1 2F-1 Bookkeeping

- **0 pts** Correct

Exercise 2F-2

The flaw occurs in the *Induction Step*, which is copied here with the first point of issue highlighted:

> Let n be arbitrary and assume that all subsets of F of size at most n contain flowers that smell the same. We will prove that the same thing holds for all subsets of size at most $n+1$. Pick an arbitrary set X such that $|X| = n+1$. Pick two distinct flowers $f, f' \in X$ and let's show that $\mathsf{smells}(f) = \mathsf{smells}(f')$. Let $Y = X - \{f\}$ and $Y' = X - \{f'\}$. Obviously Y and Y' are sets of size at most n so the induction hypothesis holds for both of them. Pick any arbitrary $x \in Y \cap Y'$. Obviously, $x \neq f$ and $x \neq f'$. We have that $\mathsf{smells}(f') = \mathsf{smells}(x)$ (from the induction hypothesis on Y) and $\mathsf{smells}(f) = \mathsf{smells}(x)$ (from the induction hypothesis on Y'). Hence $\mathsf{smells}(f) = \mathsf{smells}(f')$, which proves the inductive step, and the theorem.

At this point in the proof, we aren't guaranteed that $Y \cap Y' \neq \varnothing$, so we might not be able to pick an arbitrary $x \in Y \cap Y'$. This can occur when $n = 1$, so $|X| = 2$ and $X = \{f, f'\} \implies Y = \{f'\}$ and $Y' = \{f\}$. This causes the inductive argument to fall apart, since we can't prove by this method that all sets of 2 distinct flowers have those flowers smelling the same.

Exercise 2F-3

Restating our big-step rules for the `while` command is a good start:

$$\frac{\langle b, \sigma\rangle \Downarrow \textbf{false}}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma} \qquad \frac{\langle b, \sigma\rangle \Downarrow \textbf{true} \quad \langle c, \sigma\rangle \Downarrow \sigma'' \quad \langle \texttt{while}\ b\ \texttt{do}\ c, \sigma''\rangle \Downarrow \sigma'}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma'}$$

We'll prove by induction on the structure of derivations formed by the command $W = (\texttt{while}\ b\ \texttt{do}\ x := x + 2)$ using these rules. The property we want to show for all derivations D is:

$$P(D) = \sigma(x) \text{ even}, D :: \langle W, \sigma\rangle \Downarrow \sigma' \implies \sigma'(x) \text{ even}$$

Note that we can recursively define our derivations in terms of the following (where $D' \prec D$; my definition of D may be abusing notation a bit, but I

2 2F-2 Mathematical Induction

- **0 pts** Correct

Exercise 2F-2

The flaw occurs in the *Induction Step*, which is copied here with the first point of issue highlighted:

> Let n be arbitrary and assume that all subsets of F of size at most n contain flowers that smell the same. We will prove that the same thing holds for all subsets of size at most $n+1$. Pick an arbitrary set X such that $|X| = n+1$. Pick two distinct flowers $f, f' \in X$ and let's show that $\mathsf{smells}(f) = \mathsf{smells}(f')$. Let $Y = X - \{f\}$ and $Y' = X - \{f'\}$. Obviously Y and Y' are sets of size at most n so the induction hypothesis holds for both of them. Pick any arbitrary $x \in Y \cap Y'$. Obviously, $x \neq f$ and $x \neq f'$. We have that $\mathsf{smells}(f') = \mathsf{smells}(x)$ (from the induction hypothesis on Y) and $\mathsf{smells}(f) = \mathsf{smells}(x)$ (from the induction hypothesis on Y'). Hence $\mathsf{smells}(f) = \mathsf{smells}(f')$, which proves the inductive step, and the theorem.

At this point in the proof, we aren't guaranteed that $Y \cap Y' \neq \varnothing$, so we might not be able to pick an arbitrary $x \in Y \cap Y'$. This can occur when $n = 1$, so $|X| = 2$ and $X = \{f, f'\} \implies Y = \{f'\}$ and $Y' = \{f\}$. This causes the inductive argument to fall apart, since we can't prove by this method that all sets of 2 distinct flowers have those flowers smelling the same.

Exercise 2F-3

Restating our big-step rules for the `while` command is a good start:

$$\frac{\langle b, \sigma\rangle \Downarrow \textbf{false}}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma} \qquad \frac{\langle b, \sigma\rangle \Downarrow \textbf{true} \quad \langle c, \sigma\rangle \Downarrow \sigma'' \quad \langle \texttt{while}\ b\ \texttt{do}\ c, \sigma''\rangle \Downarrow \sigma'}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma'}$$

We'll prove by induction on the structure of derivations formed by the command $W = (\texttt{while}\ b\ \texttt{do}\ x := x + 2)$ using these rules. The property we want to show for all derivations D is:

$$P(D) = \sigma(x) \text{ even}, D :: \langle W, \sigma\rangle \Downarrow \sigma' \implies \sigma'(x) \text{ even}$$

Note that we can recursively define our derivations in terms of the following (where $D' \prec D$; my definition of D may be abusing notation a bit, but I

think the intent is clear):

$$D_0 := \frac{\langle b, \sigma\rangle \Downarrow \textbf{false}}{\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma}$$

$$D := \frac{\langle b, \sigma\rangle \Downarrow \textbf{true} \quad \langle x := x + 2, \sigma\rangle \Downarrow \sigma'' \quad D' :: \langle W, \sigma''\rangle \Downarrow \sigma'}{\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma'}$$

First, consider the base case of D_0. Here, $\langle W, \sigma\rangle \Downarrow \sigma' = \sigma$ certainly has the property that $\sigma(x)$ being even $\implies \sigma'(x) = \sigma(x)$ is even.

