In-Class Problems Week 5, Mon.

Problem 1.
The Elementary 18.01 Functions (F18's) are the set of functions of one real variable defined recursively as follows:

Base cases:

- The identity function, $\mathrm{id}(x) ::= x$ is an F18,
- any constant function is an F18,
- the sine function is an F18,

Constructor cases:
If f, g are F18's, then so are

1. $f + g$, fg, 2^g,
2. the inverse function f^{-1},
3. the composition $f \circ g$.

(a) Prove that the function $1/x$ is an F18.

Warning: Don't confuse $1/x = x^{-1}$ with the inverse id^{-1} of the identity function $\mathrm{id}(x)$. The inverse id^{-1} is equal to id.

(b) Prove by Structural Induction on this definition that the Elementary 18.01 Functions are *closed under taking derivatives*. That is, show that if $f(x)$ is an F18, then so is $f' ::= df/dx$. (Just work out 2 or 3 of the most interesting constructor cases; you may skip the less interesting ones.)

Problem 2.
Let p be the string [ ] . A string of brackets is said to be *erasable* iff it can be reduced to the empty string by repeatedly erasing occurrences of p. For example, here's how to erase the string [[[ ]] [ ]] [ ] :

$$[\,[\,[\,]\,]\,[\,]\,]\,[\,] \to [\,[\,]\,] \to [\,] \to \lambda.$$

On the other hand the string [ ]] [[[[[ ]] is not erasable because when we try to erase, we get stuck:] [[[:

$$[\,]\,]\,[\,[\,[\,[\,[\,]\,] \to\,]\,[\,[\,[\,[\,] \to\,]\,[\,[\,[\ \not\to$$

Let Erasable be the set of erasable strings of brackets. Let RecMatch be the recursive data type of strings of *matched* brackets defined recursively:

- **Base case**: $\lambda \in$ RecMatch.

- **Constructor case**: If $s, t \in \text{RecMatch}$, then $[\ s\ ]\ t \in \text{RecMatch}$.

.

(a) Use structural induction to prove that

$$\text{RecMatch} \subseteq \text{Erasable}.$$

(b) Supply the missing parts (labeled by "(*)") of the following proof that

$$\text{Erasable} \subseteq \text{RecMatch}.$$

Proof. We prove by strong induction that every length n string in Erasable is also in RecMatch. The induction hypothesis is

$$P(n) ::= \forall x \in \text{Erasable}.\ |x| = n \text{ IMPLIES } x \in \text{RecMatch}.$$

Base case:

(*) What is the base case? Prove that P is true in this case.

Inductive step: To prove $P(n+1)$, suppose $|x| = n + 1$ and $x \in \text{Erasable}$. We need to show that $x \in \text{RecMatch}$.

Let's say that a string y is an *erase* of a string z iff y is the result of erasing a *single* occurrence of p in z.

Since $x \in \text{Erasable}$ and has positive length, there must be an erase, $y \in \text{Erasable}$, of x. So $|y| = n - 1 \geq 0$, and since $y \in \text{Erasable}$, we may assume by induction hypothesis that $y \in \text{RecMatch}$.

Now we argue by cases:

Case (y is the empty string):

(*) Prove that $x \in \text{RecMatch}$ in this case.

Case ($y = [\ s\ ]\ t$ for some strings $s, t \in \text{RecMatch}$): Now we argue by subcases.

- **Subcase**($x = py$):
  (*) Prove that $x \in \text{RecMatch}$ in this subcase.
- **Subcase** (x is of the form $[\ s'\ ]\ t$ where s is an erase of s'):
  Since $s \in \text{RecMatch}$, it is erasable by part (b), which implies that $s' \in \text{Erasable}$. But $|s'| < |x|$, so by induction hypothesis, we may assume that $s' \in \text{RecMatch}$. This shows that x is the result of the constructor step of RecMatch, and therefore $x \in \text{RecMatch}$.
- **Subcase** (x is of the form $[\ s\ ]\ t'$ where t is an erase of t'):
  (*) Prove that $x \in \text{RecMatch}$ in this subcase.

(*) Explain why the above cases are sufficient.

This completes the proof by strong induction on n, so we conclude that $P(n)$ holds for all $n \in \mathbb{N}$. Therefore $x \in \text{RecMatch}$ for every string $x \in \text{Erasable}$. That is, $\text{Erasable} \subseteq \text{RecMatch}$. Combined with part (a), we conclude that

$$\text{Erasable} = \text{RecMatch}.$$

■

Problem 3.
Here is a simple recursive definition of the set, E, of even integers:

Definition. Base case: $0 \in E$.
Constructor cases: If $n \in E$, then so are $n+2$ and $-n$.

Provide similar simple recursive definitions of the following sets:

(a) The set $S ::= \{2^k 3^m 5^n \in \mathbb{N} \mid k, m, n \in \mathbb{N}\}$.

(b) The set $T ::= \{2^k 3^{2k+m} 5^{m+n} \in \mathbb{N} \mid k, m, n \in \mathbb{N}\}$.

(c) The set $L ::= \{(a, b) \in \mathbb{Z}^2 \mid (a - b) \text{ is a multiple of } 3\}$.

Let L' be the set defined by the recursive definition you gave for L in the previous part. Now if you did it right, then $L' = L$, but maybe you made a mistake. So let's check that you got the definition right.

(d) Prove by structural induction on your definition of L' that

$$L' \subseteq L.$$

(e) Confirm that you got the definition right by proving that

$$L \subseteq L'.$$

(f) See if you can give an *unambiguous* recursive definition of L.

Supplemental problem:

Problem 4.

Definition. The recursive data type, binary-2PTG, of *binary trees* with leaf labels, L, is defined recursively as follows:

- **Base case:** $\langle \texttt{leaf}, l \rangle \in$ binary-2PTG, for all labels $l \in L$.
- **Constructor case:** If $G_1, G_2 \in$ binary-2PTG, then

$$\langle \texttt{bintree}, G_1, G_2 \rangle \in \text{binary-2PTG}.$$

The *size*, $|G|$, of $G \in$ binary-2PTG is defined recursively on this definition by:

- **Base case:**

$$|\langle \texttt{leaf}, l \rangle| ::= 1, \quad \text{for all } l \in L.$$

- **Constructor case:**

$$|\langle \texttt{bintree}, G_1, G_2 \rangle| ::= |G_1| + |G_2| + 1.$$

For example, the size of the binary-2PTG, G, pictured in Figure 1, is 7.

(a) Write out (using angle brackets and labels `bintree`, `leaf`, etc.) the binary-2PTG, G, pictured in Figure 1.

The value of flatten(G) for $G \in$ binary-2PTG is the sequence of labels in L of the leaves of G. For example, for the binary-2PTG, G, pictured in Figure 1,

$$\text{flatten}(G) = (\texttt{win}, \texttt{lose}, \texttt{win}, \texttt{win}).$$

(b) Give a recursive definition of flatten. (You may use the operation of *concatenation* (append) of two sequences.)

(c) Prove by structural induction on the definitions of flatten and size that

$$2 \cdot \text{length}(\text{flatten}(G)) = |G| + 1. \tag{1}$$

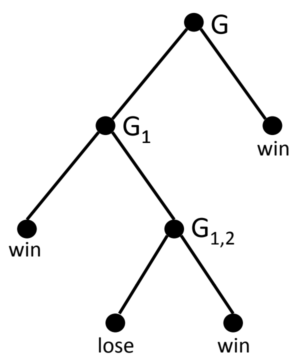

Figure 1 A picture of a binary tree G.