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` ($\geq$)

   ```
   geq = λx. λy. iszro (x prd y)
   ```

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

   ```
   gt = λx. λy. (iszro (y prd x)) fls tru
   ```

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

   Solution:

   ```
   mod =
     fix \me. \x. \y.
       test (geq x y)
       (\then. (me (y prd x) y))
       (\else. x)
   ```

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

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

   Solution:

   ```
   ack =
     fix \me. \x. \y.
       test (iszro x)
       (\then. (scc y))
       (\else.
         test (iszro y)
         (\then. (me (prd x) c1))
         (\else. (me (prd x) (me x (prd y)))))
   ```

2 Closed terms

We recall that a term $\mathtt{t}$ is closed if it contains no free variables, i.e., $FV(\mathtt{t}) = \emptyset$. 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 $\mathtt{t}$ is closed and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\mathtt{t}'$ is closed as well.

Solution:

We prove the theorem by induction on the structure of $\mathtt{t}$. We proceed by case analysis on the shape of $\mathtt{t}$.

1. Case $\mathtt{t} = \mathtt{x}$.
   There is no evaluation rule for $\mathtt{t} = \mathtt{x}$, i.e., there is no $\mathtt{t}'$ such that $\mathtt{x} \longrightarrow \mathtt{t}'$. This contradicts the hypothesis $\mathtt{t} \longrightarrow \mathtt{t}'$, therefore this case cannot happen.

2. Case $\mathtt{t} = \lambda\mathtt{x}.\mathtt{t}_1$.
   Similar.

3. Case $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$.
   By the definition of FV, we know that $FV(\mathtt{t}) = FV(\mathtt{t}_1) \cup FV(\mathtt{t}_2)$. Since $FV(\mathtt{t}) = \emptyset$, we conclude that $FV(\mathtt{t}_1) = \emptyset$ and $FV(\mathtt{t}_2) = \emptyset$.
   From the induction hypothesis on $\mathtt{t}_1$, we then get that if $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$, then $FV(\mathtt{t}_1') = \emptyset$. Similarly for $\mathtt{t}_2$.
   We proceed by case analysis of the last rule used to derive $\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}'$.

   (a) Case `E-APP1`:
   We know $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$ and $\mathtt{t}' = \mathtt{t}_1'\ \mathtt{t}_2$. From what we said above, we know that $FV(\mathtt{t}_1') = \emptyset$. Then by the definition of FV, we know that $FV(\mathtt{t}') = FV(\mathtt{t}_1') \cup FV(\mathtt{t}_2) = \emptyset \cup \emptyset = \emptyset$.

   (b) Case `E-APP2`:
   Similar.

   (c) Case `E-APPABS`:
   We know $\mathtt{t}_1 = \lambda\mathtt{x}.\mathtt{t}_{12}$, $\mathtt{t}_2 = \mathtt{v}_2$ and $\mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$. We also know from before that $FV(\lambda\mathtt{x}.\mathtt{t}_{12}) = \emptyset$ and $FV(\mathtt{v}_2) = \emptyset$. From the definition of FV, we know that $FV(\lambda\mathtt{x}.\mathtt{t}_{12}) = FV(\mathtt{t}_{12}) \setminus \{\mathtt{x}\}$, therefore $FV(\mathtt{t}_{12}) \subseteq \{\mathtt{x}\}$.
   We still need to prove that $FV([\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}) = \emptyset$. It *feels* like that should be true: we "start" from $\mathtt{t}_{12}$ which has at most $FV(\mathtt{t}_{12}) \subseteq \{\mathtt{x}\}$, but then we "replace" all the free $\mathtt{x}$'es by $\mathtt{v}_2$, which is closed. We *should* get a closed term as result. In general, it feels like if we have arbitrary terms $\mathtt{t}_1$ and $\mathtt{t}_2$ and $\mathtt{t}_2$ is closed, that we could get $FV([\mathtt{x} \mapsto \mathtt{t}_2]t_1) = FV(\mathtt{t}_1) \setminus \{x\}$. *If* that is true, then we are done. We prove that statement in a separate lemma. □

Lemma: If $\mathtt{t}_2$ is a closed term, i.e., $FV(\mathtt{t}_2) = \emptyset$, then $FV([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_1) = FV(\mathtt{t}_1) \setminus \{\mathtt{x}\}$.
Proof by induction on the structure of the lambda term $\mathtt{t}_1$. We proceed by case analysis of the shape of $\mathtt{t}_1$:

1. Case $\mathtt{t}_1 = \mathtt{y}$.
   If $\mathtt{x} = \mathtt{y}$, then $[\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_1 = [\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{y} = \mathtt{t}_2$. Therefore we have on the left $FV([\mathtt{x} \mapsto$

$\mathtt{t}_2]\mathtt{t}_1) = FV(\mathtt{t}_2) = \emptyset$ and on the right $FV(\mathtt{t}_1) \setminus \{\mathtt{x}\} = \{\mathtt{x}\} \setminus \{\mathtt{x}\} = \emptyset$, as desired.

