Problem 1

Give a Turing machine that decides the language \( L = \{a^ny^{n+1}c^{n+2}\} \).

**Solution:** The language \( L = \{a^ny^{n+1}c^{n+2}\} \) is decided by the Turing Machine \( M = (Q, \Sigma, \Gamma, \delta, \text{start}, \text{accept}, \text{reject}) \) where

- the set of states is \( Q = \{0, \ldots, 8, \text{start}, \text{accept}, \text{reject}\} \)
- the input alphabet is \( \Sigma = \{a, b, c\} \)
- the tape alphabet is \( \Gamma = \{a, b, c, A, B, C, \ldots\} \), (Uppercase letters \( A, B, C \) are used to mark the input symbols \( a, b, c \) when the TM reads them off the tape.)

and the transition function is defined by the diagram in the picture. In the diagram we have used the following conventions: (1) transitions that do not overwrite the tape use abbreviated label \( x, D \) instead of \( x \rightarrow x, D \); (2) all the states have a “default” transition shown as a dotted line that is taken if no other transition applies. The default transition for state \( q_2 \) moves the tape head to the left. All other default transitions take the TM to the rejecting state, so the moving direction does not matter.

The TM consists of a main loop that includes states start, 0, 1, 2, 3, 4. When going from the start state to state 2 the TM scans the input left to right. During each scan the TM marks exactly one “a”, one “b” and one “c” when moving to states 0, 1 and 2. States 3 and 4 are used to skip previously marked “B” and “C”. When state 2 is reached, the TM moves the tape head back to the left of the tape until a marked “A” is found. Notice that the tape will always contain a string of the form “A*B*B*C*c”.

When the TM finish marking all the A’s, it checks if the tape contain a string of the form “A*B*B*C*C” going from the start state to the accept state though states 5, 6, 7, and using states 8 and 5 to skip the previously marked “B” and “C”.

Problem 2

Let \( P \) be the language over the alphabet \( \{a\} \) of all words \( a^p \) such that \( p \) is a prime number. (Remember, an integer \( p \geq 2 \) is prime if it is only divisible by 1 and \( p \) itself.) Prove that the language \( P \) is decidable, e.g. give a Turing Machine that decides \( P \). You can use any of the Turing machine variant from Chapter 3 in the book.

**Solution:** We define a Turing Machine \( M \) that on input \( a^p \) checks if \( p \) is prime by trial division, i.e., \( M \) tries to divide \( p \) by any other number \( n = 2, \ldots, p - 1 \). If any
of these numbers evenly divides \( p \) than \( a^p \) is rejected. Otherwise, \( p \) is prime and \( M \) accepts. The trial division algorithm is easily implemented using a 2-tape Turing Machine. The machine works as follows:

1. Initially the first tape contains the input \( a^p \) and the second tape is empty.

2. \( M \) first inserts a special symbol \( \$ \) at the beginning of the first tape in order to be able to recognize the left margin of the tape, and copies the content of the first tape (including the dollar sign) to the second tape. This is easily accomplished by overwriting the first \( a \) on tape 1 with a \( \$ \), and appending an \( a \) at the end of the input string. Each time a character is written on the first tape exactly the same character is written also on the second tape, so that at the end both tapes contain the string \( \$a^p \). During this scan \( M \) also checks that \( p > 1 \), and immediately rejects otherwise. (0 and 1 are not prime numbers.)

3. Move the second tape head to the left and erase one \( a \) (i.e., overwrite it with a blank symbol). This decrements the length of the second string by one.

4. At this point the two tapes always contain two strings of the form \( \$a^p \) and \( \$a^n \) (Initially \( n = p - 1 \)) and the tape heads are at the end of each string. These strings are used to represent integers \( p \) and \( n \) in the trial division algorithm.

\( M \) now checks if \( n \) divides \( p \) scanning the two strings (right to left) in parallel, and moving the second tape head to the end of the string again each time that the right margin (i.e., the \( \$ \) sign) is read. More precisely, the following two operations are executed until the exit condition is met:

(a) if both heads read an \( a \), move both tape heads one position to the left.

(b) if the first head reads \( a \) and the second \( \$ \), move the first head one position to the left, and the second head to the right end of the tape (i.e., move right until a blank is found, and then move one step to the left back to the last \( a \))

(c) if the first head reads \( \$ \), exit the inner loop

5. Now, if the second head is positioned on an \( a \), that means that \( n \) does not divide \( p \), and \( M \) can move both tape heads to the right end of the respective tapes, and start over again from step (3).

