COMPSCI 3MI3

Assignment 5

1 Solution Set

1.1 Q1

We can encode the *Fib* function as given in the assignment with a λ expression in our λ-Calculus as follows:

$$\texttt{g = }\lambda\texttt{fib}.\lambda n.\,\texttt{if iszero}\,(\texttt{pred}\,n)\,\texttt{then}\ n\ \texttt{else fib}\,(\texttt{pred}\,n)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,n))$$
$$\texttt{fix = }\lambda f.(\lambda x.f\ (\lambda y.\ x\ x\ y))\ (\lambda x.f\ (\lambda y.x\ x\ y))$$

$$\texttt{fib} = \texttt{fix g}$$

We can now use our function to compute the 4^{th} number in the Fibonacci sequence which corresponds to $n = 3$.

<u>`fib 3`:</u>

$$\begin{array}{rl}
 & \texttt{fix g 3} \\
\rightarrow & (\lambda f.(\lambda x.f(\lambda y.x\,x\,y))(\lambda x.f(\lambda y.x\,x\,y)))\ \texttt{g 3} \\
\rightarrow & (\lambda x.\texttt{g}\ (\lambda y.x\,x\,y))(\lambda x.\texttt{g}\ (\lambda y.x\,x\,y))\ 3 \\
\texttt{setting} & h = \lambda x.\texttt{g}(\lambda y.x\,x\,y) \\
\texttt{yields} & (\lambda x.\texttt{g}\ (\lambda y.x\,x\,y))\ h\ 3 \\
\rightarrow & \texttt{g}\ (\lambda y.h\,h\,y)\ 3 \\
\texttt{setting} & \texttt{fib} = \lambda y.h\,h\,y \\
\texttt{yields} & \texttt{g}\,\texttt{fib}\,3 \\
\rightarrow & (\lambda\texttt{fib}.\lambda n.\,\texttt{if iszero}\,(\texttt{pred}\,n)\,\texttt{then}\ n\ \texttt{else fib}\,(\texttt{pred}\,n)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,n)))\,\texttt{fib}\,3 \\
\rightarrow\rightarrow & \texttt{if iszero}\,(\texttt{pred}\,3)\,\texttt{then}\ 3\ \texttt{else fib}\,(\texttt{pred}\,3)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,3)) \\
\rightarrow\rightarrow & \texttt{if false then}\ 3\ \texttt{else fib}\,(\texttt{pred}\,3)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,3)) \\
\rightarrow\rightarrow\rightarrow\rightarrow & \texttt{fib}\,2 + \texttt{fib}\,1 \\
\rightarrow & (\lambda y.h\,h\,y)\ 2 + \texttt{fib}\,1
\end{array}$$

