1. Recall the discussion of polymorphic typechecking in Handout 5. Assume that we have both \texttt{int} and \texttt{real} as base types. To extend the typechecker to support overloaded functions on integers and reals (\textit{e.g.,} \texttt{"+"}), we need to allow type variables that are restricted to be members of some set. For example, the type of \texttt{"+"} could be written as

\[
\forall \alpha \in \{\texttt{int}, \texttt{real}\}.(\alpha \times \alpha) \rightarrow \alpha
\]

We can model this by changing the representation of type-variable kinds:

```plaintext
and tvar_kind
    = INSTANCE of ty
    | UNIV of int
    | NUMKIND
```

Give a modified version of the unification algorithm from Section 4 of Handout 5 that deals with this new kind representation.

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

\[
\begin{align*}
\text{exp} & ::= \text{NUM} \quad \text{\text{(1)}} \\
& \quad | \text{VAR} \quad \text{\text{(2)}} \\
& \quad | \text{exp}_1 \text{ where } \text{VAR} = \text{exp}_2 \quad \text{\text{(3)}} \\
& \quad | \text{exp}_1 + \text{exp}_2 \quad \text{\text{(4)}}
\end{align*}
\]

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