Name: Erynn-Marie Phan

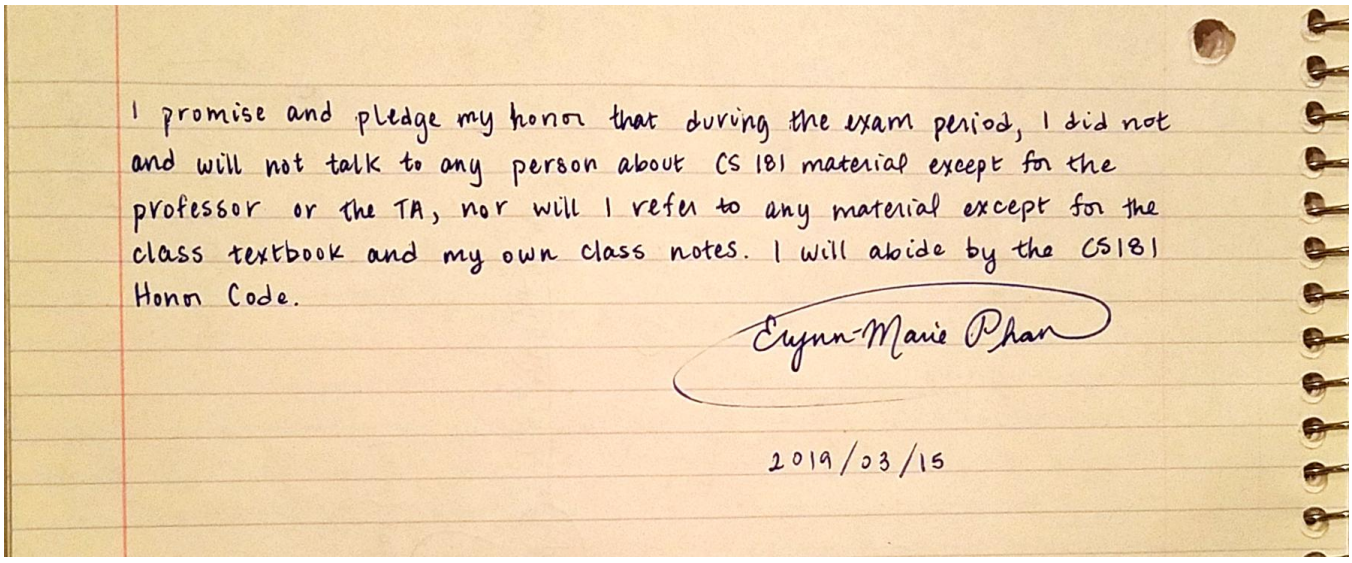
Student ID: 504 757 459

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.



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

Name: Erynn-Marie Phan

Student ID: 504 757 459

1. **Decidability/Recognizability (60 points)**

Height
For a Turing machine M, let $\mathsf{height}(M)$ denote the length of the shortest string accepted by M (or ∞ if M does not accept any string). Consider the following language

$$L_{\mathsf{height}} = \{\langle M\rangle \mid \mathsf{height}(M) < |\langle M\rangle|\},$$

that is the language of Turing machines that accept a string that is shorter than their own description.

a) (15 points) Show that L_{height} is undecidable.

Intuition:
If a decider D exists, we can construct a machine M that runs D on its own description. If $D(\langle M\rangle)$ accepts, M rejects all x. If $D(\langle M\rangle)$ rejects, M accepts all x. This M cannot exist. Therefore D cannot exist.

Formal Solution:
Suppose there exists a decider D for L_{height}. Construct a machine M that runs the following algorithm:

- On input x
- By the recursion theorem, let $y = \langle M\rangle$
- run $D(y)$
  - If it accepts, reject.
  - If it rejects, accept.

Analysis of M:

- Case 1: $D(\langle M\rangle)$ accepts
  $\langle M\rangle \in L(D)$ implies that $\mathsf{height}(M) < |\langle M\rangle|$
  However by construction, $L(M) = \emptyset$, $\mathsf{height}(M) = \infty$
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$
- Case 2: $D(\langle M\rangle)$ rejects
  $\langle M\rangle \notin L(D)$ implies that $\mathsf{height}(M) \geq |\langle M\rangle|$
  However by construction, $L(M) = \Sigma^*$
  Since $\varepsilon \in L(M)$ and $\langle M\rangle$ is not the empty string, $\mathsf{height}(M) = 0 < |\langle M\rangle|$
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$

Since there is a contradiction in every case, neither M nor D can exist.

Name: Erynn-Marie Phan

Student ID: 504 757 459

b) (15 points) Show that L_{height} is recognizable.

Intuition:
The recognizer R can run $M(s)$ for all $s \in \Sigma^*$ such that $|s| < |\langle M \rangle|$ using diagonal scheduling. (Although there are a finite number of strings, diagonal scheduling is necessary in case $M(s)$ loops for some s). If M ever accepts, R should accept.

