Automata, Computability, and Formal Languages
Fall 2024, Solutions to Test 3

1. The Portuguese alphabet does not include the letter k, it uses q when the k sound is needed. The following Turing machine replaces each letter k with q:

- start, – → moving, R (here, – means blank)
- moving, $g \to$ R
- moving, $k \to q$, R
- moving, – → back, L
- back, $q \to$ L
- back, $g \to$ L
- back, – → halt

Trace it on the example of the word kg. Explain how each step will be represented if we interpret the Turing machine as a finite automaton with two stacks.

Solution:

Moment 1:

$\underline{-}$	k	g	–	–	...

start

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

–
k
g

Moment 2:

–	$\underline{k}$	g	–	–	...

moving

Here, the stacks have the following form:

–

k
g

Moment 3:

–	q	$\underline{g}$	–	–	...	moving

Here, the stacks have the following form:

q		g
–		

Moment 4:

–	q	g	$\underline{=}$	–	...	moving

Here, the stacks have the following form:

g		–
q		
–		

Moment 5:

–	q	$\underline{g}$	–	–	...	back

Here, the stacks have the following form:

g		–
q		
–		

Moment 6:

–	$\underline{q}$	g	–	–	...	back

Here, the stacks have the following form:

		q
–		g
		–

Moment 7:

$\underline{=}$	q	g	–	–	...	back

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

–
q
g
–

Moment 8:

$\underline{\_}$	q	g	–	–	...

halt

Here, the stacks are the same as in Moment 7.

2. Arithmetic operations on Turing machines:

   a Design a Turing machine that subtracts 8 from a binary number.

   b Trace your Turing machine, step-by-step, on the example of the decimal number 10.

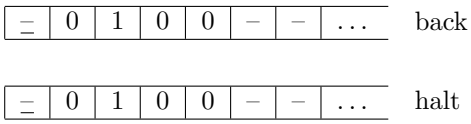
   c Why in Turing machines (and in most actual computers) the representation of a binary number starts with the least significant digit?

Solution: The idea is to skip the first three bit and then to use the algorithm for subtracting 1 from a binary number. Here are the rules:

start, – → skip1, R
skip1, 1 → skip2, R
skip1, 0 → skip2, R
skip2, 1 → skip3, R
skip2, 0 → skip3, R
skip3, 1 → move, R
skip3, 0 → move, R
move, 0 → 1, R
move, 1 → 0, back, L
back, 0 → L
back, 1 → L
back, – → halt.

Here is the tracing:

<u>–</u>	0	1	0	1	–	–	...	start
–	<u>0</u>	1	0	1	–	–	...	skip1
–	0	<u>1</u>	0	1	–	–	...	skip2
–	0	1	<u>0</u>	1	–	–	...	skip3
–	0	1	0	<u>1</u>	–	–	...	move
–	0	1	<u>0</u>	0	–	–	...	back
–	0	<u>1</u>	0	0	–	–	...	back
–	<u>0</u>	1	0	0	–	–	...	back



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.

The same representation is used for Turing machines, to make them closer to how actual computers work and thus, make them more realistic.

3. The following finite automaton checks whether a sequence of letters can be from a Portuguese word (p) or not (n): letters g and q are OK, k is not allowed. This automaton has two states: the starting state p which is also final, and the state n. Transitions are as follows:

- from the state p, symbols g and q lead back to p, while k leads to n;
- from the state n, each symbol leads back to n.

Use the general algorithm to transform this finite automaton into a Turing machine. Show, step-by-step, how your Turing machine will accept the string qg.

