arrows, we will use the general formula

$$R'_{i,j} = R_{i,j} \cup (R_{i,k} R^*_{k,k} R_{k,j}),$$

where $k$ is the state that we are eliminating, i.e., in this case, the state $k = c$.

By applying this formula, and by using simplification formulas described in the lecture, we get the following results:

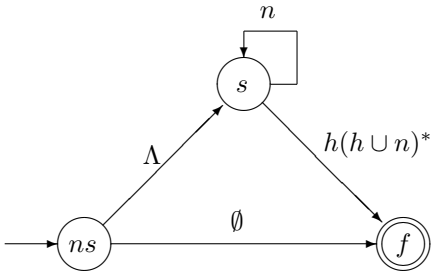
$$R'_{ns,s} = R_{ns,s} \cup (R_{ns,c} R^*_{c,c} R_{c,s}) = \Lambda \cup (\emptyset \ldots) = \Lambda \cup \emptyset = \Lambda;$$

$$R'_{ns,f} = R_{ns,f} \cup (R_{ns,c} R^*_{c,c} R_{c,f}) = \emptyset \cup (\emptyset \ldots) = \emptyset \cup \emptyset = \emptyset;$$

$$R'_{s,s} = R_{s,s} \cup (R_{s,c} R^*_{c,c} R_{c,s}) = n \cup (\ldots \emptyset) = n \cup \emptyset = n;$$

$$R'_{s,f} = R_{s,f} \cup (R_{s,c} R^*_{c,c} R_{c,f}) = \emptyset \cup (h(h \cup n)^* \Lambda) = \emptyset \cup (h(h \cup n)^*) = h(h \cup n)^*.$$

Thus, the 3-state a-automaton takes the following form:




Now, all that remains to do is to go from here to the 2-state a-automaton by eliminating the remaining state $s$:

The final expression is the corresponding expression for $R'_{ns,f}$:

$$R'_{ns,f} = R_{ns,f} \cup (R_{ns,s} R^*_{s,s} R_{s,f}) =$$

$$\emptyset \cup (\Lambda n^* h (h \cup n)^*) =$$

$$\emptyset \cup (n^* h (h \cup n)^*) =$$

$$n^* h (h \cup n)^*.$$

The formula in the previous line is also a regular expression corresponding to the original automaton.

**Problem 10.** Prove that the language $L$ of all the words that have at least twice more $h$'s than $n$'s is not regular.

**Solution to Problem 10.** We will prove it by contradiction. Let us assume that the language $L$ is regular, and let us show that this assumption leads to a contradiction.

Since this language is regular, according to the Pumping Lemma, there exists an integer $p$ such that every word from $L$ whose length $\text{len}(w)$ is at least $p$ can be represented as a concatenation $w = xyz$, where:

- $y$ is non-empty;
- the length $\text{len}(xy)$ does not exceed $p$, and
- for every natural number $i$, the word $xy^iz \stackrel{\text{def}}{=} xy\ldots yz$, in which $y$ is repeated $i$ times, also belongs to the language $L$.

Let us take the word

$$w = n^p h^{2p} = n\ldots nh\ldots h,$$

in which first $n$ is repeated $p$ times, then $h$ is repeated $2p$ times. The length of this word is $p + 2p = 3p > p$. So, by pumping lemma, this word can be represented as $w = xyz$ with $\text{len}(xy) \le p$. This word starts with $xy$, and the length of $xy$ is smaller than or equal to $p$. Thus, $xy$ is among the first $p$ symbols of the word $w$ – and these symbols are all $n$'s. So, the word $y$ only has $n$'s.

Thus, when we go from the word $w = xyz$ to the word $xyyz$, we add at least one $n$, and we do not add any $h$'s. So, in the word $xyyz$, there are now at least $p + 1$ letters $n$. Since there are at least $p + 1$ letters $n$, and only $2p$ $h$'s, this means that the number of $h$'s is no longer at least twice larger than the number of $n$'s. Thus, the word $xyyz$ cannot be in the language $L$, since by definition $L$ only contains words which have at least twice more $h$'s than $n$'s.

On the other hand, by Pumping Lemma, the word $xyyz$ must be in the language $L$. So, we proved two opposite statements:

- that this word *is not* in $L$ and
- that this word *is* in $L$.

This is a contradiction.

The only assumption that led to this contradiction is that $L$ is a regular language. Thus, this assumption is false, so $L$ is not regular.