Let $A$ be any fixed finite set of 4 or more elements. Prove that the number of subsets $A$ is less than the number of permutations of elements of $A$.

**Solution:**

Let $|A|$ denote the size of $A$. We know from the definition of permutations that the number of permutations of elements of $A$ is $|A|!$.

We must prove that the number of subsets of $A$ is equal to $2^{|A|}$. We will do so by induction.

Base case: Assume $|A| = 0$, in other words, $A$ is the empty set. Clearly, there is only one subset of $A$ which is $A$ itself. Therefore, the number of subsets of $A$ is $1 = 2^0 = 2^{|A|}$.

Induction step: Given any set $B$ such that $|B| = n$, assume that the number of subsets of $B$ is $2^n$ (Induction hypothesis). Take $A$ such that $|A| = n + 1$. We must prove that the number of subsets of $A$ is $2^{n+1}$.

Since $A$ contains $n + 1$ elements, then it contains at least one element $a$. Consider the set $B$ such that $A = B \cup \{a\}$, i.e. $B$ consists of all the elements of $A$ except $a$. Clearly, $|B| = |A| - 1 = n + 1 - 1 = n$.

By the induction hypothesis, the number of subsets of $B$ is $2^n$. The subsets of $B$ cover all the subsets of $A$ that do not include $a$. We can obtain all the subsets of $A$ that include $a$ by adding $a$ into every subset of $B$. Therefore, $A$ has twice as many subsets as $B$. The number of subsets of $A$ is $2 \times 2^n = 2^{n+1} = 2^{|A|}$. This concludes the induction proof.

We know from homework 1 (problem 1) that $2^n < n!$ for all $n \geq 4$. Therefore, the number of subsets of a set of 4 elements or more is less than the number of permutations of its elements.
2. Suppose you have a Turing machine T that computes the addition function for binary integers. So T computes \( f(n, m) = n + m \).

Describe a TM M that computes the function \( g(n) = n^2 \).

Hint: Given \( n \), you can directly use T to compute \( f(n, n) = n + n = 2n \) Also we know that \( n^2 = n + n + n + n + ... + n \), \( n \) times. So now use T repeatedly to get a TM M as described above.

Solution:

\[ M = \text{“on input string } <n> \text{ (binary representation of n):} \]

1. Copy \( <n> \) twice to the end of the tape and separate the copies with \# so that the tape becomes ‘\(<n>\#<n>\#<n>\)’.

2. Replace every digit of the first \( <n> \) with zeros and the last digit with 1, so that the tape becomes ‘000...1\#<n>\#<n>’, now the first part is the binary representation of 1 padded with zeros on the left.

3. Assume the tape is on the form ‘\(<c>\#<n>\#<m>\)’. Run machine T on inputs \(<n>, <m>\). Erase \(<m>\) from the tape and write down the result of T in its place at the end of the tape.

4. Run machine T on inputs \(<c>, <1>\). Erase \(<c>\) and write down the result of T in its place. Pad the result with zeros from the left so that it fits exactly in the space on the tape.

5. The tape now must look like ‘\(<c+1>\#<n>\#<m+n>\)’. If \(<c+1> = <n>\) go to step 6, otherwise, go back to step 2 and repeat.

6. Copy the last part of the string to the beginning of the tape and erase the rest. This string is required output.”

The machine is simple conceptually. The machine uses T to add \( n \) to itself and then to add \( n \) to the result repeatedly. It stores the result of every addition at the end of the tape since the result may become larger and larger in size and need more space. The machine keeps a counter in the beginning of the tape, and increases it with every iteration. When the counter reaches \( n \), the machine stops, since adding \( n \) to itself \( n \) times results in \( n^2 \). The counter will always fit in the space provided, since the counter will never be bigger than the size of \( n \).
3. For this problem you may assume that the sets L and J are both subsets of N.

i. Show that any infinite decidable language L has an infinite decidable subset.
Note: This is a trick question.

Solution:

Since any set is a subset of itself by definition, then L itself is the required subset.

ii. Show that any infinite decidable language L has an infinite decidable subset J with the
property that \( L - J \) is also infinite.

Solution:

Since L is decidable, then L has an enumerator E that enumerates the strings in the language in
shortlex order. This means that \( w_1 < w_2 \) if and only if E prints \( w_1 \) before \( w_2 \), for any \( w_1, w_2 \in L \).
E never prints the same string twice.

We will construct two enumerators E1 and E2 that will enumerate two languages \( L_1, L_2 \subseteq L \)
respectively. We will show that both \( L_1 \) and \( L_2 \) is infinite, and that \( L - L_1 = L_2 \).

E1 = “ignore the input:

1. Declare some flag and initially set to true. This can be represented by dividing the states of
the enumerator into two groups, where the first group means the flag is true and the second
means it is false.

