1. Consider the following lexically scoped language of integer expressions:

\[ exp ::= \begin{align*}
  &NUM \\
  &VAR \\
  &let~VAR = exp_1\text{ in } exp_2 \\
  &exp_1 + exp_2
\end{align*} \]

Give an attribute grammar that computes the value of an expression. You may assume that
\(NUM.value\) is the integer value of the numeric literal and that \(VAR.name\) is the name of a variable. Your solution may use functional data structures, such as sets and finite maps.

2. Consider the following representation of terms in SML:

\[
\text{datatype} \quad \text{term} = T \text{of} \quad (\text{string} \times \text{term} \text{ list})
\]

where the \text{string} is the operator name. It is possible to define strategy combinators for this term representation, where a strategy has the type

\[
\text{type} \quad \text{strategy} = \text{term} \rightarrow \text{term} \text{ option}
\]

and \text{NONE} denotes failure. For example,

\[
\begin{align*}
\text{fun} \quad &\leftarrow (s_1, s_2) \ t = (\text{case} \ s_1 \ t \\
&\text{of} \ \text{NONE} => s_2 \ t \\
&\text{someT} => \text{someT} \\
&(*\ \text{end case} \ *))
\end{align*}
\]

implements deterministic choice and

\[
\begin{align*}
\text{fun} \quad &\text{all} \ s \ (T(f, \ args)) = \text{let} \\
&\text{fun} \ \text{try} \ ([], \ l) = \text{SOME}(T(f, \ \text{List.rev} \ l)) \\
&\text{try} \ (t::ts, \ l) = (\text{case} \ s \ t \\
&\text{of} \ \text{NONE} => \text{NONE} \\
&\text{SOME} \ t' => \text{try}(ts, \ t':l) \\
&(*\ \text{end case} \ *)) \\
&\text{in} \\
&\text{try} \ (args, \ []) \\
&\text{end}
\end{align*}
\]

implements the \text{all} combinator.

(a) Give the SML code for the \text{test} combinator. Recall that the \text{test} combinator acts as the identity when its argument succeeds and fails when its argument fails.

(b) Give the SML code for a generic congruence operator with the following specification:

\[
\text{val} \ \text{congruence} : (\text{string} \times \text{strategy} \text{ list}) \rightarrow \text{strategy}
\]