Solutions to Test 1, Fall 2024

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 oversimplified version of a computer that has only two states: the starting state f (off) and the final state n (on), and two possible actions: p (pressing the on-off key) and k (pressing any other key). The symbol k brings us back to the same state, while the symbol p brings us from on to off and from off to on.

Problem 2. Trace, step-by-step, how this finite automaton will check that the word $pkpkp$ belongs to this language. Use the tracing to find the parts x, y, and z of the word $pkpkp$ 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 sws:

- we start in the starting state f;
- we read the first symbol p and switch to n;
- we read k and stay in the state n;
- we read p and go back to the off stats f;
- we read k and stay in the off state f;
- we read p and go to the state n.

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 $pkpkp$:

	p		k		p		k		p	
f		$\underline{n}$		$\underline{n}$		f		f		n

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

- x is what is before the first repetition, i.e., $x = p$;
- y is what is in between the repetitions, i.e., $y = k$; and
- z is what is after the second repetition, i.e., $z = pkp$.

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

	p		k		k		p		k		p	
f		$\underline{n}$		$\underline{n}$		$\underline{n}$		f		f		n

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,
- Σ 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, $Q = \{f, n\}$, $\Sigma = \{p, k\}$, $q_0 = f$, $F = \{n\}$, and the function δ is described by the following table:

	f	n
p	n	f
k	f	n

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 *pkpkp*.

Solution to Problem 4. The corresponding grammar has variables F and N corresponding to the states of the automaton. The variable H corresponding to the starting state h is the starting variable. We have the following rules:

$$F \to pN;$$

$$F \to kF;$$

$$N \to pF;$$

$$N \to kN;$$

$$N \to \varepsilon.$$

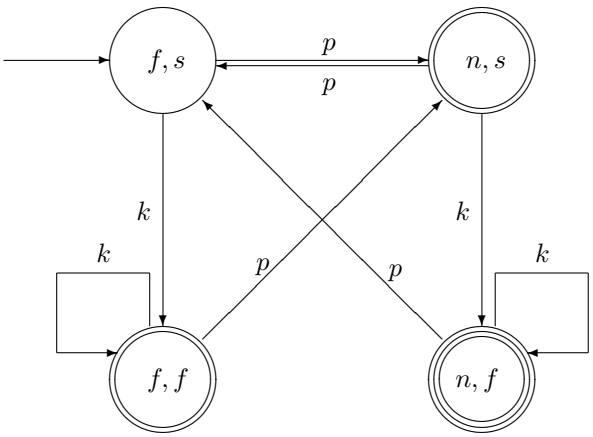
The corresponding derivation is:

$$\underline{F} \to p\underline{N} \to pk\underline{N} \to pkp\underline{F} \to pkpk\underline{F} \to pkpkp\underline{N} \to pkpkp.$$

Problem 5. Let A_1 be the automaton described in Problem 2. Let A_2 be an automaton that accepts only sequences that end with k. This automaton has two states: the starting state s and the final state f. The transitions are as follows: from any state, p leads to s and k leads to f. 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 $(pk^*)^*p$ (that corresponds to all the sequences that start and end with p):

- first, describe automata for recognizing p and k;
- then, use the automaton for k to design an automaton for recognizing k^*;
- then, combine the automata for p and k^* into an automaton for recognizing concatenation pk^*;
- then, use the automaton for pk^* to design an automaton for recognizing the Kleene star $(pk^*)^*$;
- finally, combine the automata for $(pk^*)^*$ and p into an automaton for recognizing concatenation $(pk^*)^*p$.

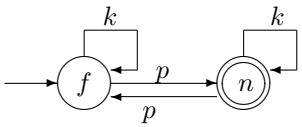
Solution to Problem 6. see an additional file.

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

Solution to Problem 7. see an additional file.

Problem 8–9. Use a general algorithm to transform the finite automaton from Problem 2 into the corresponding regular expression. Start with eliminating the state n.

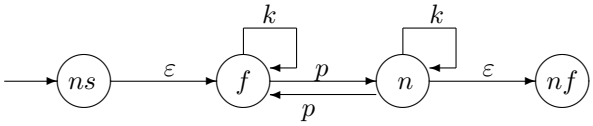
Solution to Problem 8–9. We start with the described automaton:



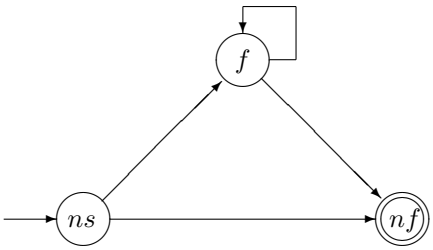
According to the general algorithm, first we add a new start state ns and a new final state nf, and we add jumps:

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

As a result, we get the following automaton.



Then, we need to eliminate the two intermediate states f and n one by one. We first eliminate the state n. 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 $k = n$.

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

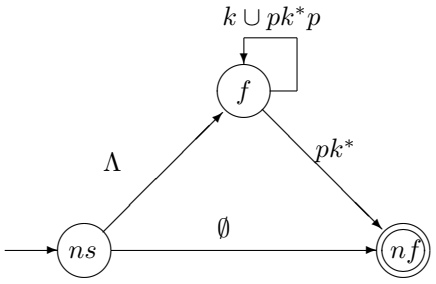
$$R'_{ns,nf} = R_{ns,nf} \cup (R_{ns,n}R^*_{n,n}R_{n,nf}) = \emptyset \cup (\emptyset \ldots) = \emptyset \cup \emptyset = \emptyset;$$

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

$$R'_{f,f} = R_{f,f} \cup (R_{f,n}R^*_{n,n}R_{n,f}) = k \cup (pk^*p);$$

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

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 f:

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

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

$$\emptyset \cup (\Lambda(k \cup pk^*p)^*pk^*) =$$

$$(k \cup pk^*p)^*pk^*.$$

The formula on the previous line is 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 k's than p'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

$$w = p^P k^{2P} = p\ldots pk\ldots k,$$

in which first p is repeated P times, then k 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 p's. So, the word y only has p's.

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