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

Pierce explains Church encodings for booleans, numerals, and operations on them in (TAPL book s. 5.2 p. 58). We would like to define some more advanced functions using only the basic operations `scc`, `plus`, `prd`, `times`, `iszro`, `test`, `fix` and the constants. 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 non-negative integers:

1. The greater equal operation `geq` ($\geq$)

2. The greater than operation `gt` ($>$)

3. The modulo operation `mod` (e.g. `mod` $\mathtt{c}_{14}$ $\mathtt{c}_3$ behaves like $\mathtt{c}_2$)

4. The Ackermann function `ack` using the basic operations and operations defined in this exercise. The Ackermann function is defined as follows:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

2 Closed terms

We recall that a term `t` is closed if it contains no free variables. With that definition in mind, prove the following property for the call-by-value untyped lambda calculus (for reference provided in Appendix 1).

Theorem: If `t` is closed and $\texttt{t} \longrightarrow \texttt{t}'$, then $\texttt{t}'$ is closed as well.

3 Associativity of Addition

Suppose natural numbers are defined inductively as follows:

`n`	::=		**naturals** :
	|	`Z`	zero
	|	`S n`	successor

And addition is defined recursively below:

$$\begin{aligned} \texttt{Z} + \texttt{n} &= \texttt{n} && (1) \\ (\texttt{S m}) + \texttt{n} &= \texttt{m} + (\texttt{S n}) && (2) \end{aligned}$$

Prove that addition is associative: $a + (b + c) = (a + b) + c$ for all naturals a, b, c.

For reference: **Untyped lambda calculus**

The complete reference of the untyped lambda calculus with call-by-value semantics is:

$$
\begin{array}{rcll}
\mathtt{t} & ::= & & \textbf{terms}: \\
& | & \mathtt{x} & \text{variable} \\
& | & \lambda \mathtt{x.t} & \text{abstraction} \\
& | & \mathtt{t\ t} & \text{application}
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{v} & ::= & & \textbf{values}: \\
& | & \lambda \mathtt{x.t} & \text{abstraction}
\end{array}
$$

Small-step reduction rules:

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

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

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

For reference: **Predefined Lambda Terms**

Predefined lambda terms that can be used as-is in the first exercise:

$$
\begin{array}{lcl}
\texttt{unit} & = & \lambda\texttt{x.x} \\
 & & \\
\texttt{tru} & = & \lambda\texttt{t.}\lambda\texttt{f.t} \\
\texttt{fls} & = & \lambda\texttt{t.}\ \lambda\texttt{f.f} \\
\texttt{iszro} & = & \lambda\texttt{m.m}\ (\lambda\texttt{x.fls})\ \texttt{tru} \\
\texttt{test} & = & \lambda\texttt{b.}\lambda\texttt{t.}\lambda\texttt{f.b t f unit} \\
 & & \\
\texttt{pair} & = & \lambda\texttt{f.}\lambda\texttt{s.}\lambda\texttt{b.b f s} \\
\texttt{fst} & = & \lambda\texttt{p.p tru} \\
\texttt{snd} & = & \lambda\texttt{p.p fls} \\
 & & \\
\texttt{c}_0 & = & \lambda\texttt{s.}\lambda\texttt{z.z} \\
\texttt{c}_1 & = & \lambda\texttt{s.}\lambda\texttt{z.s z} \\
\texttt{scc} & = & \lambda\texttt{n.}\lambda\texttt{s.}\lambda\texttt{z.s}\ (\texttt{n s z}) \\
\texttt{plus} & = & \lambda\texttt{m.}\lambda\texttt{n.}\lambda\texttt{s.}\lambda\texttt{z.m s}\ (\texttt{n s z}) \\
\texttt{times} & = & \lambda\texttt{m.}\lambda\texttt{n.m}\ (\texttt{plus n})\ \texttt{c}_0 \\
 & & \\
\texttt{zz} & = & \texttt{pair c}_0\ \texttt{c}_0 \\
\texttt{ss} & = & \lambda\texttt{p.pair}\ (\texttt{snd p})\ (\texttt{scc}\ (\texttt{snd p})) \\
\texttt{prd} & = & \lambda\texttt{m.fst}\ (\texttt{m ss zz}) \\
 & & \\
\texttt{fix} & = & \lambda\texttt{f.}(\lambda\texttt{x.f}\ (\lambda\texttt{y.x x y}))\ (\lambda\texttt{x.f}\ (\lambda\texttt{y.x x y}))
\end{array}
$$