1. We can formally define an enumerator to be similar to an ordinary two-tape Turing machine with the exception that its δ function differs as the following:

   $\delta : Q$ x $\Gamma \rightarrow$ Q x Γ x Σ_p x $\{L, R\}$

   Where Σ_p is identical to the input alphabet but is used solely for printing enumerated strings to the output tape representing a printer.

   Initially both the work tape and the output tape are blank.

   The machine works by transitioning based on the language it is programmed to recognize. for each transition, if the machine determines that a character from Σ can be appended to the current string on the output tape, it writes that character to the output tape. Using its transition function it will eventually write the characters of some word in the language that it recognizes. When this happens it might "print" this string out as a member of the language and move to an empty portion of the tape where it will perform a similar process for some other word in the language. Eventually, if the enumerator does not halt, it may print an infinite list of strings– all of which are in the language it recognizes. Every String in the language will at some point be printed by the enumerator.

   a formal definition of the language L(E). which is the language recognized by some enumerator E is as follows:

   L(E) = {w|w is in the collection of words printed by E}

   we know from Sipser that the language enumerated by E is the collection of all the strings that it eventually prints out, albeit in any order and potentially with repetition.

---

2. In order to show that a Turing Machine M_{bad} is not a legitimate Turing Machine it is sufficient to show that for any input ¡P¿ where P is of the form described in the problem it is incapable of performing its algorithm.

   Consider some input P which is a polynomial over multiple terms. During the first step of M_{bad}'s algorithm we ask it to try all possible settings of each term to integer values. But there are infinitely many integer values to set each term to. Therefore the machine will never advance to any other step in its algorithm. Because of this the machine is useless.

---

3. In order to show that the language described in question 4.2 is decidable it is sufficient to show that there is a Turing machine which decides it.

   In order to show this we will utilize a proof by construction.

   We can construct a Turing Machine T1 which implements the algorithms of other Turing machines we know to exist from Sipser's texts.

First, we know that every Regular expression can be converted into an equivalent NFA using Lemma 1.55. so program T1 to perform this procedure on input S to construct some NFA NS.

Second, we know that for ever NFA there is an equivalent DFA. we can use theorem 1.39 to construct such a DFA DS which is equivalent to NS and as such equivalent to S.

Third, we will make use of the Turing Machine EQ_{DFA} from Theorem 4.5 which takes in as input two DFAs and decides if they are equal. We simulate the procedure of this Turing machine within T1 on the DFAs R and DS. if EQ_{DFA} accepts then T1 accepts, if EQ_{DFA} rejects then T1 rejects.

In This way we have constructed a Turing Machine which decides this language and proved that this language is decidable.

---

4. Similarly to the last problem we can prove that a language is decidable if there is some Turing Machine, we will call T2 which decides it.

   we will utilize a proof by construction as well as the property of closure under regular languages for complementation and intersection in order to show that there is a Turing Machine which decides this language and thus the language is decidable.

   First, we know that if R and S are regular expressions , then L(R) and L(S) are regular languages, therefore the complement of S, we will call S' is also a regular language. We also know that R and S can be converted into equivalent DFAs just as S was in the last problem. T2 performs this operation, gaining the ability to simulate DFAs DS and DR which are equivalent to S and R respectively.

   Second, we know we can convert any DFA which recognizes some language, to recognize the complement of that language. T2 performs the procedure for this on DS to gain the ability to simulate DS' which is the DFA which recognizes S'.

   Third, we know that for any two DFA's we can construct a DFA which recognizes the intersection of the languages they recognize. T2 performs this operation on DS' and DR which produces another virtual DFA we will call DRS', gaining the ability to recognize the language equivalent to S' $\cap$ R.

   we know that any word which DRS' recognizes is in R and not in S. Therefore if DRS' recognizes any word at all, then R can not be a subset of S, because R will contain a word which is not in S.

   So how do we show that the language DRS' recognizes is empty and decide if R is a subset of S? we use our trusty friend, the Turing Machine EQ_{DFA} from theorem 4.5. So we program T2 to have the ability to simulate EQ_{DFA}. We also must program T2 construct a DFA which recognizes only the empty Language, we will call DE.

   Finally, T2 simulates EQ_{DFA} on the DFAs DE and DRS'. If EQ_{DFA} Accepts we know they are equal and thus, S' $\cap$ R = the empty language. if S' $\cap$ R = the empty language then we know L(R) is a subset of L(S), and thus T2 accepts. IF EQ_{DFA} rejects, then T2 rejects.

In this way we have constructed a Turing Machine which decides this language and proved that this language is decidable.

---

5. EXTRA CREDIT

If there is a root at x = x0, then we know that some c_1 must have the highest positive or negative value. Therefore, it's absolute value must be the greatest making it CMAX. Then there can be no absolute value of x0 which is greater. since n can not be negative multiplying a greater than or equal value by (n+1) will also produce an even greater value.