Theory of Computations,
Test 2 for the course
CS 5315/6315, Spring 2017

General comments:

20
20

- you are allowed up to 5 pages of handwritten notes;
- if you need extra pages, place your name on each extra page.

Good luck!

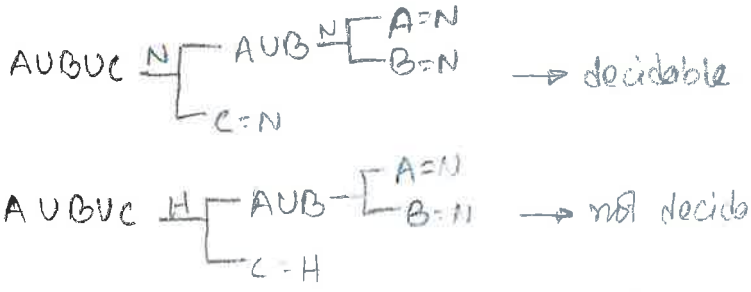
1-2. Let A be decidable, B be decidable, and C be r.e.

- is the union of the three sets r.e.? decidable?
- is the intersection of the three sets r.e.? decidable?
- is the complement to the union of the three sets r.e.? decidable?
- is the complement to the intersection of the three sets r.e.? decidable?

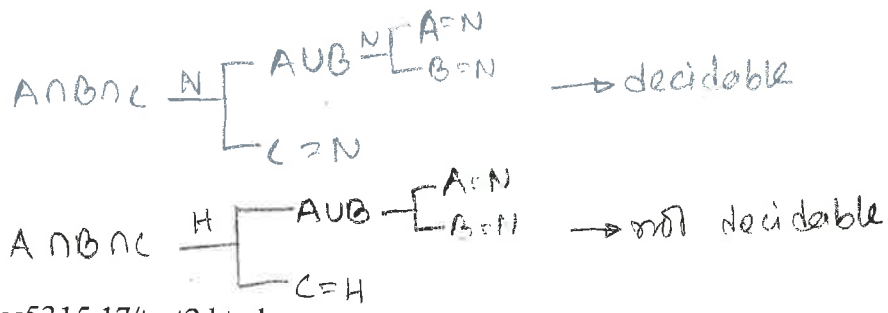
For each of these questions, either provide a proof, or provide a counterexample. In your proofs, you can use the results that we proved in class, without the need to reproduce the proofs of these auxiliary results.

A – decidable ∈ r.e.
B - decidable ∈ r.e.
C – r.e.

- the union of A,B,C will be r.e. As A,B,C is r.e.



- The intersection of A,B,C will be r.e.



10/10

3. Suppose that both the set A and its complement −A are r.e. Suppose that number 6 belongs to A, and the algorithm for generating elements of set A produces number 6 at moment 3. At what moment of time will the deciding algorithm (i.e., the algorithm for checking whether a given number belongs to A) inform us that 6 belongs to the set A? Explain your answer.

if A and -A are r.e., then A is decidable.

So, we run A for 1 hr
-A for 1 hr

:

n will eventually appear in one of the lists.

$t_A = 3$

$(2t_A - 1) = (6-1) = 5$

So, after moment 5, the algorithm will tell us 6 belongs to set A.

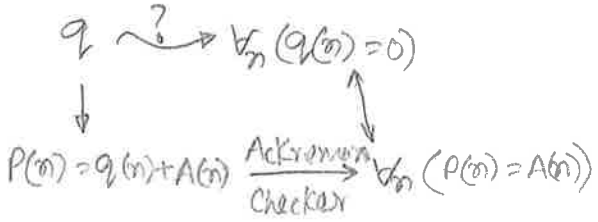
10/10

4. Prove that no algorithm is possible that, given a program p, checks whether this program always computes the value of the Ackermann function A(n), i.e., whether p(n) = A(n) for all n.

Lets assume that, Ackermann-function (A(n)) checker exists

$$\text{Ackermann function checker}(P) = \begin{cases} 1, & \text{if } \forall n\,(P(n) = A(n)) \\ 0, & \text{if } \exists n\,(P(n) \neq A(n)) \\ \text{whatever}, & \text{if } P \text{ doesn't always halt.} \end{cases}$$

we will build a zero-checker based on this program



$$\text{zero-checker}(q) = \text{Ackermann-function-checker}(q(n)+A(n))$$

$$q(n) = 0 \longleftrightarrow q(n) + A(n) = A(n)$$

if $q(n) \equiv 0$ then $p(n) = q(n) + A(n) = A(n)$

if $\exists (q(n) \neq 0)$ then $p(n) = q(n) + A(n) \neq A(n)$

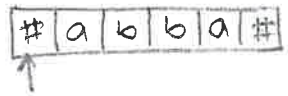
So we have build an Ackermann function checker assuming that zero checker exists. But zero-checker isn't possible. So Ackermann function checker isn't possible.