6. On the other hand, if the second tape head reads \( \$ \), then \( n \) divides \( p \) and the Turing machine terminates because a divisor of \( p \) has been found. \( M \) accepts if \( n = 1 \) (i.e., 1 is the only divisor of \( p \) smaller than \( p \)), and rejects otherwise.
Problem 3

Push down automata are very simple computational models that cannot recognize even relatively simple languages like \( L = \{ w\#w \mid w \in \{0,1\}^* \} \). To overcome this limitations we defined a more powerful model of computation: the Turing machine. In this problem we explore a different way to enhance the computational power of a push down automaton. The idea is to enhance the push down automaton with an extra stack, i.e. we consider push down automata with two stacks. In this problem you are required to formalize this model of computation, show that it can recognize the language \( L \), and finally prove that push down automata are indeed equivalent to turing machines.

(a) Give a formal definition of 2-stack push down automaton, configurations, computations and the language accepted by such an automaton

Solution: A 2-stack PDA is defined by a 6-tuple \( M = (Q, \Sigma, \Gamma, \delta, q_0, F) \) where \( Q, \Sigma, \Gamma \) are finite sets, \( q_0 \) is an element of \( Q \), \( F \) is a subset of \( Q \) and \( \delta \) is a function from \( Q \times \Sigma e \times \Gamma e \times \Gamma e \) to \( \varphi(Q \times \Gamma e \times \Gamma e) \) where \( \Sigma e \) and \( \Gamma e \) denote the extended alphabets \( \Sigma \cup \{\epsilon\} \) and \( \Gamma \cup \{\epsilon\} \).

The components have the following meanings:

- \( Q \) is the set of states
- \( \Sigma \) is the input alphabet
- \( \Gamma \) is the stack alphabet. We assume both stacks use the same alphabet.
- \( q_0 \) is the start state
- \( F \) is the set of accepting states
- the transition function \( \delta \) takes the current state \( q \), an input symbol \( a \) and the symbols on top of the two stacks \( x \) and \( y \) (possibly equal to the empty string if the automaton is not reading anything from the input or either stack), and return the set of all possible triples \( (r, w, z) \) where \( r \) is the new state and \( w \) and \( z \) are the symbols to be pushed on the stacks replacing \( x \) and \( y \). As usual \( w \) and \( z \) can be the empty string if no symbol should be pushed.

A configuration for the automaton can be represented by a triple \( (q, \alpha, \gamma_1, \gamma_2) \) where \( q \) is the current state, \( \alpha \) the portion of the input that is still to be read, and \( \gamma_1 \) and \( \gamma_2 \) are the content of the two stacks.

Transitions between configurations are defined by the following rule:

\[
(q, \alpha a, x\gamma_1, y\gamma_2) \rightarrow (r, \alpha, w\gamma_1, z\gamma_2)
\]

if \( (r, w, z) \in \delta(q, a, x, y) \).
A computation on input $w$ is a sequence of configurations $c_0, \ldots, c_n$ such that $c_0$ is the start configuration $(q_0, w, \varepsilon, \varepsilon)$ and $c_{i-1} \rightarrow c_i$ for all $i = 1, \ldots, n$. The computation is accepting if $c_n = (r, \varepsilon, \gamma_1, \gamma_2)$ for some $r \in F$ and $\gamma_1, \gamma_2 \in \Gamma^*$.

The language accepted by the automaton is defined as the set of all strings $w \in \Sigma^*$ such that $(q_0, w, \varepsilon, \varepsilon) \rightarrow^* (r, \varepsilon, \gamma_1, \gamma_2)$ for some $r \in F$ and $\gamma_1, \gamma_2 \in \Gamma^*$.

(b) Prove that any enumerable language can also be recognized by a 2-stack push down automaton, i.e., show that any turing machine can be transformed into an equivalent 2-stack push down automaton. This proves that 2-stack PDAs are at least as powerful as Turing machines.

**Solution:** We show how any single tape Turing Machine can be transformed into a 2-stack PDA. The idea is to use the first stack to represent the portion of the tape to the right of the tape head and the second stack to store the content of the tape to the left of the tape head (in reverse order).

First of all the PDA pushes two special symbols ($\$$) on the two stacks so to mark the bottoms. Before starting the simulation, the PDA reads the entire input and pushes it (symbol by symbol) on the second stack. Then the content of the second stack is moved to the first stack (one symbol at a time). At this point the PDA is in configuration $(q, \varepsilon, w\$$, \$$)$ where $w$ is the original input string. At this point configurations $(q, \varepsilon, \gamma_1\$$, \gamma_2\$$)$ of the PDA can be used to represent the corresponding TM configurations $(\gamma_2^R, q, \gamma_1)$, where $\gamma_2^R$ is the reversal of string $\gamma_2$.

