• When handing in this test, please go to a TA with your GTID to verify the name on the test.

• Do not open this quiz booklet until you are directed to do so. Read all the instructions first.

• You have 75 minutes to earn up to $1 + 7 + 6 + 5 + 7 = 26$ points, the test is graded out of 25.

• This booklet contains 4 questions on 7 pages, including this one. Please note that there are questions on the back of sheets as well.

• Write your name and user id (letters, not numbers) on the top of every page.

• Write your solutions in the space provided. If you run out of space, continue your answer on the back of the last page, and make a notation on the front of the sheet.

• You may use a sheet with notes on both sides.

• All numbers on this test are non-negative integers, and the largest ones outside of the box at the top of this page are the ones indicating points. So calculators are not permitted.

• You may use any of the theorems/facts/lemmas from the lecture notes, homeworks, or textbook without re-proving them unless explicitly stated otherwise. However, we may check with you about the citation / reference afterwards, possibly via regrades.

• If necessary make reasonable assumptions but please be sure to state them clearly

• Do not spend too much time on any one problem. Generally, a problem’s point value is a good indication of how many minutes to spend on it.

• Good luck!
0 ( /1 point) Write your name and user id on top of every page.

1. ( /7 points)

Here is the state diagram of a Turing Machine $M_1$ with starting state is $q_1$.

(a) (1 point) Give the sequence of configurations that that $M_1$ enters on the input string 11:

**SOLUTION:**

$q_111 \Rightarrow Xq_51 \Rightarrow X1q_5\omega \Rightarrow Xq_61 \Rightarrow q_7XX \Rightarrow q_7XX \Rightarrow q_1XX \Rightarrow Xq_1X \Rightarrow XXq_1\omega \Rightarrow XXq_{accept}\omega$

(b) (1 point) Give the sequence of configurations that that $M_1$ enters on the input string 001:

**SOLUTION:**

$q_1001 \Rightarrow Xq_901 \Rightarrow X0q_21 \Rightarrow X01q_2\omega \Rightarrow X0q_31 \Rightarrow q_{reject}$

(c) (2 points) Give the sequence of configurations that that $M_1$ enters on the input string 1001:

**SOLUTION:**

$q_11001 \Rightarrow Xq_5001 \Rightarrow X0q_51 \Rightarrow X00q_51 \Rightarrow X001q_5\omega \Rightarrow X00q_61 \Rightarrow X0q_70X \Rightarrow Xq_70X \Rightarrow q_7X00X \Rightarrow q_7X00X \Rightarrow Xq_100X \Rightarrow XXq_20X \Rightarrow XXq_20x \Rightarrow XX0Xq_2\omega \Rightarrow XX0Xq_3X \Rightarrow XXq_30X \Rightarrow Xq_4XX \Rightarrow q_4XX \Rightarrow \omega \Rightarrow q_4XX \Rightarrow q_4XXXX \Rightarrow q_4XXXX \Rightarrow q_1XXXX \Rightarrow Xq_1XXXX \Rightarrow XXq_1XX \Rightarrow XX\omega \Rightarrow XXXXXq_1\omega \Rightarrow XXXXXq_{accept}\omega$  

**GRADING:**

For (a), (b), (c),

i. It’s acceptable to have no explicit tape configuration for $q_{accept}$ and $q_{reject}$.

ii. For every part, a mistake(wrong state) will get 0.5 points deducted. Once a mistake is made, we will assume it’s correct and keep going.

For example, if a student copy the wrong initial state $q_{10}$ for (a), then we should do -0.5 and check whether the rest part is correct assuming the initial state is $q_{10}$.
iii. If the representation is wrong, then do -2 points from 4 points (1 + 1 + 2). Don’t deduct points repeatedly for the same reason. The 4 points are deducted as -0.5 from part a, -0.5 from part b and -1 point from part c.

(d) (1 point) Note: this is still about $M_1$, but got bumped to here due to space reasons. Please don’t skip over this. give a succinct description of $L(M_1)$, the set of words accepted by $M_1$.

