1 2F-1 Bookkeeping

- 0 pts Correct

Exercise 2. Intuitively, "All flowers smell the same" is false even for $n = 2$. Correspondingly, it can be spotted that the following sentence is flawed:

> *Induction Step:* ... Pick any arbitrary $x \in Y \cap Y'$. ...

This move implicitly assumes $Y \cap Y' \neq \varnothing$ so that x can be picked, which is wrong for $n = 2$, where $X = \{f, f'\}$, $Y = X - \{f\} = \{f'\}$, $Y' = X - \{f'\} = \{f\}$, and $Y \cap Y' = \varnothing$.

2 2F-2 Mathematical Induction

- 0 pts Correct

Exercise 3. Prove by induction on the derivation $D :: \langle \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma\rangle \Downarrow \sigma'$; the goal is to show that if $\sigma(x)$ is even then $\sigma'(x)$ is even as well. The base case is when the last rule used in D is while-false, i.e.,

$$D :: \frac{\langle b, \sigma\rangle \Downarrow \texttt{false}}{\langle \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma\rangle \Downarrow \sigma}\ .$$

In this case $\sigma' = \sigma$, and hence obviously if $\sigma(x)$ is even then $\sigma'(x)$ is even as well. Otherwise, by inversion the last rule used in D must be while-true, i.e.,

$$D :: \frac{\langle b, \sigma\rangle \Downarrow \texttt{true} \qquad \widetilde{D} :: \langle x := x+2; \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma\rangle \Downarrow \sigma'}{\langle \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma\rangle \Downarrow \sigma'}\ .$$

Moreover by further inversion on $\widetilde{D}$,

$$D :: \frac{\langle b, \sigma\rangle \Downarrow \texttt{true} \qquad \widetilde{D} :: \dfrac{\langle x := x+2, \sigma\rangle \Downarrow \widetilde{\sigma} \qquad D' :: \langle \texttt{while}\ b\ \texttt{do}\ x := x+2, \widetilde{\sigma}\rangle \Downarrow \sigma'}{\langle x := x+2; \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma\rangle \Downarrow \sigma'}}{\langle \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma\rangle \Downarrow \sigma'}\ .$$

It is easy to see (while requiring a couple of steps, elaborated below) that $\widetilde{\sigma}(x) = \sigma(x) + 2$. Hence if $\sigma(x)$ is even, then $\widetilde{\sigma}(x)$ is even. Then by inductive hypothesis on D', $\widetilde{\sigma}(x)$ being even implies that $\sigma'(x)$ is even as well, which completes the induction.

To see why $\widetilde{\sigma}(x) = \sigma(x) + 2$, observe the following (unique) derivation:

$$\frac{\dfrac{\overline{\langle x, \sigma\rangle \Downarrow \sigma(x)} \qquad \overline{\langle 2, \sigma\rangle \Downarrow 2}}{\langle x+2, \sigma\rangle \Downarrow \sigma(x) + 2}}{\langle x := x+2, \sigma\rangle \Downarrow \sigma[x := \sigma(x) + 2]}\ .$$

3 2F-3 While Induction

- 0 pts Correct

Peer Review ID: 65425415 — enter this when you fill out your peer evaluation via gradescope

Exercise 4. The six new rules are as follows. For `throw`:

$$\frac{\langle e, \sigma\rangle \Downarrow n}{\langle \texttt{throw}\ e, \sigma\rangle \Downarrow \sigma\ \texttt{exc}\ n}\ ;$$

for `try`:

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \texttt{try}\ c_1\ \texttt{catch}\ x\ c_2, \sigma\rangle \Downarrow \sigma'}\ , \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \texttt{exc}\ n \qquad \langle c_2, \sigma'[x := n]\rangle \Downarrow t}{\langle \texttt{try}\ c_1\ \texttt{catch}\ x\ c_2, \sigma\rangle \Downarrow t}\ ;$$

and for `finally`:

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \qquad \langle c_2, \sigma'\rangle \Downarrow t}{\langle \texttt{after}\ c_1\ \texttt{finally}\ c_2, \sigma\rangle \Downarrow t}\ , \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \texttt{exc}\ n \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''}{\langle \texttt{after}\ c_1\ \texttt{finally}\ c_2, \sigma\rangle \Downarrow \sigma''\ \texttt{exc}\ n}\ ,$$