5-6. Write down the rules of a Turing machine that checks whether the word written on a tape is a palindrome, i.e., whether:

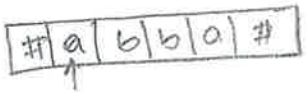
- the first letter is equal to the last one,
- the second letter is equal to the last-but-one letter, etc.

20/20

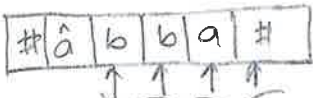
For simplicity, assume that the words only use letter 'a' and 'b'. Trace your newly designed sign-inverting Turing machine on the example of the word abba.



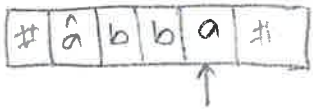
Start



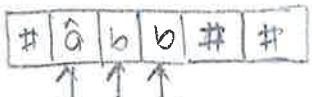
Checking



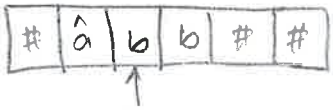
carry a



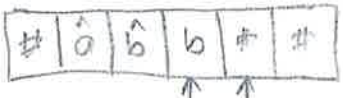
backa



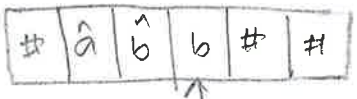
back



Checking



carry b



backb

back

Start, # → checking, R

Checking, # → halt, L

Checking, a → â, carry a, R

Checking, b → b̂, carry b, R

Carry a, a → R

Carry a, b → R

Carry a, # → backa, L

backa, a → #, back, L

backa, b → halt-not palindrom

back, a → L

back, b → L

back, â → checking, R

back, b̂ → checking, R

Carry b, a → R

Carry b, b → R

Carry b, # → backb, L

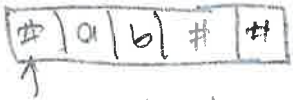
backb, a → halt, not palindrom

backb, b → #, back, L

back, â → a, L

back, b̂ → b, L

back, # → halt-palindrom



halt-palindrom

7. Write down the Church-Turing thesis.

- Is it a mathematical theorem?
- Is it a statement about the physical world?

10/10

Church-Turing thesis:

Anything that can be computed on any physical device can also be computed on a turing machine or programmed by a Programming language.

- It's not a mathematical theorem
- It's a statement about physical world.

8. To solve an equation $p * u^2 - q / u^2 = r$ with an unknown u, we multiply both sides by u^2 and introduce a new variable $v = u^2$. This reduces our original problem to the problem of solving an appropriate quadratic equation $a * v^2 + b * v + c = 0$. Once we know v, we can compute u as the square root of v. Describe, for this reduction:

- what is x, what is y, what is C(x, y),
- what is x', what is y', what is C'(x', y'),
- what is U_1,
- what is U_2, and
- what is U_3.

10/10

$pu^v - q/u^v = r$

$\Rightarrow pu^4 - q = ru^v$

$\Rightarrow pu^4 - ru^v - q = 0$

$av^v + bv + c = 0$ - reduced form

$x = (p, q, r)$ $\quad y = u$ $\quad c(x,y) = (pu^v - q/u^v = r)$

$x' = (a, b, c)$ $\quad y' = v$ $\quad c'(x',y') = (av^v + bv + c = 0)$

$u_1(x) = (p, -r, -q)$

$u_1(v) = \sqrt{v}$

$u_1(u) = u^v$

9. Give a formal definition of feasibility. Give two examples:

10/10

- an example when an algorithm is feasible in the sense of the formal definition but not practically feasible, and
- an example when an algorithm is practically feasible, but not feasible according to the formal definition.

These examples must be different from the examples that we had in class.

An algorithm is feasible if it is polynomial time. example: search, sorting.

example 1:

$t_A(x) = 10^{20} \cdot len(x)$

- not feasible from practical viewpoint
- perfectly feasible from the viewpoint of definition

example 2:

$t_A(x) = \exp(10^{100} \cdot len(x))$

- feasible from practical viewpoint
- not feasible from the viewpoint of definition

10. Define: 8/10

- what is P,
- what is NP, and
- what is NP-hard.

P: Class of problems that can be solved in polynomial times.

$t_A(x) \leq P(len(x))$

NP: Class of all problems, for which, once we get a candidate for a solution, we can check, in polynomial time, whether it is needed a solution.

NP-hard: Any NP problem can be reduced to a $\widetilde{NP}$ hard problem

A problem is NP-hard if every NP problem can be reduced to it.