Formal Solution:
Construct a machine R that runs the following algorithm:

- On input $\langle M \rangle$
- If the input is not a description of a Turing machine, reject.
- Let $X = \{x_1, x_2, \dots\}$ be a list of all strings $x_i \in \Sigma^*$ such that $|x_i| < |\langle M \rangle|$
- Run $M(x_i)$ for all $x_i \in S$ using diagonal scheduling.
  - If any computation $M(x_i)$ accepts, accept.
- Reject.

Analysis of R:

- Case 1: $\langle M \rangle \in L_{\mathsf{height}}$
  There exists an $x \in L(M)$ such that $|x| < |\langle M \rangle|$
  $x \in X$
  (From lecture) Using diagonal scheduling, R will eventually run $M(x)$, and it will eventually accept.
  By construction, R also accepts.
  $\langle M \rangle \in L(R)$
- Case 2: $\langle M \rangle \notin L_{\mathsf{height}}$
  For all $x_i \in X$, $M(x_i)$ either loops or rejects.
  By construction, R either loops or rejects.
  $\langle M \rangle \notin L(R)$

Therefore R recognizes L_{height}.

Name: Erynn-Marie Phan

Student ID: 504 757 459

Trump Machine
A Turing machine M over alphabet $\Sigma = \{0, 1, \$\}$ and tape alphabet $\Gamma = \{\sqcup, 0, 1, \$, \sharp\}$ is a Trump machine if for all $x \in \Sigma^*$ such that x contains at least 5.7 billion \$ symbols, during the execution of M on input x, M erases all \$ symbols from its tape (replaces them with blanks), writes a single wall symbol $\sharp$, and then shuts down (either accepts or rejects).

Let

$$L_{\mathsf{Trump}} = \{\langle M \rangle \mid M \text{ is a Trump machine}\}.$$

c) (15 points) Show that L_{Trump} is undecidable.

Intuition:
If a decider D exists, we can construct a machine M that runs D on its own description. If $D(\langle M \rangle)$ accepts, M loops. Then M can't be a Trump machine. If $D(\langle M \rangle)$ rejects, M can satisfy all the necessary conditions to be a Trump machine. This M cannot exist. Therefore D cannot exist.

Formal Solution:
Suppose there exists a decider D for L_{Trump}. Construct a machine M that runs the following algorithm:

- On input x
- By the recursion theorem, let $y = \langle M \rangle$
- Run $D(y)$
  - If it accepts, loop.
  - If it rejects
    - Overwrite the \$ symbols with blanks.
    - Write a wall symbol.
    - Accept.

Analysis of M:

- Case 1: $D(\langle M \rangle)$ accepts
  $\langle M \rangle \in L(D)$ implies that M is a Trump machine.
  However by construction, M loops on every input. Therefore M is not a Trump machine.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$
- Case 2: $D(\langle M \rangle)$ rejects
  $\langle M \rangle \notin L(D)$ implies that M is not a Trump machine.
  However by construction, for all x, M overwrites all the \$ symbols with blanks, writes a single wall symbol, and accepts. Therefore M is a Trump machine.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$

Since there is a contradiction in every case, neither M nor D can exist.

Name: Erynn-Marie Phan

Student ID: 504 757 459

d) (15 points) Show that L_{Trump} is unrecognizable.

Intuition:
If a recognizer R exists, we can construct a machine M that on input x, runs R on its own description for $|x|$ steps. If $R(\langle M \rangle)$ accepts, M loops. If $R(\langle M \rangle)$ rejects or doesn't finish in $|x|$ steps, then M can satisfy all the necessary conditions to be a Trump machine. This M cannot exist. Therefore R cannot exist.

Formal Solution:
Suppose there exists a recognizer R for L_{Trump}. Construct a machine M that runs the following algorithm:

- On input x
- By the recursion theorem, let $y = \langle M \rangle$
- Run $R(y)$ for $|x|$ steps.
  - If it accepts, loop.
  - If it rejects and or doesn't finish
    - Overwrite the \$ symbols with blanks.
    - Write a wall symbol.
    - Accept.

Analysis of M:

- Case 1: $R(\langle M \rangle)$ accepts in t steps.
  $\langle M \rangle \in L(R)$ implies that M is a Trump machine.
  However by construction, M loops on $\$^{t+1}$. Therefore M is not a Trump machine.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$
- Case 2: $R(\langle M \rangle)$ rejects or loops.
  $\langle M \rangle \notin L(R)$ implies that M is not a Trump machine.
  However by construction, for all x, M overwrites all the \$ symbols with blanks, writes a single wall symbol, and accepts. Therefore M is a Trump machine.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$

Since there is a contradiction in every case, neither M nor R can exist.

Name: Erynn-Marie Phan

Student ID: 504 757 459

2. **Inconclusive Decider (40 points)**
An Inconclusive Decider is like a decider except that on some set of inputs, instead of accepting or rejecting, it may instead halt and output "INCONCLUSIVE".

