1F-2

I think a really great point that Hoare makes is in the section on efficiency. The arguments he me mentions that are used to justify a lackadaisical approach to program optimization in favor of development speed are all very common. I see discussions like this every day, especially in the context of supposedly modern, high-level languages which claim to do more work, enabling to programmer to spend less time thinking about details such as efficiency. I am always skeptical of such claims, largely due to consistent evidence that such claims are at best a hand-waving distraction from the very real problem Hoare describes. The comment that "all too frequently [this argument] is used to an justify an efficiency loss of a factor of two, or ten, or even more immediately brings to mind an experience I had in the last week while attempting to play the game Tetris with a friend online. It was quite surprising to me when this friend commented that the game was running slow wasn't able to keep up, especially due to the fact that the game is largely unchanged from its release in 1984. In this case, the behavior in question is much more than ten times older than the machine it is being run on, but even that 37 years of increase in the speed and capacity of computers wasn't enough to compensate for the inefficiencies of the program.

The one point that Hoare asserts in that section on Efficiency that I would take issue with is "However cheap and fast a computer is, it will be cheaper and faster to use it more efficiently." In the context from which this quote is taken, it is clear that this refers to the cost and speed of running a program, but the particular language used brings up an interesting discussion about the context in which a program is both run and developed. The most extreme example of this would be a one-off script that an individual writes such as one for performing some data analysis. In this case, it is likely that the computer the code would be run on is sitting largely idle until the program is run, and any discussion on the efficiency of the process would be, in my opinion, incomplete without consideration of that potentially wasted time. In a situation such as this, it would almost certainly be cheaper and faster by any metric worth considering to focus on improving the development process rather than anything to do with the efficiency of the resulting program. Although this is a very specific example, the software is always written by individuals or organizations with an end goal in mind, and development time and costs are very real things to consider in an analysis of the cost and speed of a program.

1F-3

An inference rule for integer division can be defined as follows:

$$\frac{< e_1, \sigma >\Downarrow n_1 \qquad < e_2, \sigma >\Downarrow n_2 \qquad \exists\, n \in \mathbb{I}\ s.t.\ n_1 = n_2 * n}{< e_1/e_2, \sigma >\Downarrow n_1/n_2}$$

This is would make the language incomplete as it is not defined for operands that would produce a non-integer result. It could be expanded to include integers by truncating the decimal part, as is common in many programming languages with a strict integer type:

$$\frac{< e_1, \sigma >\Downarrow n_1 \qquad < e_2, \sigma >\Downarrow n_2 \qquad n_2 \neq 0 \qquad n_1/n_2 \geq 0}{< e_1/e_2, \sigma >\Downarrow \lfloor n_1/n_2 \rfloor}$$

$$\frac{< e_1, \sigma >\Downarrow n_1 \qquad < e_2, \sigma >\Downarrow n_2 \qquad n_2 \neq 0 \qquad n_1/n_2 < 0}{< e_1/e_2, \sigma >\Downarrow \lceil n_1/n_2 \rceil}$$

This is still not complete, due to the restriction $n_2 \neq 0$. This is an incompleteness that is present in division as well, so there isn't a clear definition for division that is complete. Alternatives include expanding the range to include a special non-numerical value indicating division by zero, or to use a panic, exception, interrupt or similar that halts execution of the program. Such a behavior would not be expressible with big-step operation semantics, as in that case the division expression would not produce a value.

1F-4

The let expression amends σ with the new variable during evaluation of c. σ', which results from evaluation of c notable does not include a mapping for the new variable.

$$\frac{< e, \sigma >\Downarrow v \qquad < c, \sigma[x := v] >\Downarrow \sigma'}{< \text{let } x = e \text{ in } c, \sigma >\Downarrow \sigma'}$$

Peer Review ID: 62817498 — enter this when you fill out your peer evaluation via gradescope

1F-5

Redex:

$$r ::= ... \\ \quad | \text{ let } x = n \text{ in } c$$

Context:

$$H ::= ... \\ \quad | \text{ let } x = \bullet \text{ in } c$$

Reduction rule:

$$< \text{ let } x = n \text{ in } c, \sigma > \rightarrow < c, \sigma[x := n] >$$

Peer Review ID: 62817498 — enter this when you fill out your peer evaluation via gradescope

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

- **0 pts** Correct