Exercise 1F-2. Language Design.

Hoare's article put emphisis on the the importance of programming documentation and programming debugging. I strongly agree with that. During my last internship, I worked as a software development engineer. However, the most of my time was spent on reading and understanding the code rather than developing. It is painful to read code written by others without a clear documentation, especially when the code is made by many people. As a programmer, debugging is the most usual task. However, this usual task can be unusually hard. Especally when it comes to multi-thread programming. It is very hard to locate or even reproduce a bug. It will be quite helpful to have a language that can detect errors before actually running the software.

I'm in favor of the importance of efficiency of object code. Besides the reasoning given in the article, I'd like to extend the 'efficiency' to include not only time-efficiency but also energy-efficiency. There are uncountablely many computers all over the world, which consume a lot of energy. Different programming languages have very different energy-efficiency. For example, Go consumes about 2 times more energy than C and Python uses 70 times more[1].If the programming languages are more efficient, a lot of energy will be saved.

In the reasoning of fast translation, the author argues that to split a large program into moodules is cumbersome and has many other disadvantages. I'm against this. If a program can be completed using just a single source file, there won't be many lines of code (around 5000 lines for example). To compile such a short program won't take long. And I believe it is not too small to hold a complete logic unit. However, for a truely large scale program, like Linux, there are about 27.8 million lines of code [2]. Can you image just one file contains these lines of code? So, it is very important to split large scale programs into modules.

Exercise 1F-3. Simple Operational Semantics.

To add a division operator, the expressions need to be able to change the state. Because devided by 0 will lead to an error. Since an error can occur, we also need to add an error state σ_{err}. And the judgement will be like $< \text{e}, \sigma > \Downarrow < \text{n}, \sigma' >$ which means if an error occurs, $\sigma' = \sigma_{\text{err}}$ and the evaluated value n is ignored and if no error, $\sigma' = \sigma$ and n is correctly evaluated.

There will be new general rules of inferrence:

$$\frac{< \text{e}_1, \sigma > \Downarrow < \text{n}, \sigma_{\text{err}} >}{< \text{e}_1 \text{ op } \text{e}_2, \sigma > \Downarrow < \text{n}, \sigma_{\text{err}} >}, \quad \frac{< \text{e}_2, \sigma > \Downarrow < \text{n}, \sigma_{\text{err}} >}{< \text{e}_1 \text{ op } \text{e}_2, \sigma > \Downarrow < \text{n}, \sigma_{\text{err}} >}$$

where $\text{op} = \{+, -, *, /\}$. These two rules mean that if an error occurs during the evaluation of sub-expressions, the evaluation changes the state to error state.

[1] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. ***Energy Efficiency across Programming Languages.***

[2] Swapnil Bhartiya. 2020. ***Linux in 2020: 27.8 million lines of code in the kernel, 1.3 million in systemd.***

For the division operator, the new rules are:

$$\frac{< \mathrm{e}_2, \sigma > \Downarrow < 0, \sigma >}{< \mathrm{e}_1/\mathrm{e}_2, \sigma > \Downarrow < \mathrm{n}, \sigma_{\mathrm{err}} >}, \quad \frac{< \mathrm{e}_1, \sigma > \Downarrow < \mathrm{n}_1, \sigma > \quad < \mathrm{e}_2, \sigma > \Downarrow < \mathrm{n}_2 \neq 0, \sigma >}{< \mathrm{e}_1/\mathrm{e}_2, \sigma > \Downarrow < \mathrm{n}_1/\mathrm{n}_2, \sigma >}$$

These two rules mean that if e_2 evaluates to 0, the evaluation will change the state to error state. Otherwise, the evaluation is correctly done.

Exercise 1F-4. Language Feature Design, Large Step.

$$\frac{\dfrac{< e, \sigma > \Downarrow n}{< x := e, \sigma > \Downarrow \sigma''} \quad < c;\ x := \sigma(x), \sigma'' > \Downarrow \sigma'}{< \text{let } x = e \text{ in } c, \sigma > \Downarrow \sigma'}$$

Exercise 1F-5. Language Feature Design, Small Step.

Redex:

$$\begin{aligned} r ::== \ & \cdots \\ & \mid \text{let } x = n \text{ in } c \\ & \mid \text{let skip in } c \end{aligned}$$

Reduction rule:

$$\begin{aligned} &< \text{let } x = n \text{ in } c, \sigma > \rightarrow < \text{let skip in } x := n; c; x := \sigma(x), \sigma > \\ &< \text{let skip in } c, \sigma > \rightarrow < \text{skip}; c, \sigma > \end{aligned}$$

Context:

$$\begin{aligned} H ::== \ & \cdots \\ & \mid \text{let } H \text{ in } c \end{aligned}$$

1 HW1 (select all pages: your first page has your name and bookkeeping, and all others are anonymous))

- **0 pts** Correct