Homework 3

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

Solution. Prove by induction. First show the following lemma: given s and t are normal forms, $\lambda x.xst$ is also a normal form. You can show that via enumerating all possible beta reduction rules. With the lemma, it is easy to show the base case $a_n :: \mathbf{nil}$ is normal, and similar for the induction steps. Note that it is recommended that you do the induction in a backward fashion.

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

Solution. $\mathbf{filter} \equiv \lambda lf.\mathbf{reduce}\, l\, (\lambda ab.f\ a(\mathbf{cons}\, a\, b)b)\ \mathbf{nil}$

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 $x_1, x_2, \dots, x_n \in \{0,1\}$,

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

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

Solution. Recall that every Boolean function can be written as a CNF/DNF formula, and we have already constructed the $\mathbf{and}, \mathbf{or}, \mathbf{not}$ in homework 2.

A more explicit construction:

Note that $\lambda[x].[x][F(1)][F(0)]$ implements the function $F : \{0,1\} \to \{0,1\}$. Thus we can inductively construct the λ term for n bit functions from $n-1$ bit functions by:

$$\lambda x_1 x_2 \cdots x_n.x_1\,(f_1\, x_2 x_3 \cdots x_n)\,(f_0\, x_2 x_3 \cdots x_n),$$

where f_0, f_1 implements $F(1, x_2, x_3, \cdots, x_n), F(0, x_2, x_3, \cdots, x_n)$

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. If direction: If C is Turing-recognizable, for $x \in C$ we can let $y = 1^n$, where n is the number of steps that the corresponding M_C halts on x. The turing machine for D simulates M_C on x for n steps, and accepts iff M_C accepts x.

Only if direction: If there is some decidable language D that satisfies the requirement, the turing machine for C works as follows: For each $n \in \mathbb{N}$, it would enumerate all possible $y \in \Sigma^n$, and accept iff $M_D(\langle x, y \rangle)$ accepts. Since M_D always halts, thus for any finite y, the TM M_C can reach it in the search process. Thus any x that M_C accepts, there must exist some y such that $\langle x, y \rangle \in D$.

Please note that when proving two sets are equal, it is suggested you should prove the both directions.($\subseteq$ and $\supseteq$)