Formally, we say that a language L is k-Inconclusive-Decidable for some $k \in \mathbb{N} \cup \{\infty\}$ if there exists a Turing machine M such that

- For all x such that $x \in L$, M either halts and outputs "ACCEPT" or halts and outputs "INCONCLUSIVE".
- For all x such that $x \notin L$, M either halts and outputs "REJECT" or halts and outputs "INCONCLUSIVE".
- M halts and outputs "INCONCLUSIVE" on at most k distinct inputs x.

(Note that L is decidable if and only if it is 0-Inconclusive-Decidable)

a) **(5 points)** Show that all languages are ∞-Inconclusive-Decidable.

Intuition:
A machine that halts and outputs "INCONCLUSIVE" for every x is an Inconclusive Decider for every language.

Formal Solution:
Consider the machine M which runs the following algorithm:

- On input x
- Halt and output "INCONCLUSIVE"

Consider the arbitrary language L.
For all $x \in L$, M halts and outputs "INCONCLUSIVE."
For all $x \notin L$, M halts and outputs "INCONCLUSIVE."
M halts and outputs "INCONCLUSIVE" for an infinite number of distinct inputs x.
Therefore M is an Inconclusive Decider for L, and L is ∞-Inconclusive-Decidable.

Name: Erynn-Marie Phan

Student ID: 504 757 459

b) **(20 points)** Show that $L_{\mathsf{Halt}_\varepsilon} = \{\langle M \rangle \mid M(\varepsilon) \text{ halts}\}$ is **not** 1-Inconclusive-Decidable.

Intuition:
If an Inconclusive Decider D exists, we can construct two machines M_1 and M_2. Both can run D on their own descriptions and contradict the accepting and rejecting cases in the normal way. If $D(M_1)$ outputs "INCONCLUSIVE", M_1 loops. If $D(M_2)$ outputs "INCONCLUSIVE", M_2 halts. Then the decider would be wrong if it output accept or reject for either machine, but it cannot output "INCONCLUSIVE" for both, since they are not the same machine.

Formal Solution:
Suppose $L_{\mathsf{Halt}_\varepsilon}$ is 1-Inconclusive-Decidable. Then there exists a Turing machine D such that

- For all $x \in L_{\mathsf{Halt}_\varepsilon}$, D either halts and outputs "ACCEPT" or halts and outputs "INCONCLUSIVE."
- For all $x \notin L_{\mathsf{Halt}_\varepsilon}$, D either halts and outputs "REJECT" or halts and outputs "INCONCLUSIVE."
- D halts and outputs "INCONCLUSIVE on at most 1 distinct input x.

Construct a machine M_1 which runs the following algorithm:

- On input x
- By the recursion theorem, let $y = \langle M_1 \rangle$
- Run $D(y)$
  - If it accepts, loop.
  - If it rejects, halt and accept.
  - If it outputs "INCONCLUSIVE," loop.

Construct another machine M_2 which runs the following algorithm:

- On input x
- By the recursion theorem, let $y = \langle M_2 \rangle$
- Run $D(y)$
  - If it accepts, loop.
  - If it rejects, halt and accept.
  - If it outputs "INCONCLUSIVE," halt and accept.

Analysis of M_1 and M_2:

- Case 1: $D(\langle M_1 \rangle)$ accepts OR $D(\langle M_2 \rangle)$ accepts
  D accepts at least one of the machines. Let one of the accepted machines be M.
  $\langle M \rangle \in L(D)$ implies that $M(\varepsilon)$ halts.
  However, by construction, $M(\varepsilon)$ loops.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$
- Case 2: $D(\langle M_1 \rangle)$ rejects OR $D(\langle M_2 \rangle)$ rejects
  D rejects at least one of the machines. Let one of the rejected machines be M.
  $\langle M \rangle \notin L(D)$ implies that $M(\varepsilon)$ loops.
  However, by construction, $M(\varepsilon)$ halts and accepts.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$
- Case 3: $D(\langle M_1 \rangle)$ outputs "INCONCLUSIVE" AND $D(\langle M_2 \rangle)$ outputs "INCONCLUSIVE"
  D outputs "INCONCLUSIVE" for both machines.
  Then D outputs "INCONCLUSIVE" for at least 2 distinct inputs.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$

Since there is a contradiction in every case, M_1, M_2, and D cannot exist.

Name: Erynn-Marie Phan

Student ID: 504 757 459

c) **(15 points)** Show that $L_{\mathsf{Halt}_\varepsilon} = \{\langle M \rangle \mid M(\varepsilon) \text{ halts}\}$ is **not** 2-Inconclusive-Decidable.

Intuition:

If an Inconclusive Decider D exists, we can construct three machines M_1, M_2, and M_3, which run D on their own descriptions and contradict the accepting and rejecting cases in the normal way. If $D(M_1)$ outputs "INCONCLUSIVE", M_1 loops. If $D(M_2)$ outputs "INCONCLUSIVE", M_2 accepts. If $D(M_3)$ outputs "INCONCLUSIVE", M_3 rejects. Then the decider would be wrong if it output accept or reject for any of these machines, but it cannot output "INCONCLUSIVE" for all three, since they are not the same machine.

