Exercise 1F-2. Language Design

Hoare's *Hints on Programming Language Design* remains strikingly relevant, even over 50 years later. One particularly insightful point is his emphasis on the importance of fast compilation to provide feedback during development. In my experience, immediate feedback is essential for productivity. Whether through fast compilers, interpreted languages that bypass full compilation, or modern static analysis tools that highlight errors as you type, minimizing delay accelerates iteration. For example, when I worked with Java, we kept module compilation and test times under five seconds, and tools like IntelliJ flagged static errors within milliseconds, enabling a seamless development flow.

However, Hoare's claim that lexing and parsing are the slowest parts of compilation is outdated. While this may have been true in his era, advances in parsing techniques, hardware, and compiler design have shifted the bottleneck to later stages like optimization and type checking. Modern compilers spend far more time on these phases, with parsing typically being one of the fastest. Although complex or unconventional syntax can occasionally complicate parsing, such cases are exceptions rather than the norm.

Hoare's discussion of independent compilation also raises valuable points about program design. His examples using **PRECOMPILE** and **DUMP** directives anticipate the modern concept of incremental compilation. Incremental compilation allows programmers to define boundaries based on conceptual domains rather than to minimize recompilation. This approach fosters granularity and flexibility, enabling faster feedback and more efficient workflows than strict independent compilation.

Exercise 1F-3. Simple Operational Semantics

We first extend the grammar of arithmetic expressions to include division.

$$e \quad ::= \quad \cdots \mid e \div e$$

We then define a new rule E-Div to perform big step integer division in the case that e_2 does not evaluate to 0. Similarly to in class we are using n as a metavariable ranging over the integers. In the conclusion we are performing actual mathematical integer division on n_1 and n_2.

$$\textsc{E-Div}\ \frac{\langle e_1, \sigma\rangle \Downarrow n_1 \qquad \langle e_2, \sigma\rangle \Downarrow n_2 \qquad n_2 \neq 0}{\langle e_1 \div e_2, \sigma\rangle \Downarrow \lfloor n_1 \div n_2 \rfloor}$$

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

We will use x as a metavariable over variables.

$$c \quad ::= \quad \cdots \mid \text{let } x = e \text{ in } c \mid \text{print } e$$

$$\textsc{E-Let}\ \frac{\langle e, \sigma\rangle \Downarrow n \qquad \langle c, \sigma[x := n]\rangle \Downarrow \sigma'}{\langle \text{let } x = e \text{ in } c, \sigma\rangle \Downarrow \text{revert}(\sigma, \sigma', x)}$$

where $\text{revert}(\sigma, \sigma', x) = \sigma'[x := \sigma(x)]$ if $x \in \sigma$ and otherwise $\text{revert}(\sigma, \sigma', x) = \sigma' \setminus \{x\}$. This denotes reverting the variable binding in x to the state in σ.

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

We will introduce an unset redex into the language to represent removing a variable from σ.

$$\begin{array}{lll} r & ::= & \cdots \mid \text{let } x = n \text{ in } c \mid \text{unset } x; c \\ H & ::= & \cdots \mid \text{let } x = H \text{ in } c \end{array}$$

Question assigned to the following page: 5

$$\langle \text{let } x = n \text{ in } c, \sigma \rangle \to \langle c; x := \sigma(x), \sigma[x := n] \rangle \text{ where } x \in \sigma$$

$$\langle \text{let } x = n \text{ in } c, \sigma \rangle \to \langle c; \text{unset } x, \sigma[x := n] \rangle \text{ where } x \notin \sigma$$

$$\langle \text{unset } x, \sigma \rangle \to \langle \text{unset } x, \sigma \setminus \{x\} \rangle \text{ where } x \in \sigma$$