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 *gr* ($>$)

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

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

```
mod =
  fix λme. λx. λy.
    test (geq x y)
    (λthen. (me (y prd x) y))
    (λelse. x)
```

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

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

```
ack =
  fix λme. λx. λy.
    test (iszro x)
    (λthen. (scc y))
    (λelse.
      test (iszro y)
      (λthen. (me (prd x) c1))
      (λelse. (me (prd x) (me x (prd y)))))
```

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 $\to^\beta$ be a reduction relation defined by the following computation rules:

$$B\ E\ t_2 \to^\beta t_2 \tag{β_1}$$

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

and congruence rules:

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

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

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

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

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

where $\to^{\beta *}$ refers to zero or more $\to^\beta$ 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.

Solution: We prove the thesis using induction on the structure of s, and case analysis on the possible final rules of the derivations of $s \to^\beta t$ and $s \to^\beta v$; we call those rules respectively β_t and β_v.

To follow the solution, remember to try doing the proof yourself in detail, following the text as hints, as most proofs are written to be read this way.

- Case $s = E$. No reduction rule applies so this case is impossible. Done.
- Case $s = U\ s_1$. Hence, both $\beta_t = \beta_v = \beta_3$. At a high level, in this case we have the same subterm s_1 reduces to two different terms t_1 and v_1, by IH they both reduce to w_1, and by congruence then $t = U\ t_1$ and $v = U\ v_1$ both reduce to $w = U\ w_1$.

- Case $s = B\ s_1\ s_2$. All reduction rules but β_3 can apply to s, so we have in principle 16 cases to consider, but we can group them together. First, once we consider for instance $\beta_t = \beta_1$ and $\beta_v = \beta_2$, we can use the same strategy but swapping t and v for $\beta_t = \beta_2$ and $\beta_v = \beta_1$, because the problem statement is "symmetric". Second, using other tricks, we can group these cases in 6 groups of analogous ones.

  You can draw a 4×4 grid to check we cover all cases.

  1. (2 cases) If $\beta_t = \beta_v$ and both are β_4 or β_5, we proceed as in the $s = U\ s_1$ case. Take the β_4 case: t and v are of form $B\ t_1\ t_2$ and $B\ v_1\ v_2$, $s_1 \to^{\beta} t_1$ and $s_1 \to^{\beta} v_1$. By IH, there exists w_1 such that $t_1 \to^{\beta*} w_1$ and $v_1 \to^{\beta*} w_1$, so we pick $w = B\ w_1\ w_2$ and we have $t \to^{\beta*} w \wedge v \to^{\beta*} w$.
  2. (7 cases) If $\beta_t = \beta_2$ or $\beta_v = \beta_2$. Focus on $\beta_t = \beta_2$: then $t = B\ t_2\ t_1$. We see that $t \to^{\beta_2} s \to^{\beta} v$ so $t \to^{\beta*} v$, while $v \to^{\beta*} v$ in zero steps. So we pick $w = v$.
  3. (1 case) If $\beta_t = \beta_v = \beta_1$, then $t = v = s_2$, and $s_2 \to^{\beta*} s_2$ in 0 steps. So we pick $w = s_2$.
  4. (2 cases) If $\beta_t = \beta_1$ and $\beta_v = \beta_4$ (or symmetrically): here $t = s_2$ and $v = B\ s_1'\ s_2$ with $E \to^{\beta} s_1'$ — but E does not reduce! So this case is impossible.
  5. (2 cases) If $\beta_t = \beta_1$ and $\beta_v = \beta_5$ (or symmetrically): here $t = s_2$ and $v = B\ s_1\ s_2'$ with $s_2 \to^{\beta} s_2'$. Then $v \to^{\beta_1} s_2'$, and we pick $w = s_2'$.
  6. (2 cases) If $\beta_t = \beta_4$ and $\beta_v = \beta_5$ (or symmetrically): here $t = B\ s_1'\ s_2$ and $v = B\ s_1\ s_2'$. Then $w = B\ s_1'\ s_2'$, $t \to^{\beta_5} w$ and $v \to^{\beta_4} w$.

q.e.d

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.

