Homework 3

Problem 1. Prove that if terms $a_1, a_2, \dots a_n$ are in normal form, then so is the list $a_1 :: a_2 :: \cdots a_n :: \mathbf{nil}$.

Solution. We only need to prove that if a, b are in normal form, then so is $a :: \mathbf{nil}$ and $a :: b$. Thus, if $a_i :: a_{i+1} :: \cdots :: a_n :: \mathbf{nil}$ is in normal form, then so is $a_{i-1} :: a_i :: a_{i+1} :: \cdots :: a_n :: \mathbf{nil}$. By induction we can prove the original statement.

If a, b are in normal form, $a :: b \equiv \mathbf{pair}\, ab = \lambda x.xab$. To do β-reduction on this term we have the following approaches: 1. Do the reduction within a single term x, a or b first. 2. Do the reduction on xa first. 3. View xa as a whole and conduct reduction on $(xa)b$ first. Because x, a, b are all in normal form, they can't perform β-reduction singally. Meanwhile because x is a single variable and does not start with λ (so xa does not start with λ as well), xa and $(xa)b$ can't be reduced.

Since $\mathbf{nil} = \lambda x.\mathbf{t}$ is in normal form, when terms $a_1, a_2, \dots a_n$ are also in normal form, by the property proven above $a_n :: \mathbf{nil}$ is also a normal form. Then by using this property inductively we can prove that $a_1 :: a_2 :: \cdots a_n :: \mathbf{nil}$ is also in normal form.

Problem 2. Show that **filter** is a special case of **reduce** for **filter** and **reduce** defined in the class.

Solution. Consider $\mathbf{filter} \equiv \lambda lf.\mathbf{reduce}\, l\, (\lambda ab.(\mathbf{ite}(fa)(\mathbf{cons}\, ab)b))\, \mathbf{nil}$. For any list $l \equiv x :: l_1$, since $\mathbf{reduce}\, (h :: t)\, f\, z \twoheadrightarrow_\beta f\, h\, (\mathbf{reduce}\, t\, f\, z)$, so:

$$
\begin{aligned}
\mathbf{filter}\, l\, f &\twoheadrightarrow_\beta \mathbf{reduce}\, l\, (\lambda ab.(\mathbf{ite}(fa)(\mathbf{cons}\, ab)b))\, \mathbf{nil} \\
&\equiv \mathbf{reduce}\, (x :: l_1)\, (\lambda ab.(\mathbf{ite}(fa)(\mathbf{cons}\, ab)b))\, \mathbf{nil} \\
&\twoheadrightarrow_\beta (\lambda ab.(\mathbf{ite}(fa)(\mathbf{cons}\, ab)b))\, x\, (\mathbf{reduce}\, l_1\, (\lambda ab.(\mathbf{ite}(fa)(\mathbf{cons}\, ab)b))\, \mathbf{nil}) \\
&\twoheadrightarrow_\beta \mathbf{ite}(fx)(\mathbf{cons}\, x(\mathbf{filter}\, l_1\, f))(\mathbf{filter}\, l_1\, f) \\
&\twoheadrightarrow_\beta \begin{cases} x :: \mathbf{filter}\, l_1\, f & \text{if } fx \twoheadrightarrow_\beta \mathbf{t} \\ \mathbf{filter}\, l_1\, f & \text{otherwise} \end{cases}
\end{aligned}
$$

which is the inductive definition of **filter**.

Problem 3. Let $F : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Prove that there is a λ term f representing F in the sense that for all $x_1, x_2, \ldots, x_n \in \{0,1\}$,

$$f[x_1][x_2]\cdots[x_n] \twoheadrightarrow_\beta [F(x_1, x_2, \ldots, x_n)],$$

where $[0] \equiv \mathbf{f}$ and $[1] \equiv \mathbf{t}$.

Solution. Use indution on n. When $n = 1$, then there are only four functions, $F(x) = x$, $F(x) = \neg x$, $F(x) = 0$ and $F(x) = 1$, which can be represented by $\lambda x.x$, $\lambda x.\mathbf{not}\, x \equiv \lambda x.x\mathbf{ft}$, $\lambda x.\mathbf{t}$ and $\lambda x.\mathbf{f}$ respectively.

