Habib University – City Campus
Course: CS 212 Nature of Computation
Instructors: Asma Larik (L1), Waqar Saleem (L2), Abdullah Zafar (L3)
Term : Fall 2022
Examination: Final Term Date: 12 Dec, 2022
Total Marks: 65 Duration: 135 minutes

Instructions:

1. You are allowed to keep only the following items with you on your table: writing utensils, stationery box, your HU ID card, your "cheat sheet", and a drinks container. Please deposit other items, e.g. bags and devices at the front of the classroom.
2. Make sure that your HU ID card is clearly visible on your table.
3. Your cheat sheet must meet the requirements shared earlier. Non-compliant cheat sheets will be confiscated.
4. Please ensure sufficient writing utensils in working condition. Do not bother others during the exam.
5. The problems in this exam rely on argumentation. Make sure to provide sound justifications that are both precise and concise.
6. If your answer to a problem exceeds 10 lines, it is very likely that you are on the wrong track.
7. Please submit this problem paper and your cheat sheet with your answer book.
8. This exam consists of 6 problems for 75 points, printed on 5 sides.
9. Attempt **any 65 points**. It is not necessary to attempt all parts of a problem.
10. You may attempt the problems in any order provided that they are clearly labeled.
11. In case you attempt more than the required points, only the first 65 points will be graded.

viel Spaß und viel Glück!

1. Computability [35 points]

For each of the prompts below, provide short definitions, comments, or (counter-) arguments as appropriate.

(a) 5 points Turing decidability.

Solution: A language, L, is Turing decidable if there exists a Turing Machine, M, that accepts every $w \in L$ and rejects every $w \notin L$.

(b) 5 points Turing recognizability

Solution: A language, L, is Turing decidable if there exists a Turing Machine, M, that accepts every $w \in L$.

(c) 5 points The relation between *computability* and Turing recognizability or decidability.

Solution: A decidable language is computable. A Turing machine that decides it will eventually halt. A recognizable language may not be computable as a Turing machine that recognizes it may not halt.

(d) 5 points Every language is computable.

Solution: Saying that a language is computable is to say that it is decidable. Not all languages are decidable. So the statement is False.

(e) 5 points Every problem can be expressed as a language.

Solution: A language that expresses a problem can be checked to contain or not contain a given string. The answer therefore is binary. Not every problem has a binary answer. Thus, not every problem can be expressed as a language.

(f) 5 points A Turing machine can perform any possible computation.

Solution: The Church-Turing thesis establishes that any algorithm with a finite number of steps in which every step does a finite amount of work can be expressed as a Turing machine. If we consider algorithms that involve an infinite amount of work, whatever tease may be, they will be beyond the scope of a Turing machine. We have already encountered examples of languages that cannot be recognized, let alone decided, by a Turing machine.

(g) 5 points The complement of A_{TM} is Turing recognizable.

Solution: We prove by contradiction that $\overline{A_{TM}}$ is not Turing recognizable.

Proof. Assume that $\overline{A_{TM}}$ is Turing recognizable.
It is known that A_{TM} is Turing recognizable.

Then A_{TM} is decidable.
But A_{TM} is not decidable. $\perp$ $\square$

2. Halting Problem [5 points]

What is the halting problem and what, in your view, is its significance?

Solution: The halting problem is to determine whether a given algorithm will eventually halt on a given input. It turns out that the answer is no. In general, given an algorithm and an input, it cannot be determined whether the algorithm will halt. This conveys that there is at least one problem that a computer cannot solve. And if there is one, there may be others. This result is also counter-intuitive, as it originally seems that the problem must have an easy positive answer. Also, once we know that one problem is undecidable, we can use it along with reducibility to argue about the undecidability of other problems.

3. Mapping Reducibility [5 points]

What is meant by mapping reducibility and what good is it to know that a language A is mapping reducible to another language B?

