Exercice 1 : Normal Forms (10 points)

Let $\mathcal{V}$ be the set of variables and Λ the set of λ-terms.

Let $\mathcal{N} \subset \Lambda$ be the set of normal forms ($\mathcal{N} = \{t \in \Lambda \mid \nexists t' \in \Lambda : t \to t'\}$).
We define inductively the subset $\mathcal{N}'$ of Λ:

$$(\textsc{Var})\ \frac{x \in \mathcal{V}}{x \in \mathcal{N}'} \qquad (\textsc{Abs})\ \frac{x \in \mathcal{V} \quad t \in \mathcal{N}'}{\lambda x.t \in \mathcal{N}'} \qquad (\textsc{App}_n)\ \frac{x \in \mathcal{V} \quad t_1 \in \mathcal{N}' \quad \cdots \quad t_n \in \mathcal{N}'}{x\ t_1 \cdots t_n \in \mathcal{N}'}\ n \in \mathbb{N}, n > 0$$

Show that $\mathcal{N}' = \mathcal{N}$.

<u>Hint:</u> Show that if $t \in \mathcal{N}$ then $t \in \mathcal{N}'$ by induction on $t \in \Lambda$ and that if $t \in \mathcal{N}'$ then $t \in \mathcal{N}$ by induction on the derivation $t \in \mathcal{N}'$.

Answer:
We first prove $\mathcal{N}' \subset \mathcal{N}$ and then prove $\mathcal{N} \subset \mathcal{N}'$:

- let $t \in \mathcal{N}'$ and we prove that $t \in \mathcal{N}$ by induction.
  - $t = x$ then clearly $t \in \mathcal{N}$ because variables are normal forms.
  - $t = \lambda x.t'$. By $\textsc{Abs}$ $t' \in \mathcal{N}'$ and by induction hypothesis $t' \in \mathcal{N}$. Then clearly t can't take any reduction steps so $t \in \mathcal{N}$
  - $t = x\ t_1 \ldots t_n$. By $\textsc{App}_n$ $x \in \mathcal{V}$ and $t_1 \ldots t_n \in \mathcal{N}'$. By induction hypothesis $t_1 \ldots t_n \in \mathcal{N}'$ are also in $\mathcal{N}$. Since none of the applications contains any redexes, t can't take a reduction step and therefore $t \in \mathcal{N}$
- let $t \in \mathcal{N}$ and we prove that $t \in \mathcal{N}'$ by induction on the structure of t
  - $t = x$ and by rule $\textsc{Var}$ $t \in \mathcal{N}'$
  - $t = \lambda x.t'$. Since $t \in \mathcal{N}$ we have that also t' can't contain any redexes, so $t' \in \mathcal{N}$. By induction hypothesis we have $t' \in \mathcal{N}'$ and by $\textsc{Abs}$ we have $t \in \mathcal{N}$.
  - $t = t_1\ t_2$. Since $t \in \mathcal{N}$ and is a normal form, we have that t_1, t_2 are also normal forms, therefore $t_1, t_2 \in \mathcal{N}$. By induction hypothesis we have that $t_1, t_2 \in \mathcal{N}'$:
    * $t_1 = x$, we have $t = x\ t_2$ and $t \in \mathcal{N}'$ by $\textsc{App}_1$.
    * $t_1 = \lambda x.t_1'$. Impossible, since then $t = \lambda x.t_1'\ t_2$ and that is a redex, but $t \in \mathcal{N}$.
    * $t_1 = x\ t_{11} \ldots t_{1n}$. We have $t = x\ t_{11} \ldots t_{1n}\ t_2$ and by $\textsc{App}_{n+1}$ $t \in \mathcal{N}'$.

Exercice 2 : Typed Arithmetic Expressions (10 points)

We first recall the syntax for arithmetic expressions (TAPL, p.91):

t ::=	**terms :**	v ::=	**values :**
$\mid$ $true$	*constant true*	$\mid$ $true$	*true value*
$\mid$ $false$	*constant false*	$\mid$ $false$	*false value*
$\mid$ $if\ t\ then\ t\ else\ t$	*condition*	$\mid$ nv	*numeric value*
$\mid$ 0	*constant zero*		
$\mid$ $succ\ t$	*successor*	nv ::=	**numeric values :**
$\mid$ $pred\ t$	*predecessor*	$\mid$ 0	*zero value*
$\mid$ $iszero\ t$	*zero test*	$\mid$ $succ\ nv$	*successor value*

and the evaluation rules for numbers (TAPL, p.41):

$$\text{(E-Succ)}\ \frac{t_1 \longrightarrow t_1'}{succ\ t_1 \longrightarrow succ\ t_1'}$$

$$\text{(E-PredZero)}\ pred\ 0 \longrightarrow 0 \qquad \text{(E-IsZeroZero)}\ iszero\ 0 \longrightarrow true$$

$$\text{(E-PredSucc)}\ pred\ (succ\ nv_1) \longrightarrow nv_1 \qquad \text{(E-IsZeroSucc)}\ iszero\ (succ\ nv_1) \longrightarrow false$$

$$\text{(E-Pred)}\ \frac{t_1 \longrightarrow t_1'}{pred\ t_1 \longrightarrow pred\ t_1'} \qquad \text{(E-IsZero)}\ \frac{t_1 \longrightarrow t_1'}{iszero\ t_1 \longrightarrow iszero\ t_1'}$$

Suppose we remove the E-PredZero rule.

