Name: ________________________________

Student ID: ________________________________

CS181 Winter 2019 - Final

Due Friday, March 15, 11:59 PM

- This exam is open-book and open-notes, but any materials not used in this course are prohibited, including any material found on the internet. **Collaboration is prohibited.** Please avoid temptation by not working on the final while you are in the presence of any other student who has taken or is currently taking CS181. **Be extra careful if you live with or meet regularly with a student of this class.** If you have any questions about the exam, ask the TA or Professor Sahai by email or after class. **Do not ask other students.** You are allowed to use any theorem shown in class or in the textbook, as long as you clearly cite it. Please monitor Piazza for any clarifications. **Do not** post any questions on piazza.
- We suggest that you spend approximately 12 hours (not necessarily contiguous) to take this exam. Start early so that you have time to understand and think about the problems. **The solutions must be submitted on Gradescope by 11:59 PM on Friday, March 15.**
- Place your name and UID on every page of your solutions. Retain this cover sheet and the next sheet with the table as the first pages of your solutions. **Please use separate pages for each question. All problems require clear and well-written explanations.**
- There are 4 questions worth a total of 230 points and an extra credit question worth 40 points.
- For each part (except for the extra credit), if you describe a non-trivial approach that you tried using to solve the problem but realized it doesn't work and explain correctly why it doesn't work and then write "I don't know" you will get 20% points for that problem. You will **not** get 20% points for just writing "I don't know". Whether your stated approach was indeed non-trivial is solely at the discretion of the grader.
- 5% extra credit will be awarded to solutions written in LaTeX.

Please **handwrite** the following honor code agreement and sign and date in the spaces provided.
Honor Code Agreement: I promise and pledge my honor that during the exam period, I did not and will not talk to any person about CS 181 material except for the professor or the TA, nor will I refer to any material except for the class textbook and my own class notes. I will abide by the CS181 Honor Code.

Signature: ________________________________

Date: ________________________________

Question	**Points**	
1		
2		
3		
4		
EC		
Total		

Problem 1

(a) Intuition: To prove undecidability, the most straightforward way is to build a Turing Machine that can cause a contradiction using recursion theorem.

Assume there is a Turing Machine M that decides L_{height}. We can build a Turing Machine F in the following way:

```
F = "On input w:"
  • Obtain, via the recursion theorem, own description $< F >$
  • Run $M$ on input $< F >$
  • if $M$ accepts:
        reject $w$
    if $M$ rejects:
        accept $w$
```

Hence, there are two cases:

Case 1: F accepts all $w \in \Sigma^*$. This happens when M rejects on input $< F >$. Hence, since $|M| \geq 1$, there exists $w \leq |M|$ which is a contradiction, since height$(M) \leq |M|$.

Case 2: F rejects all $w \in \Sigma^*$. This happens when M accepts on input $< F >$. Hence, since $|M|$ is finite, height$(F) = \infty > |M|$, which is a contradiction.

Hence, the construction of F proves the undecidability of L_{height}

(b) Intuition: we can construct a TM that recognizes L_{height} by enumerate all the strings with length smaller than the length of the TM description. The actual construction is similar to the proof of co-recognizability of E_{TM}.

Since the alphabet Σ and the length of the input TM is finite, there are a finite number of possible input strings with length smaller than $|M|$ for the TM M. Thus, we can run all of them in the recognizer to see if any of them accepts. If any of them is accepted, then $M \in L_{height}$. However, one problem is, if one of the strings causes the recognizer to loop forever, it will never have the chance to run the next string. To avoid this problem, we can build a recognizer R in this way:

```
R = "On input $< M >$, where $M$ is a TM:"
  • Repeat the following for $i = 1, 2, 3, ...$
  • Run $M$ for $i$ steps for each of possible input strings with length smaller than $|M|$ (we have stated above the set of input strings is finite).
  • if any of the computations is accepted, accept
```

Thus, this recognizer will either accept, all go on looping for increasing i.

(c) Intuition: To prove undecidability, the most straightforward way is to build a Turing Machine that can cause a contradiction using recursion theorem.