Solution:
We prove the Theorem by induction on the structure of t.

1. Let t be a variable t. The solution is immediate since t is closed and the case cannot occur.

2. Let t be an abstraction $\lambda x.t_1$. Since $\lambda x.t_1 \not\longrightarrow$, the solution is immediate.

3. Let t be an application $t_1\ t_2$. Then we can have three different cases, based on the used reduction rule for $t \longrightarrow t'$:

   (a) `E-APP1` - then $t_1 \longrightarrow t_1'$. As t is closed, then both t_1 and t_2 are closed as well. By induction t_1' is closed as well. Therefore $t_1' t_2$ is closed as well.

   (b) `E-APP2` - then $t_2 \longrightarrow t_2'$. As t is closed, then both v_1 and t_2 are closed as well. By induction t_2' is closed as well. Therefore $v_1 t_2'$ is closed as well.

   (c) `E-APPABS` - then $(\lambda\ x.t_{12})\ v_2 \ \longrightarrow\ [x \ \rightarrow\ v_2]\ t_{12}$. As t is closed, then both $\lambda x.t_{12}$ and v_2 are closed as well. From $FV(\lambda x.t_{12}) = \emptyset$, we know $FV(t_{12}) \subseteq \{x\}$, otherwise it's easy to prove $FV(\lambda x.t_{12}) \neq \emptyset$ by the definition of FV, a contradiction. Now, we need another lemma which says that substitution with a closed term removes a free variable. Thus, $FV([x \ \rightarrow\ v_2]t_{12}) \ =\ FV(t_{12}) \ \setminus\ x \ =\ \emptyset$. Done.

Lemma: If t_2 is a closed term, $FV([x \rightarrow t_2]t_1) = FV(t_1) \setminus x$.
Proof by induction on the structure of the lambda term t_1.

1. Case $t_1 = y$.
   If $x = y$, we have $FV([x \rightarrow t_2]t_1) = FV([x \rightarrow t_2]y) = FV([x \rightarrow t_2]x) = FV(t_2) = \emptyset = FV(x) \setminus x = FV(y) \setminus x = FV(t_1) \setminus x$.

   If $x \neq y$, we have $FV(t_1) \setminus x = FV(y) \setminus x = \{y\}$, and substitution results in y.

2. Case $t_1 = \lambda y.t_1'$.
   If $x = y$, then the result is immediate, since x is bound in the abstraction, i.e. $x \notin FV(t_1)$ and $[x \rightarrow t_2]t_1 = t_1$, we have $FV([x \rightarrow t_2]t_1) = FV(t_1) = FV(t_1) \setminus x$.

   If $x \neq y$, then by induction hypothesis $FV([x \rightarrow t_2]t_1') = FV(t_2') \setminus x$. Formally, $FV([x \rightarrow t_2]t_1) = FV(\lambda y.[x \rightarrow t_2]t_1') = FV([x \rightarrow t_2]t_1') \setminus y = FV(t_1') \setminus x \setminus y = FV(t_1') \setminus y \setminus x = FV(\lambda y.t_1') \setminus x = FV(t_1) \setminus x$.

3. Case $t_1 = t_1'\ t_2'$.
   By induction hypothesis, $FV([x \rightarrow t_2]t_1') = FV(t_1') \setminus x$ and $FV([x \rightarrow t_2]t_2') = FV(t_2') \setminus x$.

   Therefore $FV([x \rightarrow t_2](t_1'\ t_2')) = FV([x \rightarrow t_2]t_1') \cup FV([x \rightarrow t_2]t_2') = (FV(t_1') \setminus x) \cup (FV(t_2') \setminus x) = FV(t_1') \cup FV(t_2') \setminus x = FV(t_1'\ t_2') \setminus x = FV(t_1) \setminus x$.

For reference: **Untyped lambda calculus**

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

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

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

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