Does progress still hold ? What about preservation ?

Change the definition of values in the modified language such that both progress and preservation hold. However, you are not allowed to reintroduce E-PredZero or to add another reduction rule for terms of the form $pred(x)$.

Answer:

Progress does not hold: $pred\ 0 : Nat$ is a well-typed term which is not a value and is stuck.

Preservation still holds because removing one evaluation rule just makes the original proof shorter (one less case to prove).

We change the syntax for numeric values:

nv	::=	pos	positive
	$\mid$	neg	negative
pos	::=	0	zero
	$\mid$	$succ\ pos$	successor
neg	::=	0	zero
	$\mid$	$pred\ neg$	predecessor

We need to add a rule to deal with successors of negative numbers, one for $iszero$ with negative numbers, and also update rules that worked on nv to work on positives or negatives.:

$$\text{E-SuccPred}\ \frac{}{succ\ (pred\ neg_1) \rightarrow neg_1} \qquad \text{E-PredSucc}\ \frac{}{pred\ (succ\ pos_1) \rightarrow pos_1}$$

$$\text{E-IsZeroPred}\ \frac{}{iszero\ (pred\ neg_1) \rightarrow false} \qquad \text{E-IsZeroSucc}\ \frac{}{iszero\ (succ\ pos_1) \rightarrow false}$$

Exercice 3 : Option Types (10 points)

We extend the syntax for the simply typed λ-calculus (TAPL, p.103) with option types in a similar way as in Scala, e.g.

```
(\o: option Nat. o match {
  case some x => x
  case none   => 0
}) some (succ 0)
```

The meaning of the above is that when o is an instance of *some*, the first branch is selected and x takes the value carried inside o. When o is an instance of *none*, the second branch is evaluated. As a rule of thumb, the evaluation rules should match Scala's behavior, like call-by-value evaluation order and the usual meaning for *match*. The language should also allow you to create values of both kinds.

Formalize this extension. Your solution should include a grammar extension (for terms, values and types), evaluation rules and typing rules for the new terms. Typing rules should peserve type safety and make it impossible to find two different types for the same term (uniqueness of types). (There is no need to prove these properties).

Answer:
We propose the following syntax:

Terms

t	::=	...	terms
	|	some t	some
	|	none as T	none: T
	|	t match "{" case some x => t case none => t "}"	match

Values

v	::=	...	
	|	none as T	none values
	|	some v	some values

Types

T	::=	...	
	|	Option T	option types

Evaluation rules:

$$\textsc{E-MatchSome}\ \frac{}{\textit{some } v \textit{ match } \{ \textit{ case some } x => t_1 \textit{ case none} => t_2\} \rightarrow [x \mapsto v]t_1}$$

$$\textsc{E-MatchNone}\ \frac{}{\textit{none as } T \textit{ match } \{ \textit{ case some } x => t_1 \textit{ case none} => t_2\} \rightarrow t_2}$$

$$\textsc{E-Match}\ \frac{t \rightarrow t'}{\begin{array}{c} t \textit{ match } \{ \textit{ case some } x => t_1 \textit{ case none} => t_2\} \rightarrow \\ t' \textit{ match } \{ \textit{ case some } x => t_1 \textit{ case none} => t_2\} \end{array}}$$

$$\textsc{E-Some}\ \frac{t \rightarrow t'}{\textit{some } t \rightarrow \textit{some } t'}$$

Typing rules:

$$\textsc{T-Some}\ \frac{\Gamma \vdash t : T}{\Gamma \vdash \mathit{some}\ t : \mathit{Option}\ T} \qquad \textsc{T-None}\ \frac{T = \mathit{Option}\ S}{\Gamma \vdash \mathit{none}\ \mathit{as}\ T : T}$$

$$\textsc{T-Match}\ \frac{\Gamma \vdash t : \mathit{Option}\ T \qquad \Gamma, x : T \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_1}{t\ \mathit{match}\ \{\ \mathit{case}\ \mathit{some}\ x => t_1\ \mathit{case}\ \mathit{none} => t_2\} : T_2}$$