Consider the following pure functional language, in which \( n \) ranges over the mathematical integers.

\[
T ::= \text{int}1 | \text{int}8 | \text{int}16 | \text{uint} | \text{uint}8 | \text{uint}16 | T \rightarrow T' | T \ast T' | T + T'
\]

\[
e ::= n | e +_T e' | x | \text{fn } x : T \Rightarrow e | e e' | (e, e') | \#_1 e | \#_2 e | \text{inl}_T e | \text{inr}_T e
\]

\[
\text{case } e \text{ of } \text{inl} (x_1 : T_1) \Rightarrow e_1 | \text{inr} (x_2 : T_2) \Rightarrow e_2
\]

Its operational semantics is defined as a relation \( e \rightarrow e' \) with the standard rules for a pure call-by-value left-to-right functional language, except with the following rules for addition of values. As usual, the expression \( n +_T n' \) is stuck if one of these does not apply.

\[
\begin{align*}
n \in -2^{N-1} & \ldots 2^{N-1} - 1 & n \in 0 \ldots 2^N - 1 \\
n' \in -2^{N-1} & \ldots 2^{N-1} - 1 & n' \in 0 \ldots 2^N - 1 \\
n'' = n + n' & n'' = n + n' \\
n'' \in -2^{N-1} & \ldots 2^{N-1} - 1 & n'' = n'' \mod 2^N
\end{align*}
\]

1. Define a subtype relation \( T <: T' \) and type relation \( \Gamma \vdash e : T \) for this syntax and operational semantics that will permit flexible use of integers in the appropriate ranges. You can omit the standard type relation rules for the expressions \( (e, e') \), \#_1 e, \#_2 e, \text{inl}_T e, \text{inr}_T e \), and \text{case}. [14 marks]

2. Explain three main aspects of your definitions, with reference to the programming idioms they permit and the runtime errors they exclude, with examples. [6 marks]