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. Let's prove a stronger statement: if terms $a_1, a_2, \dots a_n$ are in formal form, then so is the list $a_1 :: a_2 :: \cdots a_{n-1} :: a_n$.

We can prove this by induction on n. When $n = 1$, the list is a_1, which is in formal form. When $n = 2$, the list is $a_1 :: a_2 \equiv \lambda x.xa_1a_2$, which is in formal form because you can not perform β-reduction on this term.

Suppose that the list $a_1 :: a_2 :: \cdots a_{n-1}$ is in formal form. Then the list $a_1 :: a_2 :: \cdots a_{n-1} :: a_n$ could be treated as $a_1 :: t$ where $t \equiv a_2 :: \cdots a_{n-1} :: a_n$. From the induction hypothesis, we know that t is in formal form because it only has $n-1$ elements. And we know that a_1 is in formal form. So $a_1 :: t$ is in formal form.

Therefore, for all ,the list $a_1 :: a_2 :: \cdots a_{n-1} :: a_n$ is in formal form. Make the substitutions $n \leftarrow n+1$ and $a_{n+1} \leftarrow \mathbf{nil}$, we can get the original statement.

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

Solution. The **filter** function keeps the elements that satisfy the condition, and deprecates the elements that do not satisfy the condition. We can define the **filter** function as follows:

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

And l is a list, f is a function that takes an element and returns a boolean value to indicate whether the element satisfies the condition.

A verification of the correctness of the definition is as follows:

$$\begin{aligned}\mathbf{filter}\, \mathbf{nil}\, f &\twoheadrightarrow_\beta \mathbf{reduce}\, \mathbf{nil}\, (\lambda ab.\mathbf{ite}\, f\, (\mathbf{cons}\, a\, b)\, b)\, \mathbf{nil} \\ &\twoheadrightarrow_\beta \mathbf{nil}.\end{aligned}$$

$$\begin{aligned}
\mathbf{filter}\,(h :: t)\,f &\twoheadrightarrow_\beta \mathbf{reduce}\,(h :: t)\,(\lambda ab.\mathbf{ite}\,f(\mathbf{cons}\,a\,b)\,b)\,\mathbf{nil} \\
&\twoheadrightarrow_\beta \mathbf{ite}\,f\,(\mathbf{cons}\,h\,\mathbf{reduce}\,(\lambda ab.\mathbf{ite}\,f(\mathbf{cons}\,a\,b)\,b)\,t\,\mathbf{nil}) \\
&\qquad (\mathbf{reduce}\,(\lambda ab.\mathbf{ite}\,f(\mathbf{cons}\,a\,b)\,b)\,t\,\mathbf{nil}) \\
&\twoheadrightarrow_\beta \mathbf{ite}\,f\,(\mathbf{cons}\,h\,(\mathbf{filter}\,t\,f))\,(\mathbf{filter}\,t\,f).
\end{aligned}$$

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. First of all, simplify the function F into its full disjunctive normal form F'. As a result, we obtain a new function F', which is equivalent to F.

F' has the following form:

$$F'(x_1, x_2, \ldots, x_n) = \bigvee_{i=1}^{t} m_i,$$

where m_i is a intersection of literals, and each literal is a variable or its negation, and t is the number of terms in the disjunctive normal form.

So we define the λ term μ_i as follows:

$$\mu_i \equiv \mathbf{and}\,y_{j_1}\,(\mathbf{and}\,y_{j_2}\,\cdots(\mathbf{and}\,y_{j_{n-1}}\,y_{j_n})),$$

where $y_{j_1}, y_{j_2}, \ldots, y_{j_n}$ are the lambda terms to the variables $x_{j_1}, x_{j_2}, \ldots, x_{j_n}$ in m_i. Then we can define the λ term f as follows:

$$f \equiv \lambda y_1 y_2 \cdots y_t.\mathbf{or}\,\mu_1\,(\mathbf{or}\,\mu_2\,\cdots(\mathbf{or}\,\mu_{t-1}\,\mu_t)).$$

Therefore, we can prove that for all $x_1, x_2, \ldots, x_n \in \{0,1\}$, there always exists a λ term f representing F in the sense that

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

simply because we've given the algorithm to construct such a λ term f.

In short, $f = \lambda y_1 y_2 \ldots y_n.g$ is the resulting λ term, where g is the Boolean function that substitutes $\wedge$ with **and**, $\vee$ with **or** in the Polish notation of the disjunctive normal form of F.

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. Suppose that C is Turing-recognizable. Then there exists a Turing Machine M that recognizes C. We can construct a decidable language D consisting of all pairs $\langle x, y\rangle$, where x is the input of M and y is a possible computation path that M performs on input x. Now, we can see that $C = \{x \mid \exists y \text{ such that } \langle x, y\rangle \in D\}$, because if an input x is accepted by M, there exists a computation path y; otherwise, there is no such y. Additionally, D is decidable because we can construct a decider N which accepts $\langle x, y\rangle$ if x is accepted by M, and rejects $\langle x, y\rangle$ when either x is rejected by M or M does not halt within steps of the length of y.

Hence, there is a decidable language D, such that $C = \{x \mid \exists y \text{ such that } \langle x, y\rangle \in D\}$.

On the other hand, suppose that there is a decidable language D such that $C = \{x \mid \exists y \text{ such that } \langle x, y\rangle \in D\}$. Consider the Turing Machine N which decides D. Select all the states(denoted by a set $\{q_i\}$) of N which are reachable from the start state, satisfying the condition that the edge pointing towards their successive states is labeled with the separation symbol ',’. Also, select all the states(denoted by a set $\{p_i\}$) of N which are the states on the paths that lead to the states in $\{q_i\}$. Then we can construct a new Turing Machine M which decides C, by including all the states in $\{p_i\}$ and $\{q_i\}$ as the states in M, and all the transitions in N between them as the transitions in M, and selecting all the states in $\{q_i\}$ as the accepting states of M, and the start state of N as the start state of M.

Therefore, C is Turing-recognizable, for we have constructed a Turing Machine M which decides C.

In conclusion, C is Turing-recognizable if and only if there is a decidable language D such that $C = \{x \mid \exists y \text{ such that } \langle x, y\rangle \in D\}$.