You have 3 hours to complete this exam. You may state without proof any fact taught in class or assigned as homework.

(4 pts) 1. Draw a transition diagram for a Turing machine that decides the language of binary strings with equally many zeroes and ones.

**Solution.** The implementation below repeatedly scans the input string, matching zeroes and ones in pairs. Smileys represent previously matched symbols.
2 Give a detailed verbal description of a multi-tape deterministic Turing machine that accepts its input if and only if its length is a prime number.

Solution. We reject all input strings of length 0 and 1, which are not primes. On longer inputs $w$, we consider each of the integers $2, 3, 4, \ldots, |w| - 1$ and see if one of them divides $|w|$. For this, we will use a Turing machine with two tapes: the input tape and an additional work tape. The Turing machine proceeds as follows.

1: Initialize the work tape to the string $\otimes \otimes$, representing the integer 2.
2: Check if the string of smileys is shorter than the input. If not, accept right away.
3: Check if the number of smileys divides $|w|$. If so, reject.
4: Append an additional smiley to the work tape, and go to Step 2.

The test on line 2 can be implemented by positioning the two heads at the left end and stepping through the two strings simultaneously, symbol by symbol, to find out if $w$ ends before the smileys. The divisibility test on line 3 can be implemented as presented in class. (Specifically, start at the left end of the tapes and cross off symbols of $w$ and smileys in pairs until there are no smileys left. Then, restore the smileys and again cross off symbols of $w$ and smileys in pairs until there are no smileys left. Continue in this fashion until every symbol of $w$ is crossed off. The number of smileys divides $|w|$ if and only if the last symbol of $w$ is crossed off at the same time as the last smiley.)

3 Prove that the following problems concerning finite automata with alphabet $\{0, 1\}$ are decidable:

- a. on input a DFA $D$, determine whether $D$ recognizes the Kleene star of some language;
- b. on input a DFA $D$, determine whether $D$ rejects all palindromes.

Solution.

a. The key observation is that $L(D)$ is the Kleene star of some language if and only if $L(D) = L(D)^*$. Indeed, if $L(D) = A^*$ for some $A$, then $L(D)^* = (A^*)^* = A^* = L(D)$. Thus, all we need to do is check whether $L(D) = L(D)^*$. For this, transform $D$ into a DFA for $L(D)^*$ and check if the resulting DFA recognizes the same language as $D$. For both of these operations, we gave an algorithm in class.

b. Rephrasing, we need to determine whether $L(D) \cap L_{\text{PAL}} = \emptyset$, where $L_{\text{PAL}}$ is the context-free language of palindromes. Construct a PDA for $L(D) \cap L_{\text{PAL}}$ by applying the cross-product construction to $D$ and a PDA for $L_{\text{PAL}}$. Then, convert the resulting PDA to a context-free grammar and see if it accepts any strings. For each of these operations, we gave an algorithm in class.
Prove that the following problems concerning pushdown automata are decidable:

(a) on input a PDA $P$, determine whether $P$ accepts more than 2017 strings;
(b) on input a PDA $P$, determine whether $P$ can ever enter an accept state with an odd number of symbols on the stack.

**Solution.**

(a) The idea is to iteratively look for strings in $L(P)$, accepting once we find 2018 strings and rejecting if we get stuck earlier. Formally:

1. $A \leftarrow \emptyset$
2. while $|A| < 2018$
3. if $L(P) \cap \overline{A} \neq \emptyset$ then
4. find a string $v \in L(P) \cap \overline{A}$
5. $A \leftarrow A \cup \{v\}$
6. else
7. return false
8. end if
9. end while
10. return true

Line 3 is implemented by constructing a DFA for $\overline{A}$ (which works by memorizing the first $k + 1$ symbols of the input, where $k$ is the longest string in $A$); applying the cross-product construction to this DFA and $P$ to obtain a PDA for $L(P) \cap \overline{A}$; converting that PDA to a context-free grammar, $G_{PA}$; and checking if that grammar generates any strings. The search in Line 4 amounts to trying out every string $v$ in lexicographic order until we find a string in $L(P) \cap \overline{A}$. The test $v \in L(P) \cap \overline{A}$ is implemented by checking whether $G_{PA}$ generates $v$ (using the algorithm from class).

(b) Let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be given. We construct a PDA $P'$ that works just like $P$ but additionally unwinds the stack before accepting, to make sure the stack has an odd number of symbols. A construction of $P$ from $P'$ is shown below, where the changes are indicated in red and $\bot$ denotes an arbitrary symbol not in $\Gamma$:

We then convert $P'$ to a context-free grammar and check if the grammar accepts any strings, using in both cases an algorithm from class.
For each of the problems below, determine whether it is decidable, and prove your answers:

(a) on input a Turing machine $M$, determine whether $M$ accepts some string of odd length;
(b) on input a Turing machine $M$, determine whether $M$ halts on all but finitely many inputs.

**Solution.**

**a.** Undecidable. Some Turing-recognizable languages contain strings of odd length and some don’t, making this a nontrivial language property. By Rice’s theorem, we conclude that it is undecidable whether a given Turing machine accepts a string of odd length.

(An alternate proof is to avoid Rice’s theorem and reduce directly from the halting problem, as done in part (b). Specifically, assume for the sake of contradiction that this language property has a decider, $T$. Then for any Turing machine $M$ and any string $w$, the algorithm in part (b) shows how to use $T$ to decide whether $M$ halts on $w$, contradicting the undecidability of the halting problem. Thus, the language property in question is undecidable.)

**b.** Undecidable. For the sake of contradiction, assume that this problem can be solved by some Turing machine, which we will call $T$. Then the following algorithm solves the (undecidable!) halting problem:

**Input:** Turing machine $M$, string $w$

1. $sourceCode ← z$
2. Run $M$ on $w$ (ignoring $z$)
3. $Accept z$
4. **if** $T(sourceCode) = “accept”$ **then**
5. **return** “$M$ halts on $w$”
6. **else**
7. **return** “$M$ does not halt on $w$”
8. **end if**

For any input $M, w$, the above algorithm constructs a Turing machine called $sourceCode$ (on the fly, specially tailored to $M$ and $w$) which halts on all but finitely many inputs if and only if $M$ halts on $w$. Then, the algorithm runs $T$ on the newly constructed machine to find out the answer.