1 STLC with Pairs

We want to add types Pairs to the Simply Typed Lambda Calculus with Booleans that we studied in the book and lectures. A Pair is the equivalent of Scala's 'Tuple2'. You can find a longer introduction to Pairs in the book, Section 11.6. For the sake of the exercise, we stay at a dry level of summarizing the syntactic forms, evaluation rules and typing rules:

$\mathtt{t}$ ::=		**terms** :
	$\mathtt{x}$	variable
	$\lambda \mathtt{x{:}T.t}$	abstraction
	$\mathtt{t\ t}$	application
	$\mathtt{true} \mid \mathtt{false}$	boolen values
	$\mathtt{if\ t\ then\ t\ else\ t}$	conditional
	$\{\mathtt{t,\ t}\}$	pair
	$\mathtt{t.1} \mid \mathtt{t.2}$	projection
$\mathtt{v}$ ::=		**values** :
	$\lambda \mathtt{x{:}T.t}$	abstraction
	$\mathtt{true} \mid \mathtt{false}$	boolen values
	$\{\mathtt{v,\ v}\}$	pair value
$\mathtt{T}$ ::=		**types** :
	$\mathtt{T} \to \mathtt{T}$	function type
	$\mathtt{Bool}$	boolean type
	$\mathtt{T} \times \mathtt{T}$	pair type

Small-step reduction rules:

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2}\ (\text{E-App1}) \qquad \frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1.1 \longrightarrow \mathtt{t}_1'.1}\ (\text{E-Proj1})$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'}\ (\text{E-App2}) \qquad \frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1.2 \longrightarrow \mathtt{t}_1'.2}\ (\text{E-Proj2})$$

$$(\lambda \mathtt{x{:}T.t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}\ (\text{E-AppAbs})$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\{\mathtt{t}_1,\ \mathtt{t}_2\} \longrightarrow \{\mathtt{t}_1',\ \mathtt{t}_2\}}\ (\text{E-Pair1}) \qquad \{\mathtt{v}_1,\ \mathtt{v}_2\}.1 \longrightarrow \mathtt{v}_1\ (\text{E-PairBeta1})$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\{\mathtt{v}_1,\ \mathtt{t}_2\} \longrightarrow \{\mathtt{v}_1,\ \mathtt{t}_2'\}}\ (\text{E-Pair2}) \qquad \{\mathtt{v}_1,\ \mathtt{v}_2\}.2 \longrightarrow \mathtt{v}_2\ (\text{E-PairBeta2})$$

Typing rules:

$$\frac{\mathtt{x{:}T} \in \Gamma}{\Gamma \vdash \mathtt{x{:}T}}\ (\text{T-Var}) \qquad \Gamma \vdash \mathtt{true{:}Bool}\ (\text{T-True})$$

$$\frac{\Gamma, \mathtt{x{:}T}_1 \vdash \mathtt{t}_2\mathtt{{:}T}_2}{\Gamma \vdash (\lambda \mathtt{x{:}T}_1\mathtt{.t}_2)\mathtt{{:}T}_1 \to \mathtt{T}_2}\ (\text{T-Abs}) \qquad \Gamma \vdash \mathtt{false{:}Bool}\ (\text{T-False})$$

$$\frac{\Gamma \vdash \mathtt{t}_1\mathtt{{:}T}_1 \to \mathtt{T}_2 \qquad \Gamma \vdash \mathtt{t}_2\mathtt{{:}T}_1}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2\mathtt{{:}T}_2}\ (\text{T-App}) \qquad \frac{\Gamma \vdash \mathtt{t}_1\mathtt{{:}Bool} \qquad \Gamma \vdash \mathtt{t}_2\mathtt{{:}T} \qquad \Gamma \vdash \mathtt{t}_3\mathtt{{:}T}}{\Gamma \vdash \mathtt{if\ t}_1\ \mathtt{then\ t}_2\ \mathtt{else\ t}_3\mathtt{{:}T}}\ (\text{T-If})$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \{\mathtt{t}_1, \mathtt{t}_2\} : \mathtt{T}_1 \times \mathtt{T}_2} \text{ (T-Pair)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \times \mathtt{T}_2}{\Gamma \vdash \mathtt{t}_1.1 : \mathtt{T}_1} \text{ (T-Proj1)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \times \mathtt{T}_2}{\Gamma \vdash \mathtt{t}_1.2 : \mathtt{T}_2} \text{ (T-Proj2)}$$

1. State the Canonical Forms lemma for this new system, without proving it.
2. Extend the proof of Progress of STLC with Booleans to deal with Pairs. Only handle the cases that are specific to Pairs.
3. Likewise, extend the proof of Preservation.

2 Hacking with the untyped call-by-value lambda calculus

In this exercise, you have to implement some operations for Church encoding of lists. There are several ways to Church encode a list, among which Church encoding based on its right fold function is more popular. As an example, an empty list (`nil`) and the `cons` construct are represented as follows in this encoding:

```
nil = λc.  λn.  n
cons = λh.  λt.  λc.  λn.  c h (t c n)
```

As another example, a list of 3 elements x, y, z is encoded as:

```
λc.  λn.  c x (c y (c z n))
```

The complete list of predefined operations can be found in the appendix, and only these operations can be used in the exercise. Define the following operations on a list:

1. (2 points) The *map* function which applies the given function to each element of the given list.

2. (2 points) The *length* function which returns the size of the given list. The result should be in Church encoding.

3. (2 points) The *sum* function which returns the sum of all elements of the given list. Assume all elements and the result are Church encoded numbers.

4. (2 points) The *concat* function which concatenates two input lists.

5. (2 points) The *exists* function which checks if there is any element satisfying the given predicate. The given predicate and the result should be both in Church encoding.