

Automata, Fall 2022, Solutions to Final Exam

Problem 1. *Finite automata and regular languages.*

Problem 1a. Design a finite automaton for recognizing words that do not contain the letter q . Assume that the input strings contain only symbols p , q , and r . The easiest is to have two states:

- the starting state s indicating that we have not yet read any q 's; and
- the state f indicating that we have already read a symbol q .

You just need to describe transitions between these states, and which states are final. Show, step-by-step, how your automaton will accept the word prp .

Solution.

- from s , q leads to f , and p and r lead back to s ;
- from f , all three symbols lead back to f .

The final state is s . Derivation of prp is as follows:

$$\begin{array}{cccc} | & p & | & r & | & p & | \\ s & & s & & s & & s \end{array}$$

Problem 1b. Explain why in most computers binary numbers are represented starting with the lowest possible digit.

Solution. In most actual computers, the representation of a number starts with the least significant digit, since all arithmetic operations like addition, subtraction, or multiplication start with the least significant digit. So, if we store the number the way we write numbers, most significant digits first, computers will have to waste time going through all the digits until they come up with the least significant digit and start the actual computations. To speed up computations, representations therefore start with the least significant digits.

Problem 1c. On the example of the above automaton, show how the word prp can be represented as xyz in accordance with the pumping lemma.

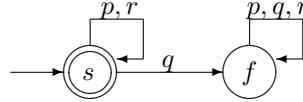
Solution. Here, the first repeating state is s :

$$\begin{array}{ccccccc} & | & p & | & r & | & p & | \\ \underline{s} & & & \underline{s} & & s & & s \end{array}$$

So, $x = \Lambda$, $y = p$, and $z = rp$.

Problem 1d. Use a general algorithm to describe a regular expression corresponding to the finite automaton from the Problem 1a. (If you are running out of time, it is Ok not to finish, just eliminate the first state.)

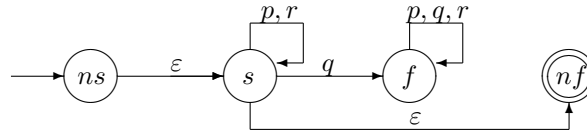
Solution. We start with the described automaton:



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 s and f one by one. The state f is a sink, so it is the easiest to eliminate. In general, we have a 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. In this case, we have $k = f$, so

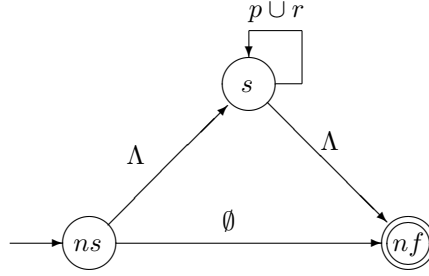
$$R'_{i,j} = R_{i,j} \cup (R_{i,f} R_{f,f}^* R_{f,j}).$$

Here, for all remaining states j , there is no arrow from f to j , so $R_{f,j} = \emptyset$. Concatenation with the empty set is always the empty set, and union with the

empty set does not change anything, so we get $R'_{i,j} = R_{i,j}$. So, we have the following:

$$\begin{aligned} R'_{ns,s} &= R_{ns,s} = \Lambda; \\ R'_{ns,nf} &= R_{ns,nf} = \emptyset; \\ R'_{s,s} &= R_{s,s} = p \cup r; \\ R'_{s,nf} &= R_{s,nf} = \Lambda. \end{aligned}$$

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



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

$$\begin{aligned} R'_{ns,nf} &= R_{ns,nf} \cup (R_{ns,s} R_{s,s}^* R_{s,nf}) = \\ &= \emptyset \cup (\Lambda (p \cup r)^* \Lambda) = \\ &= (p \cup r)^*. \end{aligned}$$

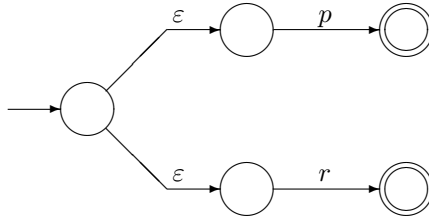
The last expression is a regular expression corresponding to the original automaton.

