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

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

\[
e ::= n \mid e + T e' \mid x \mid \text{fn} \ x : T \Rightarrow e \mid e e' \mid (e, e') \mid \#1 e \mid \#2 e \mid \text{inl} \ T e \mid \text{inr} \ T e
\]

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

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.

\[
\begin{align*}
n & \in -2^{N-1} \ldots 2^{N-1} - 1 \\
n' & \in -2^{N-1} \ldots 2^{N-1} - 1 \\
n'' & = n + n' \\
n''' & \in -2^{N-1} \ldots 2^{N-1} - 1 \\
n + \text{int}_N n' & \rightarrow n'' \\
n + \text{uint}_N n' & \rightarrow n'''
\end{align*}
\]

(a) 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, \text{and case}. [14 marks]

(b) 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]