Formal Solution:

Suppose $L_{\mathsf{Halt}_\varepsilon}$ is 2-Inconclusive-Decidable. Then there exists a Turing machine D such that

- For all $x \in L_{\mathsf{Halt}_\varepsilon}$, D either halts and outputs "ACCEPT" or halts and outputs "INCONCLUSIVE."
- For all $x \notin L_{\mathsf{Halt}_\varepsilon}$, D either halts and outputs "REJECT" or halts and outputs "INCONCLUSIVE."
- D halts and outputs "INCONCLUSIVE on at most 2 distinct input x.

Construct two machines M_1 and M_2 identical to the ones in part (b).
Construct a machine M_3 that runs the following algorithm:

- On input x
- By the recursion theorem, let $y = \langle M_2 \rangle$
- Run $D(y)$
  - If it accepts, loop.
  - If it rejects, halt and accept.
  - If it outputs "INCONCLUSIVE," halt and reject.

Analysis of M_1, M_2, and M_3:

- Case 1: D accepts at least one of the machines.
  Let one of the accepted machines be M.
  $\langle M \rangle \in L(D)$ implies that $M(\varepsilon)$ halts.
  However, by construction, $M(\varepsilon)$ loops.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$
- Case 2: D rejects at least one of the machines.
  Let one of the rejected machines be M.
  $\langle M \rangle \notin L(D)$ implies that $M(\varepsilon)$ loops.
  However, by construction, $M(\varepsilon)$ halts and accepts.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$
- Case 3: D outputs "INCONCLUSIVE" for all 3 machines
  Then D outputs "INCONCLUSIVE" for at least 3 distinct inputs.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$

Since there is a contradiction in every case, M_1, M_2, M_3, and D cannot exist.

Name: Erynn-Marie Phan

Student ID: 504 757 459

3. **Perpetual Machines (70 points)**
A perpetual machine P is like a Turing machine, except that it never halts on any input. Instead of containing accept and reject states, q_{accept} and q_{reject}, it contains a non-halting accepting state q^*. When P enters q^*, instead of terminating its computation, it continues executing and can transition out of this state.

A perpetual machine P computes a language L if

- For all $x \in L$, $P(x)$ enters q^* an infinite number of times
- For all $x \notin L$, $P(x)$ does not enter q^* an infinite number of times

If there does not exist any perpetual machine P that computes a language L, we say that L is uncomputable by perpetual machines.

a) (25 points) Show that the language

$$L_{\mathsf{loop}} = \{\langle M \rangle \mid M \text{ is a Turing machine and } M(\varepsilon) \text{ loops}\}$$

is computable by perpetual machines.

Intuition:
We can construct a machine P that on input $\langle M \rangle$ simulates $M(\varepsilon)$. After each step of $M(\varepsilon)$, P enters and exits q^*. Then P enters q^* an infinite number of times if and only if $M(\varepsilon)$ loops.

Formal Solution:
Construct a perpetual machine P. P has a special state q^* which it does not enter unless the algorithm explicitly says that it does. P runs the following algorithm:

- On input $\langle M \rangle$
- While (M is a Turing machine and $M(\varepsilon)$ has not halted):
  - Simulate one step of $M(\varepsilon)$
  - Enter and exit q^*
- While (true):
  - Go to any state q such that $q \neq q^*$

Analysis of P:

- Case 1: $\langle M \rangle \in L_{\mathsf{loop}}$
  Then M is a Turing machine and $M(\varepsilon)$ loops.
  Then $P(\langle M \rangle)$ never exits the first While loop.
  Then $P(\langle M \rangle)$ enters q^* an infinite number of times.
- Case 2: $\langle M \rangle \notin L_{\mathsf{loop}}$
  Then one of the following cases is true.
  - M is not a Turing machine.
    Then $P(\langle M \rangle)$ never enters the first While loop.
    Then $P(\langle M \rangle)$ enters q^* 0 times.
  - $M(\varepsilon)$ halts in t steps.
    Then $P(\langle M \rangle)$ exits the first While loop after t steps.
    Then $P(\langle M \rangle)$ enters q^* exactly t times.

  Therefore $P(\langle M \rangle)$ does not enter q^* an infinite number of times.

Therefore P computes L_{loop}.

Name: Erynn-Marie Phan

Student ID: 504 757 459

b) (20 points) Construct a language $L \subseteq \Sigma^*$ that is uncomputable by perpetual machines and prove that this is the case. *(Hint: Use diagonalization)*

Intuition:
Perpetual machines are countably infinite because each one can be represented by a string. Therefore there is an enumeration. Using an enumeration for strings and an enumeration for languages of perpetual machines, we can make a similar argument that we made in class to show that there exists a language which is not computed by any perpetual machine in the enumeration.