If the statement holds for n-variable functions, below we'll prove the case $n+1$. Consider functions $F(x_1, x_2 \cdots x_n, 1)$ and $F(x_1, x_2 \cdots x_n, 0)$ which are both n-variable functions. We can use the induction hypothesis to represent them by λ terms f_1 and f_0 respectively. Then we can define $f \equiv \lambda x_1 x_2 \cdots x_n x_{n+1}.(\mathbf{ite}\, x_{n+1}(f_1 x_1 x_2 \cdots x_n)(f_0 x_1 x_2 \cdots x_n))$ to represent F, as:

$$\begin{aligned}
f[x_1][x_2]\cdots[x_{n+1}] &\twoheadrightarrow_\beta \mathbf{ite}[x_{n+1}](f_1[x_1][x_2]\cdots[x_n])(f_0[x_1][x_2]\cdots[x_n]) \\
&\twoheadrightarrow_\beta \mathbf{ite}[x_{n+1}][F(x_1, x_2 \cdots x_n, 1)][F(x_1, x_2 \cdots x_n, 0)] \\
&\twoheadrightarrow_\beta \begin{cases} [F(x_1, x_2 \cdots x_n, 1)] & \text{if } x_{n+1} = 1 \\ [F(x_1, x_2 \cdots x_n, 0)] & \text{if } x_{n+1} = 0 \end{cases} \\
&\equiv [F(x_1, x_2 \cdots x_n, x_{n+1})]
\end{aligned}$$

Problem 4. Let $C \subseteq \Sigma^*$ be a language. Prove that C is Turing-recognizable if and only if there is a decidable language D such that

$$C = \{x \mid \exists y \text{ such that } \langle x, y \rangle \in D\}.$$

Solution. Say $y \in \Sigma_1^*$. Assume Σ_1 is a finite charset, then Σ_1^* is countable. Thus we can assign a positive integer starting with 1 to each $y \in \Sigma_1^*$, e.g. $y_1, y_2, ..., y_i, ...$, and this sequence can cover all strings in Σ_1^*.

If there exists such D which is Turing-decidable, suppose TM M_1 can accept D and always halt, then we can define a TM M_2 to accept C as such:

Algorithm 1: M_2 to accept C

Data: x

i=1;
while *True* **do**
  if M_1 *accepts* $\langle x, y_i \rangle$ **then**
    accept and halt;
  end
  else
    i=i+1;
  end
end

Since M_1 always halt, each step in the While loop will end in finite steps. For any $x \in C$, by condition $C = \{x \mid \exists y\, s.t. \langle x, y \rangle \in D\}$ we know $\exists k \in \mathbb{Z}_+\, s.t. \langle x, y_k \rangle \in D$. So with k steps of iteration, each ending in finite steps, M_2 will accept x. For any $x \notin C$, $\forall y \in \Sigma_1^*$, $\langle x, y \rangle \notin D$, so M_2 will never accept x.

On the other hand, if C is Turing recognizable, say by TM M_2, then $\forall x \in C, \exists T_x \in \mathbb{Z}_+$ such that M_2 will halt and accept x in T_x steps. Thus we can consider $D = \{\langle x, y \rangle \mid x \in C, y \in \Sigma_1^{T_x}\}$. By this definition we can easily verify $C = \{x \mid \exists y\, s.t. \langle x, y \rangle \in D\}$. Use TM M_1 to accept D: M_1 receives $\langle x, y \rangle$ and put x into M_2 and run only $|y| = T_x$ steps. If M_2 accepts x in T_x steps, then M_1 accepts $\langle x, y \rangle$, otherwise it rejects. Obviously all $\langle x, y \rangle \in D$ can be accepted by M_1. All tuples $\langle x, y \rangle$ acceptable by M_1 satisfies $x \in C, |y| = T_x$ and by definition is in D. Also since M_1 will always halt in finite steps (consists of T_x steps of simulation and the finite steps of preparing the configuration for M_2), so D is Turing-decidable.