Automata, Computability, and Formal Languages
Fall 2022, Solutions to Test 3

1-2. Let L be language of all the words that contain equally many digits 0, 1, 2, and 3:

$$L = \{\Lambda, 0123, 3012, \ldots, 00112233, 01302132, \ldots\}.$$

Prove that this language is not context-free.

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 $\text{len}(vy) > 0$, $\text{len}(vxy) \le p$, and for every i, we have $uv^ixy^iz \in L$.

Let us take the word $w = 0^p1^p2^p3^p \in L$, in which first we have symbol 0 repeated p times, then symbol 1 repeated p times, then symbol 2 repeated p times, and then symbol 3 repeated p times. The length of this word is $p + p + p + p = 4p > p$, 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 three different digits, e.g., 0s, 1s, and 2s – otherwise, it will have to contain all the symbols 1 between 0s and 2s – there are p of these symbols, and also at least one of 0s and one of 2s, to the total of more than p. So, we have the following possible cases:

- vxy is in 0s;
- vxy is between 0s and 1s;
- vxy is in 1s;
- vxy is between 1s and 2s;
- vxy is in 2s;
- vxy is between 2s and 3s; or
- vxy is in 3s.

In the first case, v and y contain only 0s. So, when we go from $uvxyz$ to $uvvxyyz$, we add 0s, but we do not add 1s, 2s, or 3s; thus, the desired balance between numbers of 0s, 1s, 2s, and 3s 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 0s and 1s. So, when we go from $uvxyz$ to $uvvxyyz$, we add 0s and 1s, but we do not add any 2s or 3s; thus, the desired balance between numbers of 0s, 1s, 2s, and 3s 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.

3. The following Turing machine takes words consisting of a's and b's, and replaces small a's with capital A's:

- start, – → work, R (here, – means blank)
- work, a → A, R
- work, b → R
- work, – → back, L
- back, A → L
- back, b → L
- back, – → halt

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

Solution:

Moment 1:

$\underline{-}$	a	b	–	...	start

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

–
a
b

Moment 2:

–	$\underline{\text{a}}$	b	–	...	work

Here, the stacks have the following form:

	a
–	b

Moment 3:

–	A	$\underline{\text{b}}$	–	...	work

Here, the stacks have the following form:

A	b
–	

Moment 4:

−	A	b	$\underline{-}$	. . .	work

Here, the stacks have the following form:

b	
A	−
−	

Moment 5:

−	A	$\underline{\text{b}}$	−	. . .	back

Here, the stacks have the following form:

A	b
−	−

Moment 6:

−	$\underline{\text{A}}$	b	−	. . .	back

Here, the stacks have the following form:

	A
−	b
	−

Moment 7:

$\underline{-}$	A	b	−	. . .	back

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

−
A
b
−

Moment 8:

$\underline{-}$	A	b	−	. . .	halt

Here, the stacks do not change.

4. Arithmetic operations on Turing machines:

   a Design a Turing machine that subtracts 2 from a unary number. Assume that the original number is greater than or equal to 2.

   b Trace your Turing machine, step-by-step, on the example of the number 3.

   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 get to the end of the word – i.e., to reach a blank space, then delete the first 1, the second 1, and get back. Here are the rules:

start, – $\to$ move, R
move, 1 $\to$ R
move, – $\to$ delete1, L
delete1, 1 $\to$ –, delete2, L
delete2, 1 $\to$ –, back, L
back, 1 $\to$ L
back, – $\to$ halt.

Here is the tracing:

$\underline{-}$	1	1	1	–	–	–	...	start
–	$\underline{1}$	1	1	–	–	–	...	move
–	1	$\underline{1}$	1	–	–	–	...	move
–	1	1	$\underline{1}$	–	–	–	...	move
–	1	1	1	$\underline{-}$	–	–	...	move
–	1	1	$\underline{1}$	–	–	–	...	delete1
–	1	$\underline{1}$	–	–	–	–	...	delete2
–	$\underline{1}$	–	–	–	–	–	...	back
$\underline{-}$	1	–	–	–	–	–	...	back
$\underline{-}$	1	–	–	–	–	–	...	halt

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.

5. The following finite automaton describes strings that end in 0:

- the starting state e; this state means that either we have not read any symbols yet or that the last read symbol was not 0; and
- the final state f meaning that the last symbol we read was 0.

Transitions are as follows:

- from the state e, symbol 0 leads to the state f and symbol 1 leads back to the state e;
- from the state f, symbol 1 leads to the state e and 0 leads back to the state f.

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 110.

Solution: This Turing machine will have the following rules:

- start, $-\to$ e, R
- e, $0 \to$ f, R
- e, $1 \to$ e, R
- f, $0 \to$ f, R
- f, $1 \to$ e, R
- e, $- \to$ reject
- f, $- \to$ accept

Tracing:

$\underline{-}$	1	1	0	$-$	...	start
$-$	$\underline{1}$	1	0	$-$	...	e
$-$	1	$\underline{1}$	0	$-$	...	e
$-$	1	1	$\underline{0}$	$-$	...	e
$-$	1	1	0	$\underline{-}$	...	f
$-$	1	1	0	$\underline{-}$	...	accept

6. 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.

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

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

Time complexity $t^w_A(n) = 10^{200}$ 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 an 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.

7. 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.

8. Prove that the square root of 24 is not a rational number.

Solution: Let us prove this by contradiction. Let us assume that $\sqrt{24}$ is a rational number, i.e., that $\sqrt{24} = a/b$ for some natural numbers a and b. Without losing generality, we can assume that a and b have no common factors – if they had, we could divide both numerator and denominator by this common factor.

Squaring both sides of this equality, we get $24 = a^2/b^2$. Multiplying both sides by b^2, we get

$$a^2 = 24 \cdot b^2 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot b^2.$$

The right-hand side of this equality divides by 2, so the left-hand side $a \cdot a \cdot a$ must be divisible by 2 as well. This means that one of the factors in the left-hand side product must be divisible by 2, i.e., that a is divisible by 2. This means that $a = 2p$ for some natural number p. Substituting $a = 2p$ into the equality $a^2 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot b^2$, we conclude that $2 \cdot 2 \cdot p^2 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot b^2$ i.e., dividing both sides by $2 \cdot 2$, that $p^2 = 2 \cdot 3 \cdot b^2$.

Again, the right-hand side of this equality divides by 2, so the left-hand side $p \cdot p$ must be divisible by 2 as well. This means that one of the factors in the left-hand side product must be divisible by 2, i.e., that p is divisible by 2. This means that $p = 2q$ for some natural number q. Substituting $p = 2q$ into the equality $p^2 = 2 \cdot 3 \cdot b^2$, we conclude that $2 \cdot 2 \cdot q^2 = 2 \cdot 3 \cdot b^2$ i.e., dividing both sides by 2, that $2 \cdot q^2 = 3 \cdot b^2$.

Now, the left-hand side is divisible by 2, so similarly to the above argument we can conclude that b is also divisible by 2. Thus a and b have a common factor 2 – but a and b have no common factors. This contradiction proves that our assumption is wrong, and so $\sqrt{24}$ is not a rational number.

9. Formulate the halting problem. Prove that it is not possible to check whether a given program halts on given data.

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.

10. 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.