Formal Solution:
Perpetual machines are countably infinite because each one can be represented by a string. Since each perpetual machine computes 1 language, the languages computed by perpetual machines are also countably infinite.
Let $\{s_1, s_2, s_3, \dots\}$ be an enumeration of all strings in Σ^*.
Let $\{L_1, L_2, L_3, \dots\}$ be an enumeration of all languages of perpetual machines. $L_i \subseteq \Sigma^*$ for all i.
Let there be a boolean value $L_i(s_j)$ for a language $L_i \subseteq \Sigma^*$ and a string $s_j \in \Sigma^*$ defined as follows:

$$L_i(s_j) = \begin{cases} True & s_j \in L_i \\ False & s_j \notin L_i \end{cases}$$

Construct the language L, which has the property that

$$L(s_i) = \neg L_i(s_i)\ \forall i$$

Claim: L is not computable by perpetual machines.
Proof:

- Suppose L is computable by perpetual machines. Then it is in the enumeration $\{L_1, L_2, L_3, \dots\}$.
- Therefore $L = L_i$ for some $i \in \mathbb{N}$.
- Case 1: $s_i \in L_i$
  Then $L_i(s_i) = True$.
  By definition of L, $L(s_i) = \neg L_i(s_i) = False$.
  Then $s_i \notin L$.
  Then $L \neq L_i$.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$
- Case 2: $s_i \notin L_i$
  Then $L_i(s_i) = False$.
  By definition of L, $L(s_i) = \neg L_i(s_i) = True$.
  Then $s_i \in L$.
  Then $L \neq L_i$.
  $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$

Because there is a contradiction in every case, $L \neq L_i$ for any $i \in \mathbb{N}$.

Because L is not in the enumeration of languages computed by perpetual machines, it is not computed by any perpetual machine.

Name: Erynn-Marie Phan

Student ID: 504 757 459

c) (25 points) Show that the language

$$L_{\mathsf{Empty}} = \{\langle M \rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset\}$$

is computable by perpetual machines.

Intuition:
We can construct a machine N that on input $\langle M \rangle$ simulates M on all input strings using diagonal scheduling. After each step of M, N enters and exits q^*. If M accepts a string, N never enters q^* again. Then N enters q^* an infinite number of times if and only if $L(M) = \emptyset$.

Formal Solution:
Construct a perpetual machine P. P has a special state q^* which it does not enter unless the algorithm explicitly says that it does. P runs the following algorithm:

- On input $\langle M \rangle$
- If M is a Turing machine
  - For all strings $s \in \Sigma^*$, using diagonal scheduling
    - Simulate one step of $M(s)$
    - Enter and exit q^*
    - If $M(s)$ accepts, break.
- While (true):
  - Go to any state q such that $q \neq q^*$

Analysis of P:

- Case 1: $\langle M \rangle \in L_{\mathsf{Empty}}$
  Then M is a Turing machine and $L(M) = \emptyset$.
  Then $P(\langle M \rangle)$ never exits the for loop.
  Then $P(\langle M \rangle)$ enters q^* an infinite number of times.
- Case 2: $\langle M \rangle \notin L_{\mathsf{Empty}}$
  Then one of the following cases is true.
  - M is not a Turing machine.
    Then $P(\langle M \rangle)$ never enters the for loop.
    Then $P(\langle M \rangle)$ enters q^* 0 times.
  - $M(s)$ accepts for some $s \in \Sigma^*$.
    Then $P(\langle M \rangle)$ exits the for loop.
    Then $P(\langle M \rangle)$ enters q^* a finite number of times.

  In each of these sub-cases, $P(\langle M \rangle)$ does not enter q^* an infinite number of times.

Therefore P computes L_{Empty}.

Name: Erynn-Marie Phan
Student ID: 504 757 459

d) (Extra Credit, 40 points) Show that the language

$$L_{\mathsf{EQ}} = \{(\langle M_1 \rangle, \langle M_2 \rangle) \mid M_1 \text{ and } M_2 \text{ are Turing machines and } L(M_1) = L(M_2)\}$$

is computable by perpetual machines.

Intuition:
We can construct a machine P that on input $(\langle M_1 \rangle, \langle M_2 \rangle)$ simulates M_1 and M_2 on all input strings using diagonal scheduling. After a step of M_1 and M_2, P enters and exits q^*. If $M_1(s)$ accepts, P enters a state which it cannot leave unless $M_2(s)$ accepts. Similarly if $M_2(s)$ accepts, P should enter another state and not leave until $M_1(s)$ accepts. If either machine accepts a string that the other has already rejected, or if either machine rejects a string which the other has already accepted, P enters a rejecting state and remains there forever. Then P enters q^* an infinite number of times if and only if $L(M_1) = L(M_2)$.