$$
\begin{array}{rl}
\rightarrow & (h\,h\,2) + \texttt{fib}\,1 \\
\rightarrow & ((\lambda x.\mathtt{g}\,(\lambda y.x\,x\,y))\,h\,2) + \texttt{fib}\,1 \\
\rightarrow & (\mathtt{g}\,(\lambda y.h\,h\,y)\,2) + \texttt{fib}\,1 \\
\rightarrow & (\mathtt{g}\,\texttt{fib}\,2) + \texttt{fib}\,1 \\
\rightarrow\rightarrow\rightarrow & (\texttt{if iszero}\,(\texttt{pred}\,2)\,\texttt{then}\ 2\ \texttt{else fib}\,(\texttt{pred}\,2)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,2))) + \texttt{fib}\,1 \\
\rightarrow\rightarrow\rightarrow\rightarrow & (\texttt{fib}\,(\texttt{pred}\,2)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,2))) + \texttt{fib}\,1 \\
\rightarrow\rightarrow\rightarrow & \texttt{fib}\,1 + \texttt{fib}\,0 + \texttt{fib}\,1 \\
\rightarrow & (\lambda y.h\,h\,y)\,1 + \texttt{fib}\,0 + \texttt{fib}\,1 \\
\rightarrow & (h\,h\,1) + \texttt{fib}\,0 + \texttt{fib}\,1 \\
\rightarrow & ((\lambda x.\mathtt{g}\,(\lambda y.x\,x\,y))\,h\,1) + \texttt{fib}\,0 + \texttt{fib}\,1 \\
\rightarrow & (\mathtt{g}\,\texttt{fib}\,1) + \texttt{fib}\,0 + \texttt{fib}\,1 \\
\rightarrow\rightarrow\rightarrow & (\texttt{if iszero}\,(\texttt{pred}\,1)\,\texttt{then}\ 1\ \texttt{else fib}\,(\texttt{pred}\,1)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,1))) + \texttt{fib}\,0 + \texttt{fib}\,1 \\
\rightarrow\rightarrow & (\texttt{if true}\,\texttt{then}\ 1\ \texttt{else fib}\,(\texttt{pred}\,1)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,1))) + \texttt{fib}\,0 + \texttt{fib}\,1 \\
\rightarrow & 1 + \texttt{fib}\,0 + \texttt{fib}\,1 \\
\rightarrow & 1 + (\lambda y.h\,h\,y)\,0 + \texttt{fib}\,1 \\
\rightarrow\rightarrow & 1 + ((\lambda x.\mathtt{g}\,(\lambda y.x\,x\,y))\,h\,0) + \texttt{fib}\,1 \\
\rightarrow & 1 + (\mathtt{g}\,\texttt{fib}\,0) + \texttt{fib}\,1 \\
\rightarrow\rightarrow\rightarrow & 1 + (\texttt{if iszero}\,(\texttt{pred}\,0)\,\texttt{then}\ 0\ \texttt{else fib}\,(\texttt{pred}\,0)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,0))) + \texttt{fib}\,1 \\
\rightarrow\rightarrow\rightarrow & 1 + 0 + \texttt{fib}\,1 \\
\rightarrow & 1 + \texttt{fib}\,1 \\
\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow & 1 + (\mathtt{g}\,\texttt{fib}\,1) \\
\rightarrow\rightarrow\rightarrow & 1 + (\texttt{if iszero}\,(\texttt{pred}\,1)\,\texttt{then}\ 1\ \texttt{else fib}\,(\texttt{pred}\,1)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,1))) \\
\rightarrow & 1 + (\texttt{if iszero}\,0\,\texttt{then}\ 1\ \texttt{else fib}\,(\texttt{pred}\,1)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,1))) \\
\rightarrow & 1 + (\texttt{if true}\,\texttt{then}\ 1\ \texttt{else fib}\,(\texttt{pred}\,1)\ +\ \texttt{fib}\,(\texttt{pred}\,(\texttt{pred}\,1))) \\
\rightarrow & 1 + 1 \\
\rightarrow & 2 \\
\not\rightarrow &
\end{array}
$$

By our derivation we see that `fib` 3 is equal to 2. Thus the fourth element in our Fibonacci sequence is 2.

1.2 Q2

We will prove the call-by-value evaluation strategy of λ-Calculus is determinate. Thus, any term t in λ-Calculus should satisfy the property

$$t \rightarrow t' \wedge t \rightarrow t'' \implies t' = t''$$

We will we prove this property holds by induction on the derivation $t \rightarrow t'$.

We must first define what we consider a normal form in our call-by-value strategy for λ-Calculus. We say that any term t is in normal form if t is a single abstraction of the form $\lambda x.\, t_1$ or any number of free variables. Thus we know that values as defined in the operational semantics of the call-by-value strategy are terms in normal form.

Base case. The base case denotes the final derivation where $t \to t'$ will yield a term in normal form that cannot be further evaluated. It is trivial to see that the only possible derivation that can yield a term in normal form is E-AppAbs, thus our final derivation is also determinate.
Therefore our base case holds.

Induction step. We assume that all sub-derivations of $t \to t'$, the above property holds, i.e. all sub-derivations are deterministic. We will prove that for all possible derivations $t \to t'$ the property holds. The derivation $t \to t'$ must be one of the following:

Case: E-App1
If E-App1 was the last derivation, we know that t must be of the form $t_1\, t_2$ with t_1,t_2 terms in λ-Calculus. We know that via E-App1 that $t' = t_1'\, t_2$ and have the permise there exists some sub-derivation $t_1 \to t_1'$.
We know we cannot apply E-App2 to t since E-App2 would require t_1 be a value, however values are in normal form and cannot be further evaluated which contradicts the premise of $t_1 \to t_1'$. Similarly, we cannot apply E-AppAbs to t since it would require t_1 to be of the form $\lambda x.\, t_i$ which is normal, and contradicts the premise $t_1 \to t_1'$. Therefore the only rule we can apply is E-App1.
In order to show $t' = t''$ we need to show also that $t_1' = t_1''$. By E-App1 we know that $t'' = t_1''\ t_2$ and has the premise $t_1 \to t_1''$. By our induction hypothesis we know that our property holds for all sub-derivations of $t \to t'$. Thus we know $t_1 \to t_1' \wedge t_1 \to t_1'' \implies t_1' = t_1''$. Since we have premises $t_1 \to t_1'$ and $t_1 \to t_1''$ we can conclude $t_1' = t_1''$.
Therefore our property holds for the E-App1 case.

Case: E-App2
We can prove the E-App2 case similar to E-App1. If E-App2 was the last derivation, we know that t must be of the form $t_1\, t_2$ with t_1,t_2 terms in λ-Calculus. We know that via E-App2 that $t' = t_1\, t_2'$ and we have the permise there exists some sub-derivation $t_2 \to t_2'$.
We know we cannot apply E-App1 to t since E-App1 would require v_1 be a term that can be further evaluated, however values are in normal form and cannot be further evaluated. Similarly, we cannot apply E-AppAbs to t since it would require t_2 to be a value, however values are in normal form and cannot be further evaluated, which contradicts the premise $t_2 \to t_2'$. Therefore the only rule we can apply is E-App2.
In order to show $t' = t''$ we need to show also that $t_2' = t_2''$. By E-App2 we know that $t'' = t_1\ t_2''$ and has the premise $t_2 \to t_2''$. By our induction hypothesis we know that our property holds for all sub-derivations of $t \to t'$. Thus we know $t_2 \to t_2' \wedge t_2 \to t_2'' \implies t_2' = t_2''$. Since we have premises $t_2 \to t_2'$ and $t_2 \to t_2''$ we can conclude $t_2' = t_2''$.
Therefore our property holds for the E-App2 case.

Case: E-AppAbs
Lastly we prove the E-AppAbs case. If E-AppAbs was the last derivation, we know t must be of the form $\lambda x.\, t_1$. We know we cannot apply E-App1 to t since $\lambda x.\, t_1$ is in normal form which cannot be evaluated further, contradicting E-App1's requirement that there must exist a sub-derivation $t_1 \to t_1'$. Similarly, we know we cannot E-App2 to t since v_2 is a value, which is normal form and cannot be further evaluated, contradicting E-App2's requirements that must exist a sub-derviation $t_2 \to t_2'$. Therefore our property holds for the E-AppAbs case.

We have shown that for all possible derivations $t \to t'$ our property holds. Therefore our induction step holds.

Thus, we've shown the call-by-value evaluation strategy of λ-Calculus is determinate.

1.3 Q3

We will prove that the Termination property does not hold for λ-Calculus. If we were to prove the termination property holds for our λ-Calculus we would need to show that any term in λ-Calculus can eventually be evaluated to a normal form, i.e. for any term in our λ-Calculus there exists a series of evaluation steps of finite length to evaluate the term to a normal form. Formally we prove

$$\forall\, t \in \mathcal{T}\; \exists\, t' \in \mathcal{N} \mid t \to^* t'$$

with $\mathcal{N}$ the set of terms in normal form and $\mathcal{T}$ the set of terms in our λ-Calculus.
Thus to prove the termination property does not hold for λ-Calculus it suffices to provide a counterexample, a term q in our λ-Calculus, for which we can construct an infinite series of evaluation steps $q \to q_1 \to q_2 \to ...$ for which all q_i are not in normal form.
We define our counterexample q as

$$q = (\lambda x.\, x\, x)\, (\lambda x.\, x\, x)$$

When you β-reduce q by resolving the application you obtain the same term

$$(\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \to (\lambda x.\, x\, x)\, (\lambda x.\, x\, x)$$

Thus, regardless of the evaluation strategy used q can only be evaluated to itself with $q \to q$. We can then construct an infinite evaluation chain of the form $q \to q \to q \to ...$ where q is not in normal form. Thus q is a counterexample that refutes the termination property for λ-Calculus.
Therefore we've proved that the Termination property does not hold for λ-Calculus.