CSL 665: Introduction (Logic and Functional) Programming
I semester 2003-04

Minor I Mon 08 Sep 2003 V 344 8:00-??:?? Max Marks 50 **Solutions**

Notational conventions.

- $\triangleq$ denotes a definition of a name by a term (usually a λ-term)
- $\equiv$ denotes "syntactic identity". It is very often used to replace a name by its definition
- $\equiv_\alpha$ denotes α equivalence
- $\rightarrow_\beta$ denotes a single β-reduction
- $\rightarrow^n_\beta$ denotes exactly n steps of β-reduction
- $\rightarrow^*_\beta$ denotes zero or more steps of β-reduction
- $=_\beta$ denotes β-equality
- $\lambda x\ y\ z[\ldots]$ denotes $\lambda x[\lambda\ y[\lambda\ z[\ldots]]]$, but we will avoid using too many brackets whenever there is no confusion.
- As usual application associates to the left, so $(x\ y\ z)$ denotes $((x\ y)\ z)$.

---

1. Let $\mathbf{S} \triangleq \lambda x\lambda y\lambda z[((x\ z)\ (y\ z))]$ and $\mathbf{K} \triangleq \lambda x\lambda y[x]$. Prove that application is not associative by showing that
$$((\mathbf{S}\ \mathbf{K})\ \mathbf{K}) \neq_\beta (\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))$$
10 marks

Solution.

$$\begin{array}{ll}
 & ((\mathbf{S}\ \mathbf{K})\ \mathbf{K}) \\
\equiv & ((\lambda x\ y\ z[((x\ z)\ (y\ z))]\ \mathbf{K})\ \mathbf{K}) \\
\rightarrow^2_\beta & \lambda z[((\mathbf{K}\ z)\ (\mathbf{K}\ z))] \\
\equiv & \lambda z[((\lambda x\ y[x]\ z)\ (\lambda x\ y[x]\ z))] \\
\rightarrow^2_\beta & \lambda z[(\lambda y[z]\ \lambda y[z])] \\
\rightarrow_\beta & \lambda z[z] \qquad\qquad \ldots(*)
\end{array}$$

$$\begin{array}{ll}
 & (\mathbf{S}\ (\mathbf{K}\ \mathbf{K})) \\
\equiv & (\mathbf{S}\ (\lambda x\ y[x]\ \mathbf{K})) \\
\rightarrow_\beta & (\mathbf{S}\ \lambda y[\mathbf{K}]) \\
\equiv_\alpha & (\mathbf{S}\ \lambda u[\mathbf{K}]) \\
\equiv & (\lambda x\ y\ z[((x\ z)\ (y\ z))]\ \lambda u[\mathbf{K}]) \\
\rightarrow_\beta & \lambda\ y\ z[((\lambda u[\mathbf{K}]\ z)\ (y\ z))] \\
\rightarrow_\beta & \lambda\ y\ z[(\mathbf{K}\ (y\ z))] \\
\equiv & \lambda\ y\ z[(\lambda x\ y[x]\ (y\ z))] \\
\equiv_\alpha & \lambda\ y\ z[(\lambda x\ w[x]\ (y\ z))] \\
\rightarrow_\beta & \lambda\ y\ z\ w[(y\ z)] \qquad\qquad \ldots(**)
\end{array}$$

It is clear from (*) and (**) that $((\mathbf{S}\ \mathbf{K})\ \mathbf{K}) \neq_\beta (\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))$.

2. In all of mathematics, function composition denoted by the infix operator $\circ$ is *associative*. That is, if $f : A \to B$, $g : B \to C$ and $h : C \to D$ are any three functions, then the composition $f \circ g : A \to C$ is defined as the function $(f \circ g)(a) = g(f(a))$. It is then easy to see that $\circ$ is associative, i.e.
$$(f \circ g) \circ h = f \circ (g \circ h)$$
Define a *lambda*-expression called **compose** and prove that it is associative. 12 marks

<u>*Solution*</u> Since compose may be applied only to functions and not to mere values, the appropriate definition which captures the definition $(f \circ g)(a) = g(f(a))$ is the λ-expression

$$\mathbf{compose} \triangleq \lambda x\ y\ z[(y\ (x\ z))]$$

which when applied to any two functions yields new functions as follows:

$$\begin{array}{lcllll}
\mathbf{fg} & \triangleq & ((\mathbf{compose}\ f)\ g) & =_\beta & \lambda z[(g\ (f\ z))] & \ldots(1) \\
\mathbf{gh} & \triangleq & ((\mathbf{compose}\ g)\ h) & =_\beta & \lambda z[(h\ (g\ z))] & \ldots(2)
\end{array}$$

To prove that **compose** is associative we need to show that

$$(\mathbf{compose}\ f\ (\mathbf{compose}\ g\ h)) =_\beta (\mathbf{compose}\ (\mathbf{compose}\ f\ g)\ h))$$

$$\begin{array}{lll}
 & (\mathbf{compose}\ (\mathbf{compose}\ f\ g)\ h) & \\
=_\beta & (\mathbf{compose}\ \mathbf{fg}\ h) & by(1) \\
\equiv & (\lambda x\ y\ z[(y\ (x\ z))]\ \mathbf{fg}\ h) & \\
\to_\beta^2 & \lambda z[(h\ (\mathbf{fg}\ z))] & \\
\equiv_\alpha & \lambda z[(h\ (\lambda u[(g\ (f\ u))]\ z))] & by(1) \\
\to_\beta & \lambda z[(h\ (g\ (f\ z)))] & \ldots(3)
\end{array}$$

$$\begin{array}{lll}
 & (\mathbf{compose}\ f\ (\mathbf{compose}\ g\ h)) & \\
=_\beta & (\mathbf{compose}\ f\ \mathbf{gh}) & by(2) \\
\equiv & ((\lambda x\ y\ z[(y\ (x\ z))]\ f)\ \mathbf{gh}) & \\
\to_\beta^2 & \lambda z[(\mathbf{gh}\ (f\ z))] & \\
\equiv_\alpha & \lambda z[(\lambda u[(h\ (g\ u))]\ (f\ z))] & by(2) \\
\to_\beta & \lambda z[(h\ (g\ (f\ z)))] & \ldots(4)
\end{array}$$