Assume there is a Turing Machine M that decides L_{Trump}. We can build a Turing Machine F in the following way:

```
$F$ = "On input $x$:"
• Obtain, via the recursion theorem, own description $< F >$
• Run $M$ on input $< F >$
• if $M$ accepts:
          loop
  if $M$ rejects:
          erase all $ symbol, write a wall symbol #, and accept
```

Hence, there are two cases:

Case 1: When M accepts on input $< F >$. The TM F we constructed will loop for all input $x \in \Sigma^*$, which is a contradiction.

Case 2: When M rejects on input $< F >$. The TM F we constructed will replace all \$ symbol with a single # symbol and halt by accepting for all input $x \in \Sigma^*$ (including all the x that contains at least 5.7 billion \$'s). This is a contradiction, as in this case, F is a Trump Machine.

Hence, the construction of F proves the undecidability of L_{Trump}

(d) Intuition: We can construct a valid Trump Machine T, and assume there exists a recognizer R that recognizes L_{Trump}. We then construct a Trump Machine M that will induce a contradiction.

```
$T$ = "On input $x$:"
• Scan from the start, erase all $ symbols, write a wall symbol # at the end
• accept
```

From the construction, we can see that for all input $x \in \Sigma^*$ (including those with at least 5.7billion \$), T replaces all \$ with a single # and then halts. Hence, T is a Trump Machine by definition, and T always halts.

```
$F$ = "On input $x$:"
• Obtain, via Recursion Theorem, own description $< F >$
• Run $R$ on input $< M >$ and $T$ on input $x$ in parallel
• if $R$ returns first:
          if $R$ accepts
                    loop forever
          if $R$ rejects
                    wait for $T$ to halt. Return $T(x)$.
```

- if T returns first:

  Return $T(x)$

There are two cases:

Case 1: R finishes first. If R accepts, F never halts, which contradicts the definition of Trump Machine. If R rejects, F waits T to finish. Since T is a valid trump machine, F does all the actions required by a Trump Machine and then halts, which contradicts to the result of R.

Case 2: T finishes first, F behaves as if it is a valid Trump Machine. There are again two subcases: If R is looping, it is a contradiction since R should accept F as a valid Trump Machine. If R is not looping, it will eventually halt. Since x can be any string $\in \Sigma^*$, when x is long enough, T will finish later then R. This will lead to contradiction as stated in Case 1.

Hence, we have proved the unrecognisability of L_{Trump} by contradiction.

Problem 2

(a) Intuition: All languages contain at most ∞ strings

We can build a Turing machine M:

```
M= "On input $x$, where $x \in \Sigma^*$:"
    Halt and output "INCONCLUSIVE"
```

Since this Turing Machine can be applied to any language, every language is ∞-Inconclusive-Decidable.

(b) Intuition: we can prove this by constructing 2 Turing Machine F_1 and F_2 that prove the 1-Inconclusive-Undecidability of L_{HALT_ϵ}. The descriptions $< F_1 >$ and $< F_2 >$ are two inputs that must be marked as "INCONCLUSIVE" by the decider.

Assume there is a Turing Machine M that 1-Inconclusive-Decides L_{HALT_ϵ}. We can build 2 Turing Machine F_1 and F_2 in the following way:

```
$F_1$ = "On input $w$:"
  • Obtain, via the recursion theorem, own description $< F_1 >$
  • Run $M$ on input $< F_1 >$
  • if $M$ accepts:
        loop forever
    if $M$ rejects:
        accept $w$
    if $M$ says "INCONCLUSIVE",
        accept $w$
```

```
$F_2$ = "On input $w$:"
  • Obtain, via the recursion theorem, own description $< F_2 >$
  • Run $M$ on input $< F_2 >$
  • if $M$ accepts:
        loop forever
    if $M$ rejects:
        reject $w$
    if $M$ says "INCONCLUSIVE",
        reject $w$
```

Hence, there are 3 cases:

Case 1: when M accepts, both machines loop forever for any input w including ϵ which is a contradiction .

Case 2: when M rejects, both machines halt for any input w including ϵ which is a contradiction .