**SOLUTION:**
Binary strings of even length that read the same forward and backward, aka. binary palindromes.

**GRADING:**
- 0.5 anything that says ‘front same as back’, definition of palindromes not important.
- 0.5 even length

Here is a state diagram of a nondeterministic TM $M_2$:

![State Diagram](image)

Given each of the following states, write down all possible configuration for the next step.

(e) (1 point) $00q_200$ **SOLUTION:**
$001q_20, 0q_3000$

(f) (1 point) $00q_300$ **SOLUTION:**
$000q_{accept}0$ or $q_{accept}, 000q_10$

**GRADING:**
(a) For every part, each possible configuration counts for 0.5 point. It’s acceptable to have no explicit tape configuration for $q_{\text{accept}}$ and $q_{\text{reject}}$.

(b) -0.5 for more than 2 configurations. If there are 3 configurations written and only 1 is correct, 0.5 will be deducted once.
2. ( /6 points)
Construct a Turing machine that decides the following language over the alphabet $\Sigma = \{a, b\}$:

$L = \{w : w \text{ has the same number of } a \text{ and } b\}.$

For example, $abab, abba \in L$ (and should be accepted by this Turing machine), while $a, bab, bbaa \notin L$, and should be rejected.

Make sure to explicitly state the tape symbols like in Homework 1, and provide the tape symbols, states and transitions via a state diagram.

**SOLUTION:**

We will build a TM with tape symbols

$$\Gamma = \{a, b, \omega, x\}.$$  

State diagram of a Turing machine is below: the starting state is $q_0$.

![State Diagram](image)

**GRADING:**

1 point for list of explicitly used tape symbols.

1 point for empty string case.

2 points for state diagram that accepts generally accepted strings such as $aabb, abba, baab$.

2 points for rejecting all invalid string inputs such as $aab, baaab, bbaabb$.

Deduct 1 point for missing transitions, unclear state diagram (over complicated), and any other extraneous errors. Any points that are deducted for this reason should be clearly commented during grading and carefully considered.
3. ( /5 points)  

Consider the following language over the alphabet \( \Sigma = \{0, 1, \#\} \):

\[
L = \{a\#b\#c \mid a, b, c \text{ are binary numbers of the same length, and } a \oplus b = c\},
\]

here \( \oplus \) is the bit-wise XOR, that is, it’s 1 if the corresponding bits are different, and 0 if same. For example, \( 0\#1\#1, 11\#10\#01 \in L \), \( 1###, 11\#1\#0, 0\#0\#1 \notin L \).

Give implementation descriptions of a (multi-tape) Turing machine that decides \( L \), make sure to explicitly specify the tape symbols.

**SOLUTION:**  
The following solution requires tape symbols \( \{0, 1, \#, _\} \).

First, we can check if the input string is of the form \( a\#b\#c \) for some binary \( a,b,c \). This is achieved by moving left to right across the input until the blank symbol is found, and keeping a count of the number of \# symbols found. If the condition is not met, we can reject at this stage.

We can copy each binary \( a,b,c \) to a new tape for easier handling. This can be achieved by moving left to right across the input string, copying 0,1 symbols until the terminating character (\# for \( a \) and \( b \), and the blank symbol for \( c \)) is found.

We should ensure that \( a,b,c \) have the same length. The simplest way to do this without overwriting the symbols on their own tape is to use another blank tape. For every symbol in \( a \), write a zero to the blank tape. Then for every symbol in \( b \), overwrite one of the zeros on the new tape with a 1. If there are no zeros to overwrite at any step, we can reject since the size of \( b \) is larger than the size of \( a \). Similarly if all the 0s are not overwritten at the end, then we can reject as well. Finally we can ensure the length of \( c \) is the same by overwriting 1s with 0s.

Ensuring that \( a \oplus b = c \) can be done by moving left to right across each tape simultaneously one symbol at a time. There are four possible valid transitions for each bit in \( a \) and \( b \). If the corresponding bit in \( c \) does not match up, reject. If we pass through the entire length without a reject, we can accept at this stage.

**GRADING:**  
0.5 points for stating correct tape symbols.

0.5 points for ensuring exactly two \# symbols and 3 binary numbers are present.

1 point for correctly iterating over the strings and copying them onto new tapes.

1.5 points for ensuring size of \( a, b, c \) are equal.