Problem 1e-f. The resulting language can be described by a regular expression $(p \cup r)^*$. Use a general algorithm to transform this regular expression into a finite automaton: first a non-deterministic one, then a deterministic one.

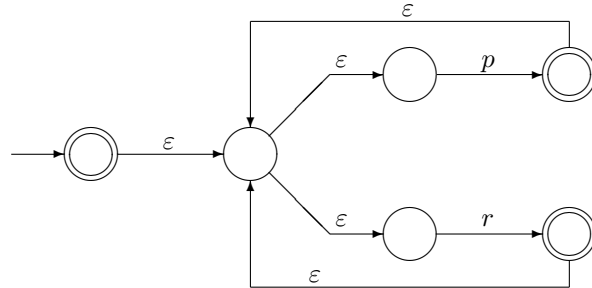
Solution. We start with the standard non-deterministic automata for recognizing the words p and r :



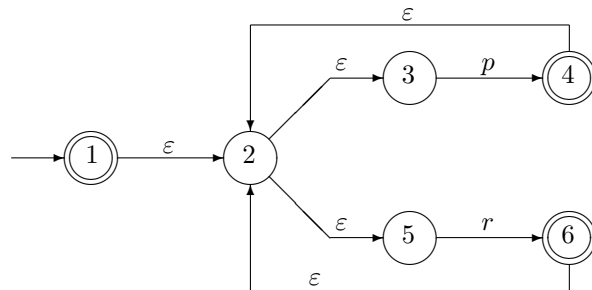
Then, we use the general algorithm for the union to design a non-deterministic automaton for recognizing the language $p \cup r$:



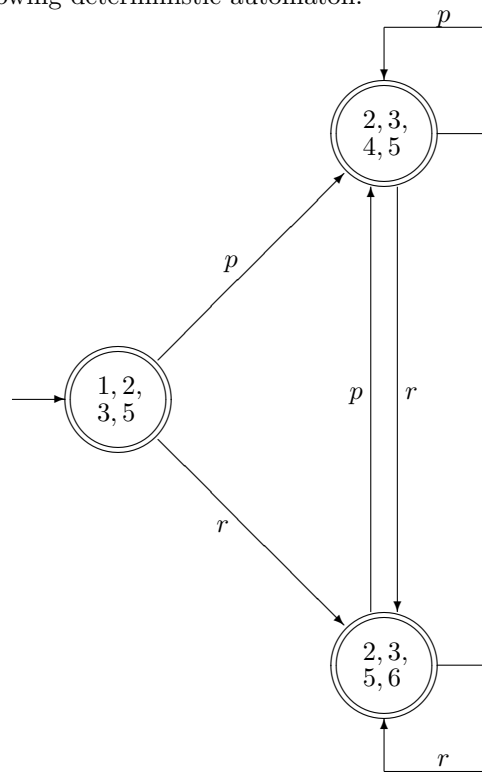
Now, we apply a standard algorithm for the Kleene star, and we get the following non-deterministic automaton for $(p \cup r)^*$:



To get a deterministic finite automation, first, we enumerate the states:



Then, we get the following deterministic automaton:



Problem 2. *Beyond finite automata: pushdown automata and context-free grammars*

Problem 2a. Suppose that a person goes to paradise if he did more good deeds (g) than bad ones (b). Prove that it is not possible to have a finite automaton that, given a list of deeds – like $ggbg$ – would decide whether a person goes to paradise or not.

Solution We will prove it by contradiction. Let us assume that the language L of all sequences leading to paradise 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 $xy^iz \stackrel{\text{def}}{=} xy \dots yz$, in which y is repeated i times, also belongs to the language L .

