1 2F-1 Bookkeeping

- **0 pts** Correct

Exercise 2F-2. Mathematical Induction [5 points]. -

Proof: Let F be the set of all flowers and let $\mathsf{smells}(f)$ be the smell of the flower $f \in F$. (The range of smells is not so important, but we'll assume that it admits equality.) We'll also assume that F is countable. Let the property $P(n)$ mean that all subsets of F of size at most n contain flowers that smell the same.

$$P(n) \stackrel{\text{def}}{=} \forall X \in \mathcal{P}(F).\ |X| \leq n \implies (\forall f, f' \in X.\ \mathsf{smells}(f) = \mathsf{smells}(f'))$$

(the notation $|X|$ denotes the number of elements of X)

One way to formulate the statement to prove is $\forall n \geq 1. P(n)$. We'll prove this by induction on n, as follows:

Base Case: $n = 1$. Obviously all singleton sets of flowers contain flowers that smell the same (by the definition of $P(n)$).

Induction Step: 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.

This proof falls apart during the inductive step with values $n = 1$ and $n + 1 = 2$. In this situation, where $|X| = 2$, the set Y would simply be $\{f'\}$ and Y' would be $\{f\}$. This makes the set $Y \cap Y'$ would be empty. We wouldn't be able to choose an element x in this case, and the highlighted sentence impossible to prove.

2 2F-2 Mathematical Induction

- **0 pts** Correct

Exercise 2F-3. While Induction [10 points]. -

Proof:

For any BExp b and any initial state σ where $\sigma(x)$ is even, we will prove that if

$\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma'$

then $\sigma'(x)$ is even. We will prove this by induction on the structure of the derivation.

Base Case:

$$D :: \frac{D_1 :: \langle b,\ \sigma\rangle \Downarrow false}{\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma}$$

This will be our base case since this rule has no subderivations we can induct on. In this rule, the BExp b evaluates to $false$. Because of that, the state σ does not change from the *while* command. Therefore, $\sigma(x)$ will remain even and the property holds for the base case.

Inductive Case:

$$D :: \frac{D_1 :: \langle b,\ \sigma\rangle \Downarrow true \quad D_2 :: x := x + 1, \sigma \Downarrow \sigma'' \quad D_3 :: \langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma''\rangle \Downarrow \sigma'}{\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma'}$$

This is the rule for when b evaluates to $true$.

In D_2, x's value is increased by 2, and this results in state σ'' ($\sigma'' == \sigma[x := x + 2]$). By general math rules, adding an even number to an even number results in a sum that is also even. In this situation, since $\sigma(x)$ is even and 2 is even the addition of $x + 2$ will also be even. Which means that $\sigma''(x)$ will be even.

Because σ'' is used for D_3, and because $\sigma''(x)$ is even, we can apply the inductive step and assume that the property holds for D_3. In other words $\sigma'(x)$ will be even, which proves the inductive step and finishes the proof.

3 2F-3 While Induction

- **0 pts** Correct

Exercise 2F-4. Language Features, Large-Step [12 points]. -

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

The above rule is for the throw command. There is only one possible inference rule, one possible outcome, where the expression e is evaluated and the resulting value n is thrown. Because the state doesn't change, the exception gets thrown with the original state σ

$$\frac{\langle c_1,\ \sigma\rangle \Downarrow \sigma'}{\langle \mathit{try}\ c_1\ \mathit{catch}\ x\ c_2,\ \sigma\rangle \Downarrow \sigma'}$$

The above rule is the first for the try command. This is the rule for when c_1 completes without throwing an exception. In this situation, the try command will also terminate normally with the state σ' that c_1 ended with.

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

The above rule is the second for try. In this rule, c_1 throws an exception. When that happens, the next step is to assign the value n that c_1 threw to x, which creates a new state $\sigma'[x := n]$, and then execute c_2 with this new state. However c_2 terminates, either successfully or with an exception, is how try will also terminate.

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

The above rule is the first for finally. In this rule, c_1 executes normally. In this situation, c_2 is executed and however c_2 terminates is how finally will terminate.

$$\frac{\langle c_1,\ \sigma\rangle \Downarrow \sigma'\ \texttt{exc}\ e_1 \quad \langle c_2,\ \sigma'\rangle \Downarrow \sigma''}{\langle \mathit{after}\ c_1\ \mathit{finally}\ c_2,\ \sigma\rangle \Downarrow \sigma''\ \texttt{exc}\ e_1}$$

The above rule is the second for finally. Here, c_1 throws an exception and c_2 executes normally. With this, finally will throw an exception with the e_1 value that c_1 threw, and the σ'' state that c_2 terminated with.

$$\frac{\langle c_1,\ \sigma\rangle \Downarrow \sigma'\ \texttt{exc}\ e_1 \quad \langle c_2,\ \sigma'\rangle \Downarrow \sigma''\ \texttt{exc}\ e_2}{\langle \mathit{after}\ c_1\ \mathit{finally}\ c_2,\ \sigma\rangle \Downarrow \sigma''\ \texttt{exc}\ e_2}$$

The above rule is the third for finally. Here, both c_1 and c_2 throw exceptions. In this situation, finally will terminate with the e_2 value that c_2 threw and the state σ'' from c_2 as well.

4 2F-4 Language Features, Large Step

- **0 pts** Correct

Exercise 2F-5. Language Features, Analysis [6 points]. Argue for or against the claim that it would be more natural to describe "IMP with exceptions" using small-step contextual semantics. You may use "simpler" or "more elegant" instead of "more natural" if you prefer. Do not exceed two paragraphs (one should be sufficient). Both your ideas and also the clarity with which they are expressed (i.e., your English prose) matter.

I am going to argue that it would be much more natural/elegant to represent "IMP with exceptions" using large-step semantics as opposed to small-step semantics. The entire exception throwing system is built on a hierarchy. Exceptions are thrown up the levels of a hierarchy until it reaches a level that catches it. And since each level has multiple different branches, this hierarchy becomes a tree-structure. Large-step semantics already follow a tree structure, which will make it easy to trace exactly where the exception will end up getting caught and what will be done with it. With small-step semantics, the rules are listed out as a List, which will make it much harder to follow the hierarchy for where the exception will end up getting caught.

5 2F-5 Language Features, Analysis

- **0 pts** Correct