Formal Solution:
Construct a perpetual machine P. P has a special state q^* which it does not enter unless the algorithm explicitly says that it does. P runs the following algorithm:
- On input $(\langle M_1 \rangle, \langle M_2 \rangle)$
- If M_1 and M_2 are both Turing machines, state = continue.
- Else, state = reject.
- Loop: {

- case state == continue
  - For all strings $s \in \Sigma^*$, using diagonal scheduling (continue from where you left off)
    - Case: $M_1(s)$ and $M_2(s)$ have both halted. Continue to the next string.
    - Case: $M_1(s)$ has rejected and $M_2(s)$ has not halted
      - Simulate 1 step of $M_2(s)$
      - If $M_2(s)$ accepts
        - state = reject
        - break
      - Else, enter and exit q^* once
    - Case: $M_1(s)$ has not halted and $M_2(s)$ has rejected
      - Simulate 1 step of $M_1(s)$
      - If $M_1(s)$ accepts
        - state = reject
        - break
      - Else, enter and exit q^* once
    - Case: both $M_1(s)$ and $M_2(s)$ are running
      - Simulate 1 step of $M_1(s)$ and $M_2(s)$
      - If $M_1(s)$ and $M_2(s)$ do the same thing, enter and exit q^* once
      - Else if 1 machine rejects and the other continues, enter and exit q^* once
      - Else if $M_1(s)$ accepts and $M_2(s)$ continues
        - state = 1A
        - $s_1 = s$
        - break
      - Else if $M_2(s)$ accepts and $M_1(s)$ continues
        - state = 2A
        - $s_1 = s$
        - break

- case state == 1A
  - Continue simulating $M_2(s_1)$
  - If it accepts, state = continue
  - If it rejects, state = reject
- case state == 2A
  - Continue simulating $M_1(s_1)$
  - If it accepts, state = continue
  - If it rejects, state = reject
- case state == reject
  - Do nothing.

}

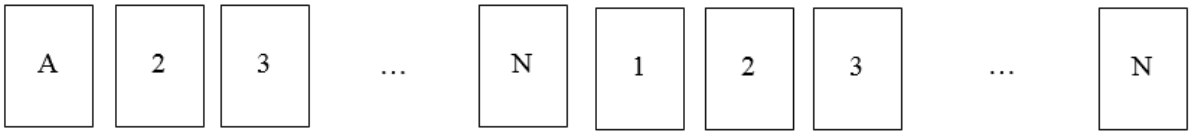
Analysis of P:

- Case: $(\langle M_1 \rangle, \langle M_2 \rangle) \in L_{\mathsf{EQ}}$
  Then for all strings $x \in \Sigma^*$, one of the following is true
  - $x \in L(M_1)$ and $x \in L(M_2)$
  - $x \notin L(M_1)$ and $x \notin L(M_2)$
- Case: $(\langle M_1 \rangle, \langle M_2 \rangle) \notin L_{\mathsf{EQ}}$ Then for some string $x \in \Sigma^*$, one of the following is true
  - $x \in L(M_1)$ and $x \notin L(M_2)$
  - $x \notin L(M_1)$ and $x \in L(M_2)$

Card Shuffling (60 points)

Let $C = (N, R)$ be a card-shuffling game, described by a natural number N and a set of allowed card-shuffling rules R. We introduce some notation and define the game below.

- Normal Deck: A deck of N cards labeled $(1, 2, 3, ..., N)$.
- Ace Deck: A normal deck but with the first card replaced by an Ace. In other words, a deck of N cards labeled $(A, 2, 3, ..., N)$.
- Shuffling Rule: A shuffling rule specifies an allowed shuffle (permutation) of any finite number of cards. In particular, each rule is a tuple $(\{A, 1, 2, ..., N\}^k, \{A, 1, 2, ..., N\}^k)$, where the left side of the tuple specifies the card sequence that can be shuffled with this rule, and the right side of the tuple specifies the new card sequence after the shuffle. For example, $((2, 7, 4, 1), (1, 2, 4, 7))$ means that if we see cards labeled $2, 7, 4, 1$ next to each other in this order, then we can shuffle/rearrange them to be in the new order of $1, 2, 4, 7$. R is the set of allowed shuffling rules and is a subset of the set of all possible shuffles.
- Game Setup: In a row, we place face-up the cards of an Ace deck in order and then place face-up the cards of a normal deck in order.