Let us take the word $w = b^p g^{p+1}$ in which first b is repeated p times, then g is repeated $p + 1$ times. The length of this word is $p + (p + 1) = 2p + 1 > p$. So, by pumping lemma, this word can be represented as $w = xyz$ with $\text{len}(xy) \leq p$. 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 bs . So, the word y only has bs .

Thus, when we go from the word $w = xyz$ to the word $xyyz$, we add bs , and we do not add any gs . In the word w , we have exactly one more good deed than bad deeds. So, in the word $xyyz$, we added at least one bad deed, so the number of good deeds is no longer larger than the number of bad deeds. Thus, the word $xyyz$ cannot be in the language L , since by definition L only contains words that contain more good deeds than bad deeds.

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.

Problem 2b. Use a general algorithm to transform the finite automaton from the Problem 1a into a context-free grammar (CFG). Show, step-by-step, how this CFG will generate the word prp .

Solution. According to the general algorithm, the corresponding grammar should have two variables S and F , and the following rules:

$$S \rightarrow pS, \quad S \rightarrow qF, \quad S \rightarrow rS, \quad F \rightarrow pF, \quad F \rightarrow qF, \quad F \rightarrow rF, \quad S \rightarrow \varepsilon.$$

The word prp is accepted by the finite automaton as follows:

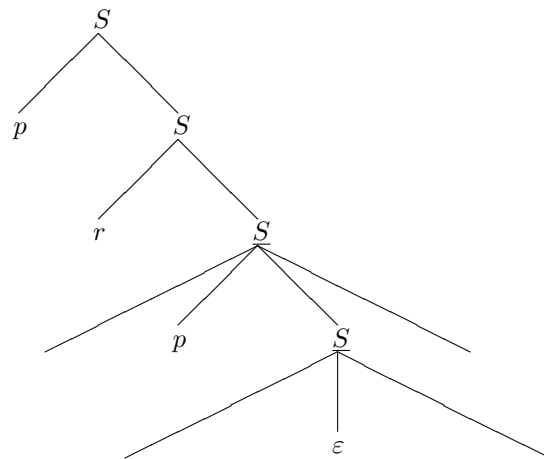
$$\begin{array}{cccc} | & p & | & r & | & p & | \\ s & & s & & s & & s \end{array}$$

Thus, its derivation takes the following form:

$$\underline{S} \rightarrow p\underline{S} \rightarrow pr\underline{S} \rightarrow prp\underline{S} \rightarrow prp.$$

Problem 2c. For the context-free grammar from the Problem 2b, show how the word prp can be represented as $uvxyz$ in accordance with the pumping lemma.

Solution. In terms of a tree, the derivation of the word prp can be represented as follows:



Here, $u = pr$, $v = p$, and $x = y = z = \varepsilon$.

Problem 2d. Use a general algorithm to translate the CFG from 2b into Chomsky normal form.

Solution.

Preliminary step. We add a new starting variable S_0 and a rule $S_0 \rightarrow S$:

$$S \rightarrow pS, \quad S \rightarrow qF, \quad S \rightarrow rS, \quad F \rightarrow pF, \quad F \rightarrow qF, \quad F \rightarrow rF, \quad S \rightarrow \varepsilon, \\ S_0 \rightarrow S.$$

Step 0. We need to eliminate the rule $S \rightarrow \varepsilon$. This means adding the rules $S \rightarrow p$, $S \rightarrow r$, and $S_0 \rightarrow \varepsilon$:

$$S \rightarrow pS, \quad S \rightarrow qF, \quad S \rightarrow rS, \quad F \rightarrow pF, \quad F \rightarrow qF, \quad F \rightarrow rF, \quad S_0 \rightarrow S, \\ S \rightarrow p, \quad S \rightarrow r, \quad S_0 \rightarrow \varepsilon.$$

