2 Language Design

I find Dr. Hoare's advice on language feature design really useful and inspiring. In *Hints On Programming Language Design*, Dr. Hoare described two general rules for designing new features. Fist, one should focus on one feature at a time and evaluate such feature through multiple testings. Second, the developer should also actively seek for alternatives designed by others. The goal should be consolidation rather than innovation.

My personal experience of designing a Relevance Vector Machine API is a prefect example of why we should follow Dr. Hoare's rules for designing new features. I got the idea of developing a Relevance Vector Machine API since it was used frequently in my research. At the beginning, I tried to output the results and also provide feedback for users based on the data input. Ironically, I noticed that my main feature was flawed when I was testing the feedback feature. It performed poorly on high dimensional data. Instead of working on offering feedback, I had to improve the performance of the Relevance Vector Machine algorithm itself. Simply focusing on delivering one reliable feature was challenging enough. This is exactly the first rule Dr. Hoare stated.

Later, I studied the source code for Relevance Vector Machine feature provided by the *Scikit-learn* library. That was extremely helpful just like what Dr. Hoare claimed. It turned out that the biggest challenge in my program was already addressed by *Scikit-learn*. By utilizing this library, I was able to spend my time mostly on the feedback feature development. Referring to the solutions provided by others can be really beneficial for developing new feature. We do not have to build from scratch. The critical task is consolidation, not innovation.

3 Simple Operational Semantics

In the IMP language, integer is the only data type for number. Therefore, division can be performed directly. However, we should check if the denominator equals to 0.

For $\frac{e_1}{e_2} = \frac{n_1}{n_2}$

$$\frac{< n_2 = 0, \sigma > \Downarrow \text{false}, \quad < e_1, \sigma > \Downarrow n_1, \quad < e_2, \sigma > \Downarrow n_2}{< \text{if } \ n_2 = 0 \text{ then skip, else } \frac{e_1}{e_2}, \ \sigma > \Downarrow \frac{n_1}{n_2}}$$

$$\frac{< n_2 = 0, \sigma > \Downarrow \text{true}}{< \text{if } \ n_2 = 0 \text{ then skip, else } \frac{e_1}{e_2}, \ \sigma > \Downarrow \frac{n_1}{n_2}}$$

4 Language Feature Design, Large Step

let x=e in c:

$$\frac{< \text{temp} := x, \sigma_1 > \Downarrow \sigma_2 \ \ , < x := e, \sigma_2 > \Downarrow \sigma_3, \quad < c, \sigma_3 > \Downarrow \sigma_4, \quad < x := \text{temp}, \sigma_4 > \Downarrow \sigma_5}{< \text{let } \ x = e \text{ in } c > \Downarrow \sigma_5}$$

The key logic of let command is to store the information of x in a temporary variable *temp*. *temp* is stored in state 2: σ_2 and carried all the way to the final state σ_5. This allow us to reassign the original information inside x to x in the final state without altering other variables and commands.

5 Language Feature Design, Small Step

Redexes:

$$r ::= ...|\ temp := n \mid x := e \mid c \mid x := temp$$

Contexts:

$$H ::= ...|\ let\ H := e\ in\ c\ with\ \sigma[H := n] \mid let\ x := H\ in\ c\ with\ \sigma[x := n]$$

$$\mid let\ x := e\ in\ H\ with\ \sigma[x := n] \mid let\ x := e\ in\ c\ with\ \sigma[x := H]$$

Reduction Rules:

$$< let\ \ x := e\ \ in\ \ c,\ \ \sigma[x := n] > \ \to < let\ \ x := e\ \ in\ \ c,\ \ \sigma[temp := n] >$$

$$< let\ \ x := e\ \ in\ \ c,\ \ \sigma[temp := n] > \ \to < skip;\ \ c,\ \ \sigma[x := e,\ temp := n] >$$

$$< skip;\ \ c,\ \ \sigma[x := e,\ temp := n] > \to < c,\ \ \sigma[x := e,\ temp := n](c) >$$

$$< c,\ \ \sigma[x := e,\ temp := n](c) > \to < x := temp,\ \ \sigma[temp := n](c) >$$

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

- 0 pts Correct