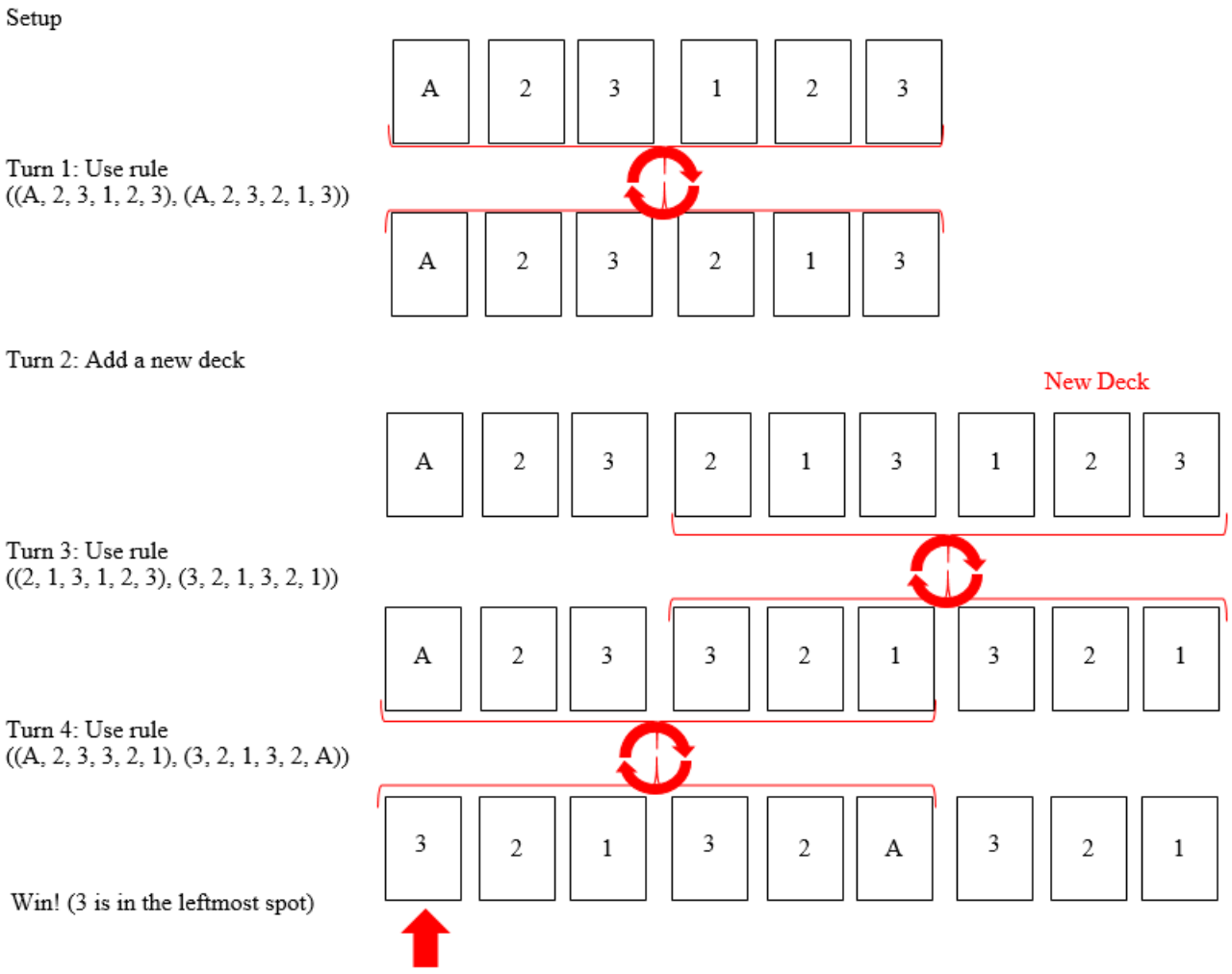
- Gameplay: Each turn, we may do one of the following:
  - Add a normal deck: Place face-up the cards of a normal deck in order to the right of all cards currently placed.
  - Perform a shuffle: Choose any number of consecutive cards and shuffle them according to some allowed shuffle rule in R. In other words, choose any number of consecutive cards that are in the order specified in the left hand side of some rule in R, and then reorder them to be in the order specified by the right hand side of the same rule.
- Win Condition: You win if you can get a card labeled N into the leftmost card position by playing this game. You can use as many turns as you want. (See example on next page.)

Let $L = \{C = (N, R) \mid \text{It is possible to win game } C\}$. Prove that L is undecidable.
Hint: Consider shuffles of 2N or 3N cards.

You may 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.

Remember to first write your intuition before writing your formal solution.

Example:
Let $C =$
$\{3, \{((A, 2, 3, 1, 2, 3), (A, 2, 3, 2, 1, 3)),\ ((2, 1, 3, 1, 2, 3), (3, 2, 1, 3, 2, 1)),\ ((A, 2, 3, 3, 2, 1), (3, 2, 1, 3, 2, A))\}$.
Then, you can win game C after four turns.



Name: Erynn-Marie Phan

Student ID: 504 757 459

Intuition:

- Let each normal deck represent a tape character.
- Different permutations of the normal deck can represent different tape characters.
- Let the ace deck represent the state.
- Different permutations of the ace deck represent different states
- The position of the ace deck in the entire game is the head position.
- To write and move the head Right, use a shuffle with 2N cards.
- To write and move the head Left, use a shuffle with 3N cards.
- The normal deck in order is the blank character.
- Add normal decks whenever the machine sees a blank and moves the head right.
- The ace deck in order is the start state.
- All halting states have the Nth card in the 1st position.
- The start of the game is the initial configuration of a Turing machine on input ε.
- Then deciding that the game can be solved is the same as deciding that the Turing machine halts on input ε.

Formal Solution:
We can define the following procedure to convert a Turing machine M into an instance $C = (N, R)$ of the card-shuffling game, in such a way that $M(\varepsilon)$ halts if and only if $C \in L$.