2. Run the enumerator E.

3. If E outputs a string w check the flag, if it is true, print w. If the flag is false, ignore w.

4. Flip the value of the flag and repeat the previous step.”

E2 is identical to E1, but the flag is initially set to false.

Both E1 and E2 enumerates a subset of L, since they only print out strings that E enumerates.
Every string enumerated by E is enumerated by either E1 or E2. E1 and E2 never print the same
string, since E1 prints the strings that appear in even positions in the list produced by E, and E2
only prints the strings that appear in odd positions. Since E enumerates an infinite set of strings,
then both E1 and E2 also enumerate infinite sets.

Therefore E1 and E2 enumerate in order two decidable sets \( L_1 \) and \( L_2 \), such that both languages
are infinite, \( L - L_1 = L_2 \), and both \( L_1 \) and \( L_2 \) are subsets of L.
iii. Does the statement in part i of this problem still true if $L$ is only recognizable. (That is, is it true that any infinite recognizable language $L$ has an infinite decidable subset?) If it is true, say why. If not true, find a counter example.

**Solution:**

Yes it does. Take $L$ to be an infinite Turing recognizable language. $L$ has some enumerator $E$. However, $E$ does not enumerate $L$ in an order and may output the same string many times.

We will construct a new Enumerator $E_1$, such that $E_1$ enumerates an infinite subset of $L$ in order.

$E =$ “ignore the input:

1. Erase the tape and make it empty.
2. Run $E$.
3. Whenever $E$ prints out a string $w$, check if the length of $w$ is greater than the length of the used part of the tape (an empty tape has length 0).
4. if $|w| \leq |tape|$, ignore $w$.
5. if $|w| > |tape|$, print $w$. Erase the tape and write $w$ on it.”

$E_1$ only prints strings that $E$ prints. Therefore the language of $E_1$ is a subset of $L$. Since $E_1$ remembers that last string it printed and only prints a new string if it has a greater length, then $E$ always prints strings in increasing order of size (shortlex order).

Since $L$ is infinite, then no matter what $w$ is, $L$ must contain strings with length larger than the length of $w$, otherwise, $L$ will not be infinite, since there are only finitely many strings with size less than or equal to the length of $w$. Therefore, for any $w$, $E$ will always find a new string to print. Therefore, $E$ prints infinitely many strings, and the language of $E$ is a decidable infinite subset of $L$.

4. Assume that your TMs are allowed to have the read head stay (S) where it is instead of only moving L or R at any transition. So a line of the TM program species either an L, S, or R tape head move. Show that any such $M$ can be simulated by a standard TM $T$ which only moves L or R as it computes.

To do this you should explicitly say how, when given a machine $M$ as above, you can change its program into a standard program for $T$ which accepts and rejects the same strings. You are free to add some new symbols or new states to $T$ if that helps.

**Solution:**

4
The modified Turing machine \(M\) in the question has all the standard Turing machine functionality, but it also has an additional command (\(S\)) that leaves the head where it is instead of moving it left or right.

A regular Turing machine \(T\) can directly simulate all the possible commands of \(M\) except \(S\). We need only show that \(T\) can simulate the \(S\) command of \(M\) to prove that \(T\) can simulate \(M\).

The program of \(T\) will be as follows:

- For every state \(q_i\) of \(M\), \(T\) will include two states, the original state \(q_i\) and a corresponding additional state \(q'_i\).
- The accept, reject, and start state of \(T\) will be the same as \(M\).
- The input and tape alphabets of \(T\) will be the same as \(M\).
- The transition function \(\delta_T\) of \(T\) will be as follows:
  1. \(\delta_T(q_i, x) = \delta_M(q_i, x)\) if \(\delta_M(q_i, x)\) moves the head either left or right.
  2. \(\delta_T(q'_i, x) = (q_i, x, L)\) if \(T\) was in one of the additional states.
  3. \(\delta_T(q_i, x) = (q'_j, y, R)\) if \(\delta_M(q_i, x) = (q_j, y, S)\).

\(T\) clearly simulates any step of \(M\) that does not involve a \(S\) command. To prove that \(T\) simulates \(M\), we only need to consider an arbitrary step of \(M\) that includes \(S\).

Assume that during the execution of \(M\), we encounter an \(S\) command as follows:

\[
\delta_M(q_i, x) = (q_j, y, S)
\]

In other words, \(M\) was in state \(q_i\) and read some character \(x\), \(M\) moves to state \(q_j\) and overwrites \(x\) with some character \(y\) and keeps the head in its place.

In that case, \(T\) will behave as dictated by its transition function:

\[
\delta_T(q_i, x) = (q'_j, y, R)
\]

\(T\) will move to the addition state \(q'_j\) that corresponds to the target state of \(M\) \((q_j)\). It will overwrite \(x\) with the same character \(M\) uses \((y)\), then it will move the head to the right.

