Honor Code: By my signature, I have acted honestly in writing this problem set, and I have followed the Duke Honor Code:

Signature: ______________________________________________________________________________

Extra Credit Problem

(a) Name (please print): _________________________________________________________________

(b) NetId (please print): _______________________________________________________________

1. Extra Credit.

Part A. Polynomials, Algebraic Numbers, and Computability.

Suppose, that we have an algorithm, encoded by a Scheme function Poly-check. Poly-check examines an arbitrary Scheme program p and determines infallibly whether p is an implementation of the root function, which takes a polynomial y and returns all integer roots of y. y is assumed to be a polynomial with rational coefficients, which can be represented by an integer n, where n is the maximum degree of y, and a sequence of rational coefficients

\[(a_{n-1}, a_{n-2}, \ldots, a_1, a_0)\].

Hence,

\[y = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0.\]

You may assume that not all the coefficients are zero.

For example, the polynomial

\[y = x^3 - 7x + 6\]

would be encoded by n=3 and coefficients (0, -7, 6). Its roots 1, 2, and -3 equate the polynomial to zero, and hence are its integer roots. In this case we would have

\[(p (list 3 '( 0 -7 6 ))) \rightarrow ( 1 2 -3)\]

and

\[(Poly-check p) \rightarrow #t\] (assuming p works for any polynomial).

Of course your proof and your reasoning must apply to any scheme program p, not just the example above.

Does Poly-check exist? Prove your answer. Either (a) write Poly-check and prove it it correct, or (b) reduce the Halting Problem to Poly-check.

If you choose (a) your proof must use the substitution model.

If you choose (b), you must use Turing reducibility \((HP \leq_T Y)\).

NB: This problem is somewhat easier to do if we assume that the polynomials all have integer coefficients. Therefore, you may assume this in doing the problem.
Imagine we are writing an interpreter or a compiler.

Suppose there exists a predicate \texttt{expr=zero?} takes an arbitrary Scheme expression \texttt{expr} as input, and returns \texttt{#t} if that expression evaluates to 0 and \texttt{#f} if it does not.

For example, suppose we have in the compiler:

\begin{verbatim}
(let* ((t1 expr)
       (t3 t1)))
\end{verbatim}

If \texttt{expr} = 0, we can optimize this to

\begin{verbatim}
(t3 0)
\end{verbatim}

which is much better and potentially much faster.

So how can we tell if an expression \texttt{expr} = 0? Sounds easy, and various cases seem pretty straightforward:

\begin{verbatim}
(+ 3 7 -10) = 0.
(+ (* 2 0) (- 1 1))) = 0.
\end{verbatim}

and

\begin{verbatim}
((lambda (a b c x) (+ (* a (square x)) (* b x) c))
  2 4 -6 1)
\end{verbatim}

is zero exactly when

\[
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

where \(a = 2\), \(b = 4\), \(c = -6\), and \(x = 1\) in this example.

Notice that, for example,

\begin{verbatim}
((lambda (a b c x) (+ (* a (square x)) (* b x) c)) 2 4 -6 1)) = 0
\end{verbatim}

since \(2 \cdot 1^2 + 4 \cdot 1 - 6 = 0\), and

\begin{verbatim}
((lambda (a b c x) (+ (* a (square x)) (* b x) c)) 2 4 -6 2)) = 10
\end{verbatim}

since \(2 \cdot 2^2 + 4 \cdot 2 - 6 = 10\).

Hence, we wish to obtain:

\begin{verbatim}
(expr=zero? '((lambda (a b c x) (+ (* a (square x)) (* b x) c)) 2 4 -6 1)) → #t, and
(expr=zero? '((lambda (a b c x) (+ (* a (square x)) (* b x) c)) 2 4 -6 2)) → #f.
\end{verbatim}

Can we do it in general?

Suppose you are asked to write a Scheme function \texttt{expr=zero?} that works for \textit{any} expression, (not just for the examples above).

Does \texttt{expr=zero?} exist? Prove your answer. Either (a) write \texttt{expr=zero?} in Scheme and show that it is correct, or, (b) prove that \texttt{expr=zero?} does not exist.
If you choose (a) your proof must use the substitution model.

If you choose (b), you must use Turing reducibility ($HP \leq_T Z$).