Exercise 1F-2. Language Design [5 points].

1. Comments related to my experience: The paper presents some *objective* criteria for evaluating programming language (PL) design and overall resonates with my general view towards programming language design, underlining the importance of simplicity. Having worked primarily with systems programming languages through the Michigan curriculum and being exposed to the functional paradigm in my research, I definitely believe that mathematical reasoning and simplicity are the key features that influence my affinity for a particular language.

2. Having a rich and simple type system has saved me countless hours of debugging compared to the *dark* side. I also agree with the point that program structure and readability are of utmost importance, specifically when working with large codebases and teams. Having a unified standard and readable code simplifies the onboarding process and makes code quality control easier, saving money at the most expensive part of software production, i.e., human hours.

3. However, I believe machine independence and the use of familiar notions are still important if you want to build things that can run anywhere and not only on a specific machine. This also helps in growing a community as a programming language designer, which then leads to a vibrant community of tooling and support around it, which are overlooked quality-of-life improvements and can go a long distance in making your programs more accessible and reproducible.

Exercise 1F-3. Simple Operational Semantics [3 points].

1. To add support for the division operation on numbers, I have added its corresponding inference rule that informs us of the required premises to carry out division safely.

2. First, we evaluate both the operands with the same starting state (σ). Then, a premise checks that the divisor is not zero. The quotient (n), in this case, should actually be the result of division from using our mathematical intuition (written in the style of `EECS 490`).

$$\frac{\langle a_0, \sigma\rangle \Downarrow n_0 \qquad \langle a_1, \sigma\rangle \Downarrow n_1 \qquad n_1 \neq 0 \qquad n_0 \div n_1 = n(\text{ambient math})}{\langle a_0/a_1, \sigma\rangle \Downarrow n}$$

Exercise 1F-4. Language Feature Design, Large Step [10 points].

1. Here, the `restore` function is used to *restore* the value of x from its previous state (σ) to account for possible shadowing. Initially, I had 2 rules that split this condition into separate cases, but I realized I needed a single rule that covered both cases.

$$\frac{\langle \texttt{e}, \sigma\rangle \Downarrow n \qquad \langle c, \sigma[x := n]\rangle \Downarrow \sigma' \qquad \sigma'' = \text{restore}(\sigma', x, \sigma(x))}{\langle \texttt{let x = e in c}, \sigma\rangle \Downarrow \sigma'}\ \text{LET}$$

Exercise 1F-5. Language Feature Design, Small Step [10 points].

$$r ::= \ldots \mid \text{let x = n in c}$$

$$H ::= \ldots \mid \text{let x = H in c}$$

$$\textit{rules} ::= \ldots \mid \langle \text{let x = n in c}, \sigma\rangle \to \langle \texttt{c}[\text{x := n}]; \sigma\rangle$$

where n is substituted for x in lexical scope of c

Peer Review ID: 302594828 — enter this when you fill out your peer evaluation via gradescope