Step 1. We eliminate the rule $S_0 \rightarrow S$ by adding the rules $S_0 \rightarrow pS$, $S_0 \rightarrow qF$, $S_0 \rightarrow rS$, $S_0 \rightarrow p$, and $S_0 \rightarrow r$:

$$S \rightarrow pS, \quad S \rightarrow qF, \quad S \rightarrow rS, \quad F \rightarrow pF, \quad F \rightarrow qF, \quad F \rightarrow rF, \\ S \rightarrow p, \quad S \rightarrow r, \quad S_0 \rightarrow \varepsilon, \\ S_0 \rightarrow pS, \quad S_0 \rightarrow qF, \quad S_0 \rightarrow rS, \quad S_0 \rightarrow p, \quad S_0 \rightarrow r.$$

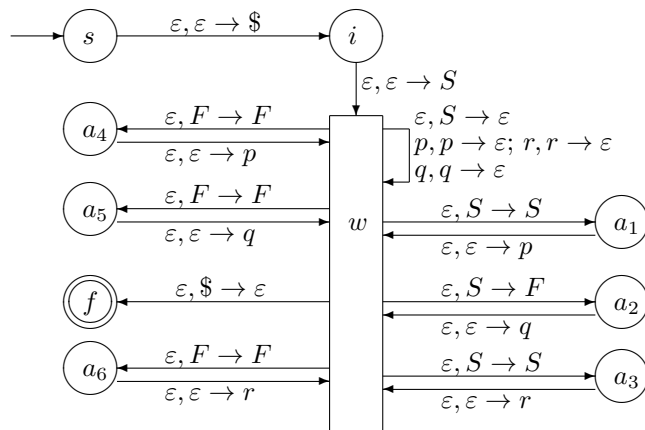
Step 2. We introduce three new variables V_p , V_q , and V_r , replace p , q , and r in length-2 right-hand sides with V_p , V_q , and V_r , and add rules $V_p \rightarrow p$, $V_q \rightarrow q$, and $V_r \rightarrow r$:

$$S \rightarrow V_pS, \quad S \rightarrow V_qF, \quad S \rightarrow V_rS, \quad F \rightarrow V_pF, \quad F \rightarrow V_qF, \quad F \rightarrow V_rF, \\ S \rightarrow p, \quad S \rightarrow r, \quad S_0 \rightarrow \varepsilon, \\ S_0 \rightarrow V_pS, \quad S_0 \rightarrow V_qF, \quad S_0 \rightarrow V_rS, \quad S_0 \rightarrow p, \quad S_0 \rightarrow r, \\ V_p \rightarrow p, \quad V_q \rightarrow q, \quad V_r \rightarrow r.$$

This is already Chomsky normal form.

Problem 2e. Use a general algorithm to translate the CFG from 2b into an appropriate push-down automaton. Explain, step-by-step, how this automaton will accept the word prp .

Solution.



The word prp is derived as follows:

$$\underline{S} \rightarrow p\underline{S} \rightarrow pr\underline{S} \rightarrow prp\underline{S} \rightarrow prp.$$

So, we have the following acceptance by the pushdown automaton:

| | | | | | | | | | | | | | |
|-----|-----|-----|-------|-----|-----|-------|-----|-----|-------|-----|-----|-----|-----|
| | | | | | p | | | r | | | p | | |
| s | i | w | a_1 | w | w | a_3 | w | w | a_1 | w | w | w | f |
| | \$ | S | S | p | S | S | r | S | S | p | S | \$ | |
| | | \$ | \$ | S | \$ | \$ | S | \$ | \$ | S | \$ | | |

Problem 2f. Use the general stack-based algorithms to show:

- how the compiler will transform a Java expression $12/8/22$, with integers 12, 8, and 22, into inverse Polish (postfix) notation, and
- how it will compute the value of this expression.