3. How will you construct binary trees in the untyped λ-calculus? Define the following λ-terms in the untyped λ-calculus to express the construction and the deconstruction of trees[1].

empty. An expression for the empty binary tree.

tree. An expression that takes a λ-term r denoting the root and two λ-expressions u, v (denoting left and right subtrees) and $(\mathbf{tree}\ r\ u\ v)$ yields a term denoting a non-empty binary tree.

Deconstruct. Let t be any λ-term denoting a non-empty tree. Then

root. $(\mathbf{root}\ t)$ yields the root of the tree t.
left. $(\mathbf{left}\ t)$ yields the left subtree, and
right. $(\mathbf{right}\ t)$ yields the right sub-tree.

Verify. Finally verify that for any λ-terms r, u, v, the following equations hold.

$$\begin{array}{lcl}
(\mathbf{root}\ (\mathbf{tree}\ r\ u\ v)) & =_\beta & r \\
(\mathbf{left}\ (\mathbf{tree}\ r\ u\ v)) & =_\beta & u \\
(\mathbf{right}\ (\mathbf{tree}\ r\ u\ v)) & =_\beta & v
\end{array}$$

15 marks

<u>*Solution*</u>. We may represent trees as 3-tuples of lambda expressions where the components represent the root node, the left subtree and the right sub-tree respectively. However we need

[1]We will assume as Gordon has done in his notes that application associates to the left. Hence $(L\ M\ N)$ will denote $((L\ M)\ N)$.

to represent the empty subtree too. Hence we use a 4-tuple in which the first component is true for a nonempty tree and use **false** as denoting the empty subtree. We then verify that our constructions are such that they can also be deconstructed. But that is easy since we know how to deconstruct tuples and obtain their individual components. So here goes.

First let's define the auxiliary functions needed:

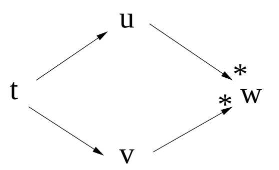
$$
\begin{array}{lcl}
\textbf{true} & \triangleq & \lambda x\ y[x] \\
\textbf{pair} & \triangleq & \lambda x\ y\ z[(z\ x\ y)] \\
\textbf{fst} & \triangleq & \lambda x[(x\ \textbf{true})] \\
\textbf{fourtuple} & \triangleq & \lambda x\ y\ z\ w[(\textbf{pair}\ x\ (\textbf{pair}\ y\ (\textbf{pair}\ z\ w)))]
\end{array}
\qquad
\begin{array}{lcl}
\textbf{false} & \triangleq & \lambda x\ y[y] \\
\\
\textbf{snd} & \triangleq & \lambda x[(x\ \textbf{false})] \\
\\
\end{array}
$$

Now comes the real masala:

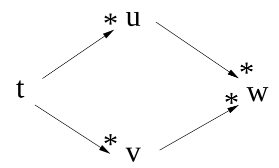
$$
\begin{array}{lcl}
\textbf{empty} & \triangleq & \textbf{false} \\
\textbf{tree} & \triangleq & \lambda r\ u\ v[(\textbf{fourtuple}\ \textbf{true}\ r\ u\ v)] \\
\textbf{root} & \triangleq & \lambda t[(\textbf{fst}\ (\textbf{snd}\ t))] \\
\textbf{left} & \triangleq & \lambda t[(\textbf{fst}\ (\textbf{snd}\ (\textbf{snd}\ t)))] \\
\textbf{right} & \triangleq & \lambda t[(\textbf{snd}\ (\textbf{snd}\ (\textbf{snd}\ t)))]
\end{array}
$$

We leave it to the reader to check that these definitions may be used to verify the given equations.

4. Let Σ be a 1-sorted signature (with at least one constant symbol) and let T_Σ be the set of all Σ-terms. Let $\longrightarrow \subseteq T_\Sigma \times T_\Sigma$ be a rewriting relation with N_Σ the set of normal forms. Assume that T_Σ satisfies the following properties.



Weak diamond



Strong Diamond

(a) Every term in T_Σ may be reduced to a normal form

(b) $\longrightarrow$ satisfies the following *weak diamond property*: For any three terms $t, u, v \in T_\Sigma$, $t \longrightarrow u$ and $t \longrightarrow v$ implies there exists a term $w \in T_\Sigma$ such that $u \longrightarrow^* w$ and $v \longrightarrow^* w$. See figure 4

Then prove that it satisfies the *strong diamond property*: For any three terms $t, u, v \in T_\Sigma$, $t \longrightarrow^* u$ and $t \longrightarrow^* v$ implies there exists a term $w \in T_\Sigma$ such that $u \longrightarrow^* w$ and $v \longrightarrow^* w$. See figure 4

13 marks