Case 3: Hence, in order not to contradict, M can only output "inconclusive" for both descriptions. At this time, $LF_1 = \Sigma*$ while $L_{F_2} = \varnothing$. Hence, $< F_1 > \neq < F_2 >$. Hence, L_{HALT_ϵ} is not 1-Inconclusive-Decidable, which is a contradiction.

Hence, we proved the 1-Inconclusive-Undecidability of L_{HALT_ϵ}.

(c) Intuition: Use the method we used for the first problem, we can construct a third TM F_3 that induces contradiction, but is different from F_1 and F_2.

Assume there is a Turing Machine M that 2-Inconclusive-Decides L_{HALT_ϵ}. We can construct F_1 and F_2 in exactly the same say as we did above. We can construct F_3 in the following way:

$F_3 =$ "On input w:"

```
• Obtain, via the recursion theorem, own description $< F_3 >$
• Run $M$ on input $< F_3 >$
• if $F$ accepts:
      loop forever
  if $M$ rejects:
      reject $w$
  if $M$ says "INCONCLUSIVE",
      accept $w$ if $|w| > 1$.
```

Hence, there are 3 cases, and the above first 2 apply:

Case 3: In order not to contradict, M can only output "inconclusive" for $< F_3 >$. However, since $L(F_3) \neq \varnothing$ and $L(F_3) \neq \Sigma*$, the three TM are different in descriptions. Hence, L_{HALT_ϵ} is not 2-Inconclusive-Decidable, which is a contradiction.

Problem 3

(a) Intuition: To show that L_{loop} is computable by perpetual machines, we can construct a perpetual machine by adding an extra non-halting accept state q^*, and make the machine enters it infinitely many times if it loops.

We construct P such that it resembles M, the only difference is P has an extra non-halting accept state q^*. Before each state transition in M from q_i to q_j, P must enter q^* before entering q_j. For the accept and reject states of M, P just keep looping in the same state.

$P =$ "On input $< M >$, where M is a TM:"

- Simulate M on input ϵ, before each state transition, P goes into q^* before entering the next state.
- if enter the accept/reject states of M, keep looping in the same state.

Thus, if M loops on input ϵ, M has infinitely state transitions (including transitions into the same state). By our construction, this will lead P to enter q^* infinitely many times. If M halts on input ϵ, M has finitely state transitions (including transitions into the same state). This will lead P to enter q^* finitely many times, then P will loop in the accept/reject states. Hence, we proved P computes L_{loop}.

(b) Intuition: Since Perpetual machine, like TM, have a description of finite length. Also, we learnt that we can enumerate of the strings $\in \Sigma^*$. We can enumerate all the perpetual machines in the order of the enumerations of strings, and use diagonalization to induce a contradiction.

Let $\mathcal{B}$ denote the given enumeration of all strings $\in \Sigma^*$. For each element in $\mathcal{B}$, if it is accepted by a perpetual machine P (go into q^* infinitely many times), we record as 1, otherwise we record as 0. Thus, we can get an infinitely long characteristic sequence for each of the perpetual machine. Then we used Cantor's Diagonalization to produce a contradiction by enumerating all the perpetual machines as stated in the Intuition, and construct a language L with a characteristic sequence that has ith entry opposite to the ith entry of P_i's characteristic sequence.

The construction is as follows, $s_1, s_2, ...$ is an enumeration of all strings $\in \Sigma^*$, denoted by $\mathcal{B}$, and $P_1, P_2, ...$ is an enumeration of all descriptions of perpetual machines following the order in $\mathcal{B}$. (with only strings that happen to be descriptions of perpetual machines listed):

$$\begin{array}{cccc} & s_1 & s_2 & s_3\ ... \\ P_1 & \mathbf{1} & 0 & 1... \\ P_2 & 1 & \mathbf{0} & 0\ ... \\ P_3 & 0 & 1 & \mathbf{1}\ ... \\ ... & ... & ... & ... \end{array}$$

According to our construction L has characteristic sequence $\overline{101...} = 010....$ Assume L is computable by a perpetual machine, all the strings $\in L$ must be accepted by a perpetual machine (go into q^* infinitely many times). Thus, there exists a P_l that has a characteristic sequence we same as L. However, according to our construction, such P_l does not exist because it is not the same (has different characteristic sequence from) as any perpetual machine p in the list because for any p_i, p_l and p_i differs in the ith position. Hence, we have proved $L \in \Sigma^*$ is not computable by perpetual machines.