In the next step, \(T\) will be in state \(q'_j\) and read some character \(z\).

\[
\delta_T(q'_j, z) = (q_j, x, L)
\]

\(T\) will move to the state \((q_j)\). \(T\) will move the head to the left, going back to the same cell it was in before. Therefore, after this step, \(T\) and \(M\) will be in the same state \(q_j\), the heads of \(T\) and \(M\) will be pointing at the same cell, and \(T\) and \(M\) will both read the same character \(y\).

Therefore, \(T\) can simulate any step of \(M\) that involves an \(S\) command in 2 steps, moving the head right and modifying the character on the tape in the first, while moving the head back left and going into the correct state in the second. Therefore \(T\) can simulate \(M\).
5. Page 161 (Page 189), problem 3.15 parts d. and e.

Solution:

Part d: Prove that decidable languages are closed under complementation.

Let $L_1$ be an arbitrary decidable language, let $M_1$ be a Turing machine that decides $L_1$. Let $\overline{L_1}$ be the complement of the language, i.e. $L_1 \cap \overline{L_1} = \emptyset$, $L_1 \cup \overline{L_1} = \Sigma^*$. \overline{L_1} is made of all the strings not in $L_1$. We will construct $M_2$ that decides $\overline{L_1}$.

$M_2 =$ “on input string $w$:
1. run $M_1$ on input $w$.
2. if $M_1$ accepts, reject. if $M_1$ reject, accept.”

If $w$ is in $\overline{L_1}$, then $w$ is not in $L_1$ and $M_1$ will reject $w$. Therefore $M_2$ will accept $w$. Similarly, if $w$ is not in $\overline{L_1}$, then $w$ is in $L_1$ and $M_1$ will accept $w$. Therefore $M_2$ will reject $w$. Since $M_1$ is a decider (it terminates on all inputs) then $M_2$ is also a decider. Therefore $M_2$ decides $\overline{L_1}$.

Part e: Prove that decidable languages are closed under intersection.

Let $L_1$, $L_2$ be arbitrary decidable languages, let $M_1$, $M_2$ be Turing machines that decide $L_1$, $L_2$ respectively. Let $L_3 = L_1 \cap L_2$ be the intersection of the two languages, $L_3$ is made of all the strings that are members of both $L_1$ and $L_2$. We will construct $M_3$ that decides $L_3$.

$M_3 =$ “on input string $w$:
1. run $M_1$ on input $w$.
2. if $M_1$ rejects, reject.
3. if $M_1$ accepts, run $M_2$ on input $w$.
4. if $M_2$ also accepts, accept.
5. if $M_2$ rejects, reject.”

$M_3$ will only accept if $w$ is accepted by both $M_1$ and $M_2$. If either one of $M_1$ or $M_2$ (or both) rejected then $M_3$ will reject. Since $M_1$ and $M_2$ are both deciders (they terminate on all inputs), then $M_3$ is also a decider. Therefore $M_3$ decides $L_3$, and $L_3$ is decidable.
6. Let $A = \{0, 1, 2\}$, and let language $L$ be defined by $L = \{wawbw \mid w \in A^*\}$. 

(i). What is $|A^4|$ (that is, the number of elements in $A^4$)? (Note: Here I use $A^4$ to mean the concatenation of $A$ with itself 4 times = $AAAA = \{wxyz \mid w,x,y,z \in A\}$. (See page 14 of the textbook.)

What is $|A^n|$? What is $|A^*|$?

**Solution:**

$A^4$ is the set of all strings of size 4 that are made up of 0s, 1s, and 2s. For example, the following strings are all in $A^4$: 0012, 1210, 1122, 0000, ... Since strings in $A^4$ have 4 characters and each character has 3 options (0, 1, or 2), then the total number of such strings is $3^4 = 81$.

Similarly, the size of $A^n$ is $3^n$. $A^*$ is the set of all strings of any size that are made from 0s, 1s, and 2s. $A^*$ is [countably] infinite.

(ii). Thinking of the language $L$, what is its alphabet?

How many elements are there in $L$ which have length 8?

**Solution:**

$L$ contains strings on the form ‘wawbw’ where $w$ is made up of characters from $A$ (0s, 1s, and 2s). Therefore, strings in $L$ are made up of the characters 0,1,2,a,b. Some strings may contain only some of these characters. No string in $L$ can contain any additional character. The alphabet of $L$ is $A \cup \{a,b\} = \{0, 1, 2, a, b\}$.

Strings in $L$ are on the form ‘wawbw’, since $a$ and $b$ are characters and have length 1, a string in $L$ can only be of length 8 if $ww$ is of length 8 - 2 = 6, and thus if $w$ is of length 6/3 = 2. The question now becomes, how many strings in $A^*$ have length 2, in other words, what is the size of $A^2$. From part (i) we know that this is equal to $3^2 = 9$. 