1.5 points for ensuring \( a \oplus b = c \).
4. ( /7 points)
Multiple choice, circle one of the choices:

(a) ( /1 point) What is the abbreviation NP stand for?
   i. Non-Polynomial Time.
   ii. Non-deterministic Polynomial Time.
   iii. Not Provable.

   SOLUTION:
   ii. Non-deterministic Polynomial Time.

(b) ( /1 point) If a language \( L_1 \) satisfies \( 3\text{-SAT} \leq_p L_1 \), what can we infer about \( L_1 \)?
   i. \( L_1 \) is in NP.
   ii. \( L_1 \) is NP-hard.
   iii. \( L_1 \) is NP-complete.

   SOLUTION:
   ii. \( L_1 \) is NP-hard

(c) ( /1 point) If a language \( L_2 \) satisfies \( L_2 \leq_p 3\text{-SAT} \), what can we infer about \( L_2 \)?
   i. \( L_2 \) is in NP.
   ii. \( L_2 \) is NP-hard.
   iii. \( L_2 \) is NP-complete.

   SOLUTION:
   i. \( L_2 \) is in NP.
(d) (4 points) Consider the 5-SAT problem, which is a satisfiability instance where each clause has 5 literals. That is, it’s a conjunction over clauses of the form of

\[ l_{i_1} \lor l_{i_2} \lor l_{i_3} \lor l_{i_4} \lor l_{i_5}, \]

and each \( l_{ij} \) is either \( x_k \) or \( \neg x_k \) for some variable \( x_k \). Such an instance is satisfiable if there is an assignment to the variables \( x_1 \ldots x_n \) so that each clause has at least one literal that evaluates to \( T \).

Show that 5-SAT is NP-complete. For the hardness portion, your reduction should be from 3-SAT (aka. show \( 3\text{-SAT} \leq_p 5\text{-SAT} \)).

**SOLUTION:**

A non-deterministic TM can guess an assignment to 5-SAT and accepts if the assignment satisfies the given formula. Hence, 5-SAT \( \in \text{NP} \).

Reduction: \( 3\text{-SAT} \leq_p 5\text{-SAT} \)

For every clause of the form \( (x \lor y \lor z) \), convert it into \( (x \lor y \lor z \lor a \lor b) \land (x \lor y \lor z \lor \neg a \lor \neg b) \land (x \lor y \lor z \lor \neg a \lor \neg b) \land (x \lor y \lor z \lor \neg a \lor \neg b) \) where \( a \) and \( b \) are arbitrarily set.

If any clause of the form \( (x \lor y \lor z) \) is satisfied, the corresponding clauses \( (x \lor y \lor z \lor a \lor b) \land (x \lor y \lor z \lor \neg a \lor \neg b) \) will also be satisfied. Thus, a satisfying assignment to 3-SAT will produce a satisfying assignment to 5-SAT instance.

If the set of clauses in 5-SAT: \( (x \lor y \lor z \lor a \lor b) \land (x \lor y \lor z \lor \neg a \lor \neg b) \land (x \lor y \lor z \lor \neg a \lor \neg b) \), the corresponding clause in 3-SAT \( ((x \lor y \lor z)) \) will also be satisfied. This is because for any combination of \( a \) and \( b \), the value of the set of clauses will be False if \( (x \lor y \lor z) \) is False. Thus, if the new instance of 5-SAT is satisfied, the original clauses of 3-SAT are satisfied by dropping \( a \) and \( b \).

Adding new literals to create new clauses can be done in polynomial time.

Hence, 5-SAT \( \in \text{NP} \) – Hard.

Since 5-SAT \( \in \text{NP} \) and 5-SAT \( \in \text{NP-Hard} \), 5-SAT is NP-Complete.

**GRADING:**

1 point for proving that 5-SAT belongs to NP.
1 point for providing a reduction from 3-SAT to 5-SAT.
When doing reduction, you can directly add T or F literals.
1 point for showing that a satisfying assignment to 3-SAT will create a satisfying assignment to 5-SAT (on applying the reduction).
1 point for showing the converse, i.e. a satisfying assignment for 5-SAT will produce a satisfactory assignment for 3-SAT.