(c) Intuition: we have learnt in class that the set of all strings in Σ^* is countable. Let $\mathcal{B}$ be a given enumeration of strings in Σ^* as shown in Q3, HW4. We then use the same method we used in 1(b), and similar construction that we used in part (a) to prove.

We construct P such that it resembles M, the only difference is P has an extra non-halting accept state q^*. Before each state transition in M from q_i to q_j, P must enter q^* before entering q_j. For the accept and reject states of M, P just keep looping in the same state.

$P =$ "On input $< M >$, where M is a TM:"

- Repeat the following for $i = 1, 2, 3, ...$
- Run for $2i$ steps for the first i strings in the enumeration $\mathcal{B}$
- if enter the accept states of M, keep looping in the same state without proceed to the next string

Therefore, if $< M > \in L_{empty}$, P will never enter the accept states of M, and M has infinitely state transitions (including transitions into the same state) for any string $\in \Sigma^*$. This will lead P to enter q^* infinitely many times (due to infinitely increasing i). if $< M > \notin L_{empty}$, say M accept a string s, this will lead F to run only finite many steps before entering the accept state. After entering the accept state, F never enters q^* again. Thus, this will lead P to enter q^* finitely many times. Hence, we proved P computes L_{empty}.

(d) Intuition: Again we let $\mathcal{B}$ be a given enumeration of strings in Σ^* as shown in Q3, HW4. If $L(M_1) \neq L(M_2)$, there must be a first string that is accepted by one that is not accepted by the other. Since the string is accepted by one, it only runs a finite many steps on one of the machines. We can make use of that. We use a similar method we used in the previous part.

We construct P such that it has all the states in M_1 and M_2, and an extra non-halting accept state q^*. Before each state transition in M_1/M_2 from q_i to q_j, P must enter q^* before entering q_j. For the accept and reject states of M, P just keep looping in the same state.

$P =$ "On input $< (M_1, M_2) >$, where M_1/M_2 are TM:"

- Repeat the following for $i = 1, 2, 3, ...$
- Run for $2i$ steps for the first i strings in the enumeration $\mathcal{B}$ for both M_1/M_2 part.
- if enter the accept states of M_1/M_2, Simulate the string on the other machine(The actual M_1/M_2 without q^*) until going into an accept state. If go into accept state, continue with the next string and increasing i. Otherwise loop forever or loop in the reject state.

If a string is accept by a machine with i step of simulation. Before this event, we only go into q^* finitely many times. Then we simulate the other machine with the string until we go into an accept state. Hence, there are 2 cases:

Case 1: If the string is accept by the other string, then we continue with the second step with increasing i. If it is always the case. The execution goes on forever and we go into q^* infinitely many times, which means $(< M_1 >, < M_2 >) \in L_{EQ}$.

Case 2: If the string is rejected/loops forever. We loop in the reject state/continue looping, and thus the number of times of entering q^* will never increase again. Thus, we only enter q^* finitely many times (the number of times before the string is accepted in i steps by the other machine). This means $(< M_1 >, < M_2 >) \notin L_{EQ}$.

Hence, we constructed P as desired, which shows L_{EQ} is computable by perpetual machines.

Problem 4 Intuition: the layout of the game reminds me of tape of TM. Also, the unique Ace Deck in the front reminds me of the head of tape. The position of the Ace deck can be used to indicate the current position of the head. The rest of the deck can be used to indicate the current state of a TM. For the normal deck, we can use it to indicate the alphabets of TM. Hence, with all the above ideas, we are able to simulate the transition function function of a TM δ by only allowing shuffling of $3N$ cards. We will use mapping reducibility ideas to reduce the problem to the undecidable L_{HALT_ϵ} defined in Problem 2.