Solution. Let us show, step by step, how the above expression is transformed into the postfix form:

$$\begin{array}{ccccccc} 12 & / & 8 & / & 22 & & \\ \hline 12 & & 8 & / & 22 & / & \\ \hline & / & / & / & / & & \end{array}$$

Let us now show how this expression will be computed:

$$\begin{array}{ccccccc} 12 & 8 & / & 22 & / & & \\ \hline 12 & 8 & 1 & 22 & 0 & & \\ & 12 & & 1 & & & \end{array}$$

Problem 3. *Beyond pushdown automata: Turing machines*

Problem 3a. To be fair, the college decided to allocate all new faculty positions between department proportionally to number of students, so that Computer Science (c) and Industrial engineering (i) get equal number of positions, and mechanical engineering (m) gets twice fewer. For example, the distribution $cicmi$ is fair, as well as $cmmiicci$. Prove that the language of fair sequences is not context-free and therefore, cannot be recognized by a pushdown automaton.

Solution: Proof by contradiction. Let us assume that this language is context-free. Then, by the pumping lemma for context-free grammars, there exists an integer p such that every word w from this language whose length is at least p can be represented as $w = uvxyz$, where $\text{len}(vy) > 0$, $\text{len}(vxy) \leq p$, and for every i , we have $uv^i xy^i z \in L$.

Let us take the word $w = c^{2p} i^{2p} m^p \in L$. The length of this word is $5p > p$, so this word can be represented as $w = uvxyz$.

Where is vxy ? Since the length of this part does not exceed p , this word cannot contain all three letters – otherwise, it will have to contain all the i symbols – there are $2p$ of these symbols – and also at least one c and at least one m , so vxy will have a length at least $2p + 2$, but its length is $\leq p$. So, we have the following possible cases:

- vxy is in c 's;
- vxy is between c 's and i 's;
- vxy is in i 's;
- vxy is between i 's and m 's; or
- vxy is in m 's.

In the first case, v and y contain only c 's. So, when we go from $uvxyz$ to $uvvxyyz$, we add c 's, but we do not add i 's; thus, the desired balance between numbers of c 's and i 's is disrupted, and so $uvvxyyz \notin L$ – while by pumping lemma, we should have $uvvxyyz \in L$. Thus, this case is impossible.

In the second case, v and y contain only c 's and i 's. So, when we go from $uvxyz$ to $uvvxyyz$, we add c 's and i 's, but we do not add any m 's; thus, the desired balance between numbers of c 's, i 's, and m 's is disrupted, and so $uvvxyyz \notin L$ – while by pumping lemma, we should have $uvvxyyz \in L$. Thus, this case is impossible.

Similarly, we can prove that the other cases are also not possible. So, none of the cases is possible, which means that our assumption that the language L is context-free is wrong.

Problem 3b-c. Use a general algorithm to design a Turing machine that accepts exactly all sequences accepted by a finite automaton from Problem 1a. Show, step-by-step, how this Turing machine will accept the word prp . Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

Solution: This Turing machine will have the following rules:

- start, $- \rightarrow s, R$
- $s, p \rightarrow s, R$
- $s, q \rightarrow f, R$
- $s, r \rightarrow s, R$
- $f, p \rightarrow f, R$
- $f, q \rightarrow f, R$
- $f, r \rightarrow f, R$
- $s, - \rightarrow \text{accept}$
- $f, - \rightarrow \text{reject}$

Tracing:

Moment 1:

| | | | | | | |
|----------|-----|-----|-----|-----|---------|-------|
| \sqcap | p | r | p | $-$ | \dots | start |
|----------|-----|-----|-----|-----|---------|-------|

Here, the left stack is empty, the right stack has the following form:

| |
|-----|
| $-$ |
| p |
| r |
| p |

Moment 2:

| | | | | | | |
|-----|-----------------|-----|-----|-----|---------|-----|
| $-$ | \underline{p} | r | p | $-$ | \dots | s |
|-----|-----------------|-----|-----|-----|---------|-----|

Here, the stacks have the following form:

| | |
|----------|-----|
| \sqcap | p |
| | r |
| | p |

Moment 3:

| | | | | | | |
|---|-----|-----------------|-----|---|-----|-----|
| - | p | \underline{r} | p | - | ... | s |
|---|-----|-----------------|-----|---|-----|-----|

