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 *gr* ($>$)

3. The modulo operation *mod* (e.g. $mod\ c_5\ c_3 = c_2$)

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 Uniqueness of terms after reductions

For this exercise assume we will be working with a simple language, defined by the following abstract syntax of terms:

$$s, t, v, w ::= E \mid U\ t \mid B\ t\ t$$

If that helps, you can understand that as isomorphic to the following algebraic datatype in `Scala`:

```
abstract class Base
case object Empty extends Base
case class Unary(i: Base) extends Base
case class Binary(i: Base, j: Base) extends Base
```

Let $\rightarrow^{\beta}$ be a reduction relation defined by the following computation rules:

$$B\ E\ t_2 \rightarrow^{\beta} t_2 \tag{β_1}$$

$$B\ t_1\ t_2 \rightarrow^{\beta} B\ t_2\ t_1 \tag{β_2}$$

and congruence rules:

$$\frac{t_1 \rightarrow^{\beta} t_1'}{U\ t_1 \rightarrow^{\beta} U\ t_1'} \tag{β_3}$$

$$\frac{t_1 \rightarrow^{\beta} t_1'}{B\ t_1\ t_2 \rightarrow^{\beta} B\ t_1'\ t_2} \tag{β_4}$$

$$\frac{t_2 \rightarrow^{\beta} t_2'}{B\ t_1\ t_2 \rightarrow^{\beta} B\ t_1\ t_2'} \tag{β_5}$$

Prove that for any terms s, t, v we have

$$s \rightarrow^{\beta} t\ \wedge\ s \rightarrow^{\beta} v\ \Rightarrow \exists w.(t \rightarrow^{\beta *} w\ \wedge\ v \rightarrow^{\beta *} w)$$

where $\rightarrow^{\beta *}$ refers to zero or more $\rightarrow^{\beta}$ reductions. Your solution has to have a clear explanation for each step in your proof.

Hint: We suggest proving the above statement using induction on the structure of s.

3 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 $t \longrightarrow t'$ then t' is closed as well.

Note: Remember to state clearly all the steps of your proof, including proofs of any lemmas that you use.

For reference: **Untyped lambda calculus**

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

$$
\begin{array}{rcll}
t & ::= & & \textbf{terms}: \\
& | & x & \textit{variable} \\
& | & \lambda x.\, t & \textit{abstraction} \\
& | & t\ t & \textit{application } (\textit{left assoc.}) \\
\\
v & ::= & & \textbf{values}: \\
& | & \lambda x.\, t & \textit{abstraction}
\end{array}
$$

Small-step reduction rules:

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \qquad \text{(E-App2)}$$

$$(\lambda\ x.\ t_1)\ v_2 \longrightarrow [x \mapsto v_2]\ t_1 \qquad \text{(E-AppAbs)}$$

For reference: **Predefined Lambda Terms**

Predefined lambda terms that can be used as-is in the exam

$$
\begin{array}{ll}
\texttt{unit} = & \lambda x.\ x \\
 & \\
\texttt{tru} = & \lambda t.\ \lambda f.\ t \\
\texttt{fls} = & \lambda t.\ \lambda f.\ f \\
\texttt{iszro} = & \lambda m.\ m\ (\lambda x.\ \texttt{fls})\ \texttt{tru} \\
\texttt{test} = & \lambda b.\ \lambda t.\ \lambda f.\ b\ t\ f\ \texttt{unit} \\
 & \\
\texttt{pair} = & \lambda f.\ \lambda s.\ \lambda b.\ b\ f\ s \\
\texttt{fst} = & \lambda p.\ p\ \texttt{tru} \\
\texttt{snd} = & \lambda p.\ p\ \texttt{fls} \\
 & \\
\texttt{c}_0 = & \lambda s.\ \lambda z.\ z \\
\texttt{c}_1 = & \lambda s.\ \lambda z.\ s\ z \\
\texttt{scc} = & \lambda n.\ \lambda s.\ \lambda z.\ s\ (n\ s\ z) \\
\texttt{plus} = & \lambda m.\ \lambda n.\ \lambda s.\ \lambda z.\ m\ s\ (n\ s\ z) \\
\texttt{times} = & \lambda m.\ \lambda n.\ m\ (\texttt{plus}\ n)\ c_0 \\
 & \\
\texttt{zz} = & \texttt{pair}\ c_0\ c_0 \\
\texttt{ss} = & \lambda p.\ \texttt{pair}\ (\texttt{snd}\ p)\ (\texttt{scc}\ (\texttt{snd}\ p)) \\
\texttt{prd} = & \lambda m.\ \texttt{fst}\ (m\ \texttt{ss}\ \texttt{zz})
\end{array}
$$