The PDA can now simulate the TM one step at a time, popping symbols from $\gamma_1$ and pushing them on $\gamma_2$ to move the tape head to the right, and popping from $\gamma_2$ and pushing on $\gamma_1$ to move left.

When the TM tries to move beyond the right margin of the tape (i.e., $\gamma_1 =^n \$$$ and wants to move right), the second stack is left unchanged, and a blank is pushed on the first stack.

When the TM tries to move beyond the left margin of the tape (i.e., $\gamma_2 =^n \$$$ and wants to move left), the second stack is left unchanged, and the first symbol of the first stack is overwritten. (Remember that the TM tape is infinite only in one direction.)

(c) Now prove that 2-stack PDAs are not more powerful than Turing machines: show that if a language is accepted by a 2-stack PDA, then the language is enumerable, i.e. show that any 2-stack PDA can be transformed into an equivalent Turing machine.

**Solution:** A 2-stack PDA is just a special case of a 3-tape Turing Machine that uses its tapes in a restricted way: the first tape is only used to read the input, the other two tapes are used as two stacks always keeping the tape head to the right of the tape (which correspond to the top of the stack), and pushing and popping symbols
from either stack moving the respective tape heads to the right (for push operations) or to the left (for pop operations).

Using the equivalence of multitape TM with single tape ones, this proves that any 2-stack PDA can be transformed into a single tape Turing Machine.

Problem 4

In this problem you are asked to study how enumerable and decidable languages behave with respect to the intersection operation.

(a) Prove that the intersection of decidable languages is decidable, i.e., given two Turing machines $M_1$ and $M_2$ deciding languages $L_1$ and $L_2$, show that there exists a Turing machine $M$ that decides the language $L = L_1 \cap L_2$.

Solution: Let $M_1$ and $M_2$ be single tape Turing Machines that decides languages $L_1$ and $L_2$ and define $M$ as follows. On input $w$, $M$ first special symbols $\$ at the beginning and the end of the input. It then copies $w$ after the second $\$ and simulate $M_1$ on the portion of the tape following the second $\$. If $M_1$ rejects, then reject. Otherwise, if $M_1$ accepts, clear the part of the tape to the right of the second $\$ (including the dollar sign), move the tape head to the beginning of the tape, and simulate $M_2$ on the original input $w$. If $M_2$ accepts, then accept, and reject otherwise.

Obviously the new machine accepts the language $L_1 \cap L_2$. Notice that since $M_1$ and $M_2$ are deciders, both simulations always terminate, either accepting or rejecting. Therefore also $M$ always terminates, and therefore it decides $L_1 \cap L_2$.

Notice that $M$ can be easily modified to decide $L_1 \cup L_2$ instead of $L_1 \cap L_2$. In this case if $M_1$ accepts, then $M$ can accepts immediately. Otherwise $M$ will go on with the simulation of $M_2$.

(b) Prove that the intersection of enumerable languages is enumerable, i.e., given two Turing machines $M_1$ and $M_2$ accepting languages $L_1$ and $L_2$, show that there exists a Turing machine $M$ that accepts the language $L = L_1 \cap L_2$.

Solution: This case is particularly simple. Build $M$ exactly in the same way as in part (a). Notice that $M$ will accept if both $M_1$ and $M_2$ accept, and will loop or reject if either of $M_1$ or $M_2$ loops or rejects. Therefore, $M$ accepts (but does not necessarily decides) the language $L_1 \cap L_2$.

Notice that for the union operation $L_1 \cup L_2$ this kind of simulation (modified as described at the end of part (a)) does not work. The reason is that $M$ might loop when simulating $M_1$ in which case it won’t accept the input $w$ even if $M_2$ accepts it. To correctly build a TM machine accepting the union of the two languages, you have to simulate $M_1$ and $M_2$ in parallel (e.g., run the simulations for $n = 1, 2, \ldots$, steps until any of them accepts). Alternatively, you can give a non-deterministic TM that non-deterministically chooses either $M_1$ or $M_2$, and simulates it on the input. If you apply the conversion from NTM to (D)TM to this machine you will see that the result is essentially the deterministic machine simulating $M_1$ and $M_2$ in parallel.
Figure 1: Turing machine deciding the language $a^{n}b^{n+1}c^{n+2}$