Here, the stacks have the following form:

| | |
|-----|-----|
| p | r |
| - | p |

Moment 4:

| | | | | | | |
|---|-----|-----|-----------------|---|-----|-----|
| - | p | r | \underline{p} | - | ... | s |
|---|-----|-----|-----------------|---|-----|-----|

Here, the stacks have the following form:

| | |
|-----|-----|
| r | |
| p | p |
| - | |

Moment 5:

| | | | | | | |
|---|-----|-----|-----|---|-----|-----|
| - | p | r | p | = | ... | s |
|---|-----|-----|-----|---|-----|-----|

Here, the stacks have the following form:

| | |
|-----|---|
| p | |
| r | - |
| p | |
| - | |

Moment 6:

| | | | | | | |
|---|-----|-----|-----|---|-----|--------|
| - | p | r | p | = | ... | accept |
|---|-----|-----|-----|---|-----|--------|

The machine halts, the stack remain the same.

Problem 3d-e. Design Turing machines for computing $a - 2$ in unary and in binary codes. Trace both Turing machines for $a = 5$.

Solution: unary case. The rules are:

start, $- \rightarrow R$, moving
moving, $1 \rightarrow R$
moving, $- \rightarrow L$, erasing1
erasing1, $1 \rightarrow -, L$, erasing2
erasing2, $1 \rightarrow -, L$, back
back, $1 \rightarrow L$
back, $- \rightarrow \text{halt}$

Tracing:

| | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|-----|----------|
| <u>-</u> | 1 | 1 | 1 | 1 | 1 | - | ... | start |
| - | <u>1</u> | 1 | 1 | 1 | 1 | - | ... | moving |
| - | 1 | <u>1</u> | 1 | 1 | 1 | - | ... | moving |
| - | 1 | 1 | <u>1</u> | 1 | 1 | - | ... | moving |
| - | 1 | 1 | 1 | <u>1</u> | 1 | - | ... | moving |
| - | 1 | 1 | 1 | 1 | <u>1</u> | - | ... | moving |
| - | 1 | 1 | 1 | 1 | 1 | <u>-</u> | ... | moving |
| - | 1 | 1 | 1 | 1 | 1 | <u>1</u> | - | erasing1 |
| - | 1 | 1 | 1 | 1 | <u>1</u> | - | - | erasing2 |
| - | 1 | 1 | <u>1</u> | - | - | - | ... | back |
| - | 1 | <u>1</u> | 1 | - | - | - | ... | back |
| - | <u>1</u> | 1 | 1 | - | - | - | ... | back |
| <u>-</u> | 1 | 1 | 1 | - | - | - | ... | back |
| <u>-</u> | 1 | 1 | 1 | - | - | - | ... | halt |

Solution: binary case. The rules are:

start, $- \rightarrow R$, skip
skip, $0 \rightarrow R$, moving
skip, $1 \rightarrow R$, moving
moving, $0 \rightarrow 1, R$
moving, $1 \rightarrow 0, L$, back
back, $0 \rightarrow L$
back, $1 \rightarrow L$
back, $- \rightarrow \text{halt}$

Here is tracing:

| | | | | | | | | |
|----------|----------|----------|----------|---|---|---|-----|--------|
| <u>—</u> | 1 | 0 | 1 | — | — | — | ... | start |
| — | <u>1</u> | 0 | 1 | — | — | — | ... | skip |
| — | 1 | <u>0</u> | 1 | — | — | — | ... | moving |
| — | 1 | 1 | <u>1</u> | — | — | — | ... | moving |
| — | 1 | <u>1</u> | 0 | — | — | — | ... | back |
| — | <u>1</u> | 1 | 0 | — | — | — | ... | back |
| <u>—</u> | 1 | 1 | 0 | — | — | — | ... | back |
| <u>—</u> | 1 | 1 | 0 | — | — | — | ... | halt |

Problem 4. *Beyond Turing machines: computability*

Problem 4a. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world?