Solution: This Turing machine will have the following rules:

- start, $- \to p$, R
- p, $g \to p$, R
- p, $q \to p$, R
- p, $k \to n$, R
- n, $g \to n$, R
- n, $q \to n$, R
- n, $k \to n$, R
- p, $- \to$ accept
- n, $- \to$ reject

Tracing:

$\underline{-}$	q	g	$-$	$-$	$\ldots$	start
$-$	$\underline{q}$	g	$-$	$-$	$\ldots$	p
$-$	q	$\underline{g}$	$-$	$-$	$\ldots$	p
$-$	q	g	$\underline{-}$	$-$	$\ldots$	p
$-$	q	g	$\underline{-}$	$-$	$\ldots$	accept

4. Give the formal definition of a feasible algorithm, and an explanation of what practically feasible means. Give two examples different from what we had in class:

- an example of a computation time which is formally feasible, but not practically feasible, and
- an example of a computation time which is practically feasible but not formally feasible.

These examples should be different from what we had in class, in homeworks, in last year's solutions. To make sure that your examples are different, use some numbers that have personal meaning to you – e.g., your birthday or digits from your car's licence plate.

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: $t_A(x) \le P(\text{len}(x))$. 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: $t_A^w(n) \le P(n)$.

An algorithm is called practically feasible if for every input of reasonable length, it finished its computations in reasonable time.

Time complexity $t_A^w(n) = 10^{1905}$ 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 $\exp(10^{-1905} \cdot n)$ is an exponential function and thus, grows faster than an polynomial, but even for largest realistic lengths n – e.g., for $n = 10^{22}$ – the resulting value is smaller than 3 and is, thus, perfectly practically feasible.

5. What is P? What is NP? What does it means for a problem to be NP-hard? NP-complete? Give brief definitions. Give an example of an NP-complete problem: explain what is the input, what is the desired output. 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.

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.

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

6. 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.

7. 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 haltChecker(p,d) that always solves this program 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 haltChecker(aux,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 haltChecker(aux,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.

8. Give definitions of a decidable (recursive) language and of a semi-decidable (recursively enumerable, Turing-recognizable) language. Give an example of a decidable language and an example of a language which is semi-decidable but not decidable.

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.

An example is the language of all even numbers.

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.

An example of a semi-decidable language which is not decidable is the set of all the pairs (p, d) for which the program p halts on data d.

9. Prove that the cubic root of 6 is not a rational number.

Solution. Let us prove it by contradiction. Let us assume that $\sqrt[3]{6}$ is a rational number, i.e., $\sqrt[3]{6} = m/n$ for some integers m and n.

If the numbers m and n have a common factor, then we can divide both m and n by this factor and get the same ratio. Thus, $\sqrt[3]{6} = a/b$ for some integers a and b that have no common factors.

Let us now get a contradiction.

- Multiplying both sides of the above equality by b, we get $\sqrt[3]{6} \cdot b = a$.
- Cubing both sides, we get $6 \cdot b^3 = a^3$, i.e., $2 \cdot 3 \cdot b^3 = a^3$.
- The left-hand side of this equality is divisible by 2, so the right-hand side $a^3 = a \cdot a \cdot a$ must also be divisible by 2.
- Since 2 is a prime number, this means that one of the factors in the product $a \cdot a \cdot a$ – i.e., number a – is divisible by 2, i.e., $a = 2 \cdot p$ for some integer p.
- For $a = 2 \cdot p$, we have $a^3 = (2 \cdot p) \cdot (2 \cdot p) \cdot (2 \cdot p) = 2^3 \cdot p^3$.
- Substituting $a^3 = 2^3 \cdot p^3$ into the formula $2 \cdot 3 \cdot b^3 = a^3$, we get $2 \cdot 3 \cdot b^3 = 2^3 \cdot p^3$.
- Dividing both sides by 2, we get $3 \cdot b^3 = 2^2 \cdot p^3$.
- The right-hand side of this equality is divisible by 2, so the left-hand side $3 \cdot b^3 = 3 \cdot b \cdot b \cdot b$ must also be divisible by 2.
- Thus, b is divisible by 2.
- So, a and b have a common factor 2 – which contradicts to the fact that a and b have no common factors.

This contradiction shows that our original assumption – that $\sqrt[3]{6}$ is a rational number – is wrong. The statement is proven.