If $\mathtt{x} \neq \mathtt{y}$, then $[\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_1 = \mathtt{t}_1 = \mathtt{y}$. Therefore on the left we have $FV([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_1) = FV(\mathtt{y}) = \{\mathtt{y}\}$, and on the right we have $FV(\mathtt{t}_1) \setminus \{\mathtt{x}\} = \{\mathtt{y}\} \setminus \{\mathtt{x}\} = \{\mathtt{y}\}$, as desired.

2. Case $\mathtt{t}_1 = \lambda\mathtt{y}.\mathtt{t}_{12}$.
Without loss of generality, we can assume $\mathtt{y} \neq \mathtt{x}$, as we can use α-conversion to avoid that situation.
By induction hypothesis on $\mathtt{t}_{12}$, $FV([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{12}) = FV(\mathtt{t}_{12}) \setminus \{\mathtt{x}\}$.
By definition of substitution, $[\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_1 = [\mathtt{x} \mapsto \mathtt{t}_2](\lambda\mathtt{y}.\mathtt{t}_{12}) = \lambda\mathtt{y}.([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{12})$.
Therefore, on the left we have $FV([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_1) = FV(\lambda\mathtt{y}.([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{12})) = FV([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{12}) \setminus \{\mathtt{y}\} = (FV(\mathtt{t}_{12}) \setminus \{\mathtt{x}\}) \setminus \{\mathtt{y}\}$, and on the right we have $FV(\mathtt{t}_1) \setminus \{\mathtt{x}\} = FV(\lambda\mathtt{y}.\mathtt{t}_{12}) \setminus \{\mathtt{x}\} = (FV(\mathtt{t}_{12}) \setminus \{\mathtt{y}\}) \setminus \{\mathtt{x}\}$, which are equal from elementary set properties.

3. Case $\mathtt{t}_1 = \mathtt{t}_{11}\ \mathtt{t}_{12}$.
By induction hypothesis on $\mathtt{t}_{11}$, $FV([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{11}) = FV(\mathtt{t}_{11}) \setminus \{\mathtt{x}\}$. Similarly for $\mathtt{t}_{12}$.
By definition of substitution, $[\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_1 = [\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{11}\ \ [\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{12}$.
Therefore on the left we have $FV([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_1) = FV([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{11}) \cup FV([\mathtt{x} \mapsto \mathtt{t}_2]\mathtt{t}_{12}) = (FV(\mathtt{t}_{11}) \setminus \{\mathtt{x}\}) \cup (FV(\mathtt{t}_{12}) \setminus \{\mathtt{x}\})$, and on the right we have $FV(\mathtt{t}_1) \setminus \{\mathtt{x}\} = (FV(\mathtt{t}_{11}) \cup FV(\mathtt{t}_{12})) \setminus \{\mathtt{x}\}$, which are equal from elementary set properties. □

3 Associativity of Addition

Suppose natural numbers are defined inductively as follows:

$$\begin{array}{rcll} \texttt{n} & ::= & & \textbf{naturals}: \\ & | & \texttt{Z} & \text{zero} \\ & | & \texttt{S n} & \text{successor} \end{array}$$

And addition is defined recursively below:

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

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

3.1 Solution

Proof: by induction on the structure of a (why not b or c? because it works for a...). We proceed by case analysis on the shape of a:

- Case $a = \texttt{Z}$.
  $\texttt{Z} + (b + c) \overset{(1)}{=} b + c \overset{(1)}{=} (\texttt{Z} + b) + c$.
- Case $a = \texttt{S}\, m$.
  $a + (b + c) = \texttt{S}\, m + (b + c) \overset{(2)}{=} m + \texttt{S}(b + c)$
  $(a + b) + c = (\texttt{S}\, m + b) + c \overset{(2)}{=} (m + \texttt{S}\, b) + c \overset{I.H.}{=} m + (\texttt{S}\, b + c)$
  These are equal if $\texttt{S}(b + c) = \texttt{S}\, b + c$. Our intuition about mathematical addition suggests that it *should* be true, but we need to prove it. We extract it in a separate lemma. □

Lemma: for all numerals b, c, $\texttt{S}(b + c) = \texttt{S}\, b + c$.
Proof: by induction on the structure of b. We proceed by case analysis on the shape of b:

- Case $b = \texttt{Z}$.
  $S(\texttt{Z} + c) \overset{(1)}{=} \texttt{S}\, c \overset{(1)}{=} \texttt{Z} + \texttt{S}\, c \overset{(2)}{=} \texttt{S}\, \texttt{Z} + c$.
- Case $b = \texttt{S}\, m$.
  $\texttt{S}(b + c) = \texttt{S}(\texttt{S}\, m + c) \overset{(2)}{=} \texttt{S}(m + \texttt{S}\, c) \overset{I.H.}{=} \texttt{S}\, m + \texttt{S}\, c \overset{(2)}{=} \texttt{S}(\texttt{S}\, m) + c = \texttt{S}\, b + c$. □

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:

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