Solution: Church-Turing thesis states that any function that can be computed on any physical device can also be computed by a Turing machine (or, equivalently, by a Java program).

Whether this statement is true or not depends on the properties of the physical world. Thus, this statement is not a mathematical theorem, it is a statement about the physical world.

Problem 4b. Prove that the halting problem is not algorithmically solvable.

Solution: The halting problem is the problem of checking whether a given program p halts on given data d . We can prove that it is not possible to have an algorithm $\text{haltChecker}(p,d)$ that always solves this problem by contradiction. Indeed, suppose that such an algorithm – i.e., such a Java program – exists. Then, we can build the following auxiliary Java program:

```
public static int aux(String x)
{if(haltChecker(x,x))
    (while(true) x= x);}
else{return 0;}}
```

If aux halts on aux , then $\text{haltChecker}(\text{aux},\text{aux})$ is true, so the program aux goes into an infinite loop – and never halts. On the other hand, if aux does not halt on aux , then $\text{haltChecker}(\text{aux},\text{aux})$ is false, so the program aux returns 0 – and thus, halts. In both cases, we get a contradiction, which proves that haltChecker is not possible.

Problem 4c. Not all algorithms are feasible, but, unfortunately, we do not have a perfect definition of feasibility. Give a current formal definition of feasibility, explain what it means to be practically feasible, and give two examples:

- an example of an algorithm's running time which is feasible according to the current definition but not practically feasible, and
- an example of an algorithm's running time which is practically feasible but not feasible according to the current definition.

These examples must be different from what we studied in class.

Solution: An algorithm A is called feasible if its running time $t_A(x)$ on each input x is bounded by some polynomial $P(\text{len}(x))$ of the length $\text{len}(x)$ of the input: $t_A(x) \leq P(\text{len}(x))$. In other words, the algorithm is feasible if for each length n , the worst-case complexity $t_A^w(n) = \max\{t_A(x) : \text{len}(x) = n\}$ is bounded by a polynomial: $t_A^w(n) \leq P(n)$.

An algorithm is practically feasible if for all inputs of reasonable length, it finishes its computations in reasonable time.

Time complexity $t_A^w(n) = 10^{200}$ is a constant – thus a polynomial, so from the viewpoint of the formal definition, it is feasible. However, this number is larger than the number of particles in the Universe, so it is clearly not practically feasible.

On the other hand, the function $\exp(10^{-22} \cdot n)$ is an exponential function and thus, grows faster than a polynomial, but even for largest realistic lengths n – e.g., for $n = 10^{18}$ – the resulting value is smaller than 3 and is, thus, perfectly practically feasible.

Problem 4d. Briefly describe what is P, what is NP, what is NP-hard, and what is NP-complete. Is P equal to NP?

Solution: P is the class of all the problems that can be solved in polynomial time.

NP is the class of all the problems for which, once we have a candidate for a solution, we can check, in polynomial time, whether it is indeed a solution.

A problem is called NP-hard if every problem from the class NP can be reduced to this problem.

A problem is called NP-complete if it is NP-hard and itself belongs to the class NP.

At present, no one knows whether P is equal to NP. Most computer scientists believe that these two classes are different.

Problem 4e. Give an example of an NP-complete problem: what is given, and what we want to find.

Solution. An example of an NP-complete problem is propositional satisfiability:

- given: a propositional formula, i.e., any expression obtained from Boolean variables by using “and”, “or”, and “not”,
- find: the values of the Boolean variables that makes this formula true.

Problem 4f. Give definitions of a recursive (decidable) language and of a recursively enumerable (Turing-recognizable) language.

Solution. A language L is called *decidable* if there exists an algorithm (or, equivalently, a Turing machine) that:

- given a word,
- returns “yes” or “no” depending on whether this word belongs to this language or not.

A language is called *semi-decidable*, *Turing-recognizable*, or *recursively enumerable* if there exists a Turing machine that:

- given a word w ,
- returns “yes” if and only if the word w belongs to the language L .