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \texttt{exc}\ n \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''\ \texttt{exc}\ m}{\langle \texttt{after}\ c_1\ \texttt{finally}\ c_2, \sigma\rangle \Downarrow \sigma''\ \texttt{exc}\ m}\ .$$

4 2F-4 Language Features, Large Step

- 0 pts Correct

Exercise 5. I agree with the claim that it is "more elegant" to describe "IMP with exceptions" using small-step contextual semantics. Compared with large-step semantics, small-step semantics follows the style of repeatedly rewriting expressions/commands in a program, and this kind of rewriting could lead to especially elegant description of exceptions. E.g. the following rule for `try`

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \ \texttt{exc}\ n \qquad \langle c_2, \sigma'[x := n]\rangle \Downarrow t}{\langle \texttt{try}\ c_1\ \texttt{catch}\ x\ c_2, \sigma\rangle \Downarrow t}$$

can be carried out by resolving c_1 generically using context $\texttt{try} \bullet \texttt{catch}\ x\ c_2$ and then rewriting using reduction rule

$$\langle \texttt{try}\ \texttt{throw}\ n\ \texttt{catch}\ x\ c_2, \sigma\rangle \to \langle x := n; c_2, \sigma\rangle \ ,$$

avoiding having some relatively heavy term $\sigma'[x := n]$, which should be viewed as some non-elegant duplicate work with the rule for assignment.[1] Similarly, and more significantly, the following two rules for `finally`

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \ \texttt{exc}\ n \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''}{\langle \texttt{after}\ c_1\ \texttt{finally}\ c_2, \sigma\rangle \Downarrow \sigma'' \ \texttt{exc}\ n} \ , \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \ \texttt{exc}\ n \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma'' \ \texttt{exc}\ m}{\langle \texttt{after}\ c_1\ \texttt{finally}\ c_2, \sigma\rangle \Downarrow \sigma'' \ \texttt{exc}\ m}$$

can be *unified by one single* reduction rule

$$\langle \texttt{after}\ \texttt{throw}\ n\ \texttt{finally}\ c_2, \sigma\rangle \to \langle c_2; \texttt{throw}\ n, \sigma\rangle \ .$$

Besides these simplifications in rules,[2] it is also, while personally, more elegant not to have some union termination type but to still use merely σ everywhere and to let terminal commands `skip` / `throw` n manifest whether a program terminates normally or exceptionally.

[1] The other rule for `try` might correspond to reduction rule

$$\langle \texttt{try}\ \texttt{skip}\ \texttt{catch}\ x\ c_2, \sigma\rangle \to \langle \texttt{skip}, \sigma\rangle \ .$$

Also by comparing these two reduction rules for `try` it is clear that the terminal commands now become both `skip` and `throw` n, and it is elegant that the reduction rules for `try` simply deal with both cases.

[2] One might argue that all these benefits for `try` and `finally` were not genuine as we might as well have the following rules in large-step semantics:

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \ \texttt{exc}\ n \qquad \langle x := n; c_2, \sigma'\rangle \Downarrow t}{\langle \texttt{try}\ c_1\ \texttt{catch}\ x\ c_2, \sigma\rangle \Downarrow t} \ , \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \ \texttt{exc}\ n \qquad \langle c_2; \texttt{throw}\ n, \sigma'\rangle \Downarrow t}{\langle \texttt{after}\ c_1\ \texttt{finally}\ c_2, \sigma\rangle \Downarrow t} \ .$$

However this argument is not necessarily valid, as n is mathematical integer in $\Downarrow \sigma'\ \texttt{exc}\ n$, while in the commands $x := n$ and `throw` n, n should be program literal, and it is vague whether this kind of matching is allowed in large-step semantics. (There could be workaround e.g. by adding another condition $\langle e, \sigma'\rangle \Downarrow n$ and using $x := e$ and `throw` e instead, which however becomes super non-elegant.)

5 2F-5 Language Features, Analysis

- 0 pts Correct