$TM - to - C(\langle M \rangle)$:

- Let Q_{tot} be the set of M's states
- Let $H = \{h_1, h_2, \dots\}$ be the set of M's halting states
- Define $Q = Q_{tot} \setminus H = \{q_s, q_2, q_3, \dots\}$
- Let $\Gamma = \{\sqcup, a_2, a_3, \dots\}$ be M's tape alphabet
- Let δ be M's transition function
- Let $N = max(|Q| + 1, |H| + 1, |\Gamma| + 1)$
- Let the following sequences of N cards correspond to states in Q
  - $(A, 2, 3, \dots N)$ represents q_s, the start state
  - $(2, A, 3, 4, \dots N)$ represents q_2
  - $(i, A, \dots i-1, i+1 \dots N)$ represents q_i for all $q_i \in Q$ such that $i > 2$
- Let the following sequences of N cards correspond to states in H
  - $(N, 1, 2, \dots N-1)$ represents h_1
  - $(N, 2, 1, 3, 4, \dots N-1)$ represents h_2
  - $(N, i, 1, \dots i-1, i+1, \dots N-1)$ represents h_i for all $h_i \in H$ such that $i > 2$
- Let the following sequences of N cards correspond to characters in Γ
  - $(1, 2, 3, \dots N)$ represents $\sqcup$
  - $(2, 1, 3, \dots, N)$ represents a_2
  - $(i, 1, \dots i-1, i+1 \dots N)$ represents a_i for all $a_i \in \Gamma$ such that $i > 2$

Name: Erynn-Marie Phan

Student ID: 504 757 459

- Let the rules of R be defined as follows
  - For every transition $\delta(q, a) = (q', a', Right)$ where $q \in Q$, $q' \in Q_{tot}$, $a, a' \in \Gamma$
    - Let S_q be the sequence of N cards that represents q
    - Let S_a be the sequence of N cards that represents a
    - Let $S_{q'}$ be the sequence of N cards that represents q'
    - Let $S_{a'}$ be the sequence of N cards that represents a'
    - Add the rule $((S_q S_a), (S_{a'} S_{q'}))$ to R
  - For every transition $\delta(q, a) = (q', a', Left)$ where $q \in Q$, $q' \in Q_{tot}$, $a, a' \in \Gamma$
    - Let $S_q, S_a, S_{q'}$, and $S_{a'}$ be defined as in the previous case
    - For every character $b \in \Gamma$ let S_b be the card sequence that represents b
    - For every possible S_b, add the rule $((S_b S_q S_a), (S_{q'} S_b S_{a'}))$ to R

Analysis of $TM - to - C$:

- Concerning the way card sequences are assigned:
  - Let S_{q_s} be the sequence of N cards that represents the start state
  - Let $S_\sqcup$ be the sequence of N cards that represents $\sqcup$.
  - Then the starting configuration of the game is $S_{q_s} S_\sqcup$. This corresponds to M being in its start state, with the head pointing to the first tape position, when the input is ε.
- Concerning the rules:
  - All sequences S_q have the Nth card in the Nth position and exactly one instance of the card A.
  - All sequences S_a have the Nth card in the Nth position and exactly one instance of the card 1.
  - Because of these constraints, it is not possible to apply a rule of the first form (corresponding to a $Right$ transition) to a sequence of $2N$ cards which does not represent a state and a tape character, in that order.
  - Additionally, all sequences S_b have the Nth card in the Nth position and exactly one instance of the card 1.
  - Because of these constraints, it is not possible to apply a rule of the second form (corresponding to a $Left$ transition) to a sequence of $3N$ cards which does not represent a tape character, a state, and a tape character, in that order.
- Furthermore, because the starting configuration of the game corresponds to the starting configuration of M and all rules in R correspond to valid transitions of M, all valid configurations of C correspond to valid configurations of M.
- Because all Turing machines can be converted into equivalent machines which never attempt to move $Left$ from the first tape position, all Turing Machines can be converted into instances of the card-shuffling game with this procedure.
- The only allowed card sequences which have N in the first position are the ones which correspond to halting states. Therefore the only way to get an N card in the first position on the board is to have M reach a halting state with the head at the first position on the tape.
- Because all Turing machines can be converted into equivalent machines which only halt with the head at the first tape position, determining whether there is a solution to the card-shuffling game is equivalent to determining whether an arbitrary Turing machine M halts on input ε.

Name: Erynn-Marie Phan

Student ID: 504 757 459

Claim: L is undecidable.
Proof:
Suppose there exists a decider D for L. Construct a new machine N that runs the following algorithm:

- On input $\langle M \rangle$, M is a Turing machine that never attempts to move left off the tape and only halts with its head at the first tape position.
- $(N, R) = TM - to - C(\langle M \rangle)$
- Run $D((N, R))$
  - If it accepts, accept.
  - If it rejects, reject.

Then N is a decider for $L_{\mathsf{Halt}_\varepsilon}$
$\Rightarrow\!\!\!\!\!\!\not\Leftarrow$