Since in the problem, there is a statement saying we can assume "that every TM can be converted into an equivalent TM that will never try to move left on the leftmost tape position, and if this new TM halts, it always halts when the head is at the leftmost tape position.", we can assume the input M is already such a TM for clarity.

Assume the input $< M >$ where M is a TM description input for L_{HALT_ϵ}, assume it has p states and q different alphabets. We can choose the number of N so that $N! > q$ and $(N-1)! > p$. Thus, each state can have a unique representation by cards 2 to N in the Ace Deck, and each alphabet can have a unique representation by a Normal Deck.

Assume the input $< M >$ has a transition function: $\delta(q,a) = (r, b, \{L, R\})$, where $a, b \in \Gamma$, $q, r \in Q$, L, R indicates the direction of the head move. We can simulate the transition function by only including the following shufflings. However, before go into details, I want to talk about how to use always make sure we can do shufflings of 3 decks without mixing up of cards originally from different decks. We have to make sure this because if mixed up, for example for a Normal deck, we cannot interpret its content as an alphabet of M. The idea is the Ace deck's position can be indicated by the leading A. Thus, the A and the following $A-1$ cards are from the original Ace deck. The Normal Deck just on the left of the Ace deck represents the tape position that the head is pointing to and the Normal Deck just on the right of the Ace deck represents the tape position that is just to the right of the current head position. The shuffling rules corresponding to the TM M will only include shuffling of $3N$ cards. Each left element of the tuple have the $(N+1)_{th}$ element is A, or have the $(2N+1)_{th}$ element as A, indicating we are trying to do a R/L movement of head. The right element of the tuple, will only have A as the $(N+1)_{th}$ element, or $(2N+1)_{th}$ element, indicating result of a L/R movement of head. Also, for both left/right element of the tuple, it has exactly $3N$ cards, and each N cards must be exactly a Normal or Ace deck. The only exceptions are when we read in the ϵ which has never been processed before (the TM M starts from a blank tape)/move the head to a tape position with a blank symbol, which is described below. A formal definition of the transition function δ is as follows:

δ's representation by shuffling rule=

- $\delta(q,a) \to (r,b,L) = ((\overline{NNA}), (\overline{NAN}))$, where A is an Ace deck that starts with an A. This allows us to always make sure that the only A is at $(kN+1)th, k \in \mathbf{N}$ position among all the cards, thus allowing us to locate each deck of cards with N cards inside, and each deck contains exactly all the cards of a Normal/Ace deck. The N in the rule represents a full Normal deck in any order that is a representation of a alphabet. Thus, we never mix up cards from 2 different decks.
- $\delta(q,a) \to (r,b,R) = ((\overline{NAN}), (\overline{NNA}))$, with the same representation as stated above. Also, when the Ace Deck is the last deck, add a Normal deck in the end so that we are still possible to move the head further to right.
- $((\overline{Ann}), (\overline{\epsilon An}))$: **The symbol n represents a Normal deck with original arrangement, which represents the blank symbol $\sqcup$ in our construction.** When we read the input alphabet ϵ we first add a new Normal deck at the end before we do the shuffling.
- $\delta(q,a) = (q_{halt}, b, L)$: This is always a left movement since the TM will only halt when the head is at the leftmost tape position and will never go beyond the leftmost position of the tape. For this case, all the

alphabet b that can satisfy this transition, we represent it by a Normal deck arrangement starting with N.

Hence, we have given and explained how the shuffling rules with $3N$ cards are constructed. Now we go on to map the rest of the TM to the card shuffling game.

The game can only start with rule 3 since the Ace deck is originally the first deck, which we read in the ϵ symbol by adding a new Normal deck and shuffle the first three decks. Then we follow the transitions function of the Turing machine. If we can win the card game, that means we the Ace deck is the second deck (pointing to the leftmost position), and the first deck is representing the b in rule 4, which starts with N. Hence, we have constructed a card game to simulate L_{HALT_ϵ}. If we can decide if the card game is winable $\Rightarrow$ we can decide L_{HALT_ϵ}. However, this is a contradiction because we know it is undecidable, or we cannot prove in Problem 2 that it is not 1-Inconclusive Decidable.