For the inductive case, assume all derivations $D' \prec D$ have the desired property, and consider the derivation of D defined above. The property holds for the derivation of $\langle x := x+2, \sigma\rangle \Downarrow \sigma''$; it is a basic mathematical fact that $\sigma(x)$ even $\implies \sigma''(x)$ even in this case. And since $\sigma''(x)$ is even, we can use the inductive assumption to conclude that $\sigma'(x)$ is even as well, since $D' \prec D$ and $D' :: \langle W, \sigma''\rangle \Downarrow \sigma'$.

As such, by structural induction, all possible derivations D of the command W must adhere to property P. This means that for all b, c, σ:

$$\sigma(x) \text{ even}, \langle \texttt{while}\ b\ \texttt{do}\ x := x + 2\rangle \Downarrow \sigma' \implies \sigma'(x) \text{ even}$$

Exercise 2F-4

First, we have a simple rule for the `throw` command:

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

And two rules for the `try-catch` command:

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

3 2F-3 While Induction

- **0 pts** Correct

think the intent is clear):

$$D_0 := \frac{\langle b, \sigma\rangle \Downarrow \textbf{false}}{\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma}$$

$$D := \frac{\langle b, \sigma\rangle \Downarrow \textbf{true} \quad \langle x := x + 2, \sigma\rangle \Downarrow \sigma'' \quad D' :: \langle W, \sigma''\rangle \Downarrow \sigma'}{\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma'}$$

First, consider the base case of D_0. Here, $\langle W, \sigma\rangle \Downarrow \sigma' = \sigma$ certainly has the property that $\sigma(x)$ being even $\implies \sigma'(x) = \sigma(x)$ is even.

For the inductive case, assume all derivations $D' \prec D$ have the desired property, and consider the derivation of D defined above. The property holds for the derivation of $\langle x := x+2, \sigma\rangle \Downarrow \sigma''$; it is a basic mathematical fact that $\sigma(x)$ even $\implies \sigma''(x)$ even in this case. And since $\sigma''(x)$ is even, we can use the inductive assumption to conclude that $\sigma'(x)$ is even as well, since $D' \prec D$ and $D' :: \langle W, \sigma''\rangle \Downarrow \sigma'$.

As such, by structural induction, all possible derivations D of the command W must adhere to property P. This means that for all b, c, σ:

$$\sigma(x) \text{ even}, \langle \texttt{while}\ b\ \texttt{do}\ x := x + 2\rangle \Downarrow \sigma' \implies \sigma'(x) \text{ even}$$

Exercise 2F-4

First, we have a simple rule for the `throw` command:

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

And two rules for the `try-catch` command:

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

And three rules for the `after-finally` command:

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

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \texttt{ exc } n \quad \langle c_2, \sigma'\rangle \Downarrow \sigma''}{\langle \texttt{after } c_1 \texttt{ finally } c_2, \sigma\rangle \Downarrow \sigma'' \texttt{ exc } n} \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \texttt{ exc } n_1 \quad \langle c_2, \sigma'\rangle \Downarrow \sigma'' \texttt{ exc } n_2}{\langle \texttt{after } c_1 \texttt{ finally } c_2, \sigma\rangle \Downarrow \sigma'' \texttt{ exc } n_2}$$

Exercise 2F-5

It would be simpler to describe "IMP with exceptions" using small-step contextual semantics as opposed to large-step operational semantics. This is because the idea of certain commands being reduced before others (like the try block before the catch block in `try-catch`) can be baked into the contexts, making for a more intuitive representation of `try-catch` and `after-finally`. Not only that, but the modification of certain existing redexes/reduction rules would be simpler. In doing the coding part of this assignment, I realized that the addition of exceptions would require some modifications/additions to the large-step rules for `while`. But with the unraveling method that small-step semantics use for `while`, the new exceptional case (of a command throwing an exception in a while loop) could be handled by modifications only to the sequencing reduction rule(s) of redex (`skip`, c). This would streamline the addition of exceptions a little bit over their addition in the large-step semantics.

4 2F-4 Language Features, Large Step

- **0 pts** Correct

And three rules for the `after-finally` command:

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

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \texttt{ exc } n \quad \langle c_2, \sigma'\rangle \Downarrow \sigma''}{\langle \texttt{after } c_1 \texttt{ finally } c_2, \sigma\rangle \Downarrow \sigma'' \texttt{ exc } n} \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \texttt{ exc } n_1 \quad \langle c_2, \sigma'\rangle \Downarrow \sigma'' \texttt{ exc } n_2}{\langle \texttt{after } c_1 \texttt{ finally } c_2, \sigma\rangle \Downarrow \sigma'' \texttt{ exc } n_2}$$

Exercise 2F-5

It would be simpler to describe "IMP with exceptions" using small-step contextual semantics as opposed to large-step operational semantics. This is because the idea of certain commands being reduced before others (like the try block before the catch block in `try-catch`) can be baked into the contexts, making for a more intuitive representation of `try-catch` and `after-finally`. Not only that, but the modification of certain existing redexes/reduction rules would be simpler. In doing the coding part of this assignment, I realized that the addition of exceptions would require some modifications/additions to the large-step rules for `while`. But with the unraveling method that small-step semantics use for `while`, the new exceptional case (of a command throwing an exception in a while loop) could be handled by modifications only to the sequencing reduction rule(s) of redex (`skip`, c). This would streamline the addition of exceptions a little bit over their addition in the large-step semantics.

5 2F-5 Language Features, Analysis

- **0 pts** Correct