Solution: A language, A, is mapping reducible to another language, B, if there is a computable function, $f : \Sigma^* \to \Sigma^*$, where $w \in A \iff f(w) \in B$. A is said to reduce to B or be mapping reducible, or simply reducible to B. If that is the case, then if B can be solved, then so can A. And if A cannot be solved, then neither can B. This can be used to solve A using known solutions of B, and to prove that B is undecidable if A is known to be undecidable.

4. Polynomial Time Reducibility [5 points]

What is meant by polynomial time reducibility and what good is it to know that a language A is polynomial time reducible to another language B?

Solution: A language, A, is polynomial time reducible to another language, B, if there is a polynomial time computable function, $f : \Sigma^* \to \Sigma^*$, where $w \in A \iff f(w) \in B$. If A is polynomial time reducible to B, then if B can be solved in polynomial time, then so can A. And if A cannot be solved in polynomial time, then neither can B. This can be used to find polynomial time solutions to A using known polynomial time solutions of B, and to prove that B cannot be solved in polynomial time if A is known to be unsolvable in polynomial time.

5. P vs. NP [10 points]

(a) [5 points] What is the **P** vs. **NP** problem and what, in your view, is its significance?

Solution: Problems in **P** are those that can be *decided* in polynomial time. Problems in **NP** are those that can be *verified* in polynomial time. **P** is a subset of **NP**. The problems in **NP** that are also in **P** can be decided in polynomial time. It is not known whether the other problems in **NP** can be decided in polynomial time or not. The **P** vs. **NP** problem is to decisively conclude whether such problems can or cannot be decided in polynomial time. The best known algorithms to such problems all take exponential time. A solution to the **P** vs. **NP** problem will either result in polynomial time algorithms for problems that previously required exponential time, or establish that the search for such algorithms is futile.

(b) [5 points] What is **NP**-completeness and how can it assist in finding a solution to the **P** vs. **NP** problem?

Solution: An **NP**-complete problem is a problem in **NP** to which all other **NP** problems reduce in polynomial time. To solve the **P** vs. **NP**, an argument needs to be made about *every* problem in **NP** which is a daunting task. As all **NP** problems reduce to **NP**-complete problems in polynomial time, the arguments can instead be made for *any* **NP**-complete problem instead.

6. Complexity [15 points]

(a) [10 points] Prove that the language, $FACT$, is in **P** where

$$FACT = \{\langle x, n\rangle \mid n! \text{ is the closest factorial lesser than or equal to } x\}$$

Solution: We prove that $FACT$ is not in **P** by showing that any algorithm for it cannot run in polynomial time.

Proof. In general, the algorithm to decide $FACT$ needs to compute $n!$. For this, it needs the magnitude of n in order to compute the $O(n)$ products for $n!$. But what it is given is $\langle n\rangle$, the string representation of n on the tape. The magnitude of the number is an exponential function of the length of its representation. (This was also the insight in WC 10). Therefore, the algorithm is not in **P**. □

(b) [5 points] Prove that the language $COLOR$, is in **NP** where

$COLOR = \{G \mid$ there is a way of assigning colors to the vertices of the graph G, such that no two adjacent vertices are assigned the same color, and the entire graph can be colored with 4 colors$\}$

Solution: The problem can be restated as deciding that a given graph, G, is 4-colorable. We prove that $COLOR$ is in **NP** by showing that it can be verified in polynomial time.

Proof. Let the certificate for the verifier be a coloring, C, of G, i.e. a function from the vertices in G to the set of colors. The size of C is $O(V)$ where V is the number

of vertices in G. Following is a verifier whose complexity is a polynomial function of V.

```
colors ← {} # empty set
for (v, c_v) ∈ C do
    colors.add(c_v)
    for (w, c_w) ∈ C where (v, w) is an edge in G do
        colors.add(c_w)
    end for
end for
if len(colors) ≤ 4 then
    ACCEPT
else
    REJECT
end if
```

□