1 2F-1 Bookkeeping

- **0 pts** Correct

Exercise 2F-2. Mathematical Induction

Exercise 2F-2. Mathematical Induction [5 points]. Find the flaw in the following inductive proof that "All flowers smell the same". Please indicate exactly which sentences are wrong in the proof via highlighting or underlining.

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.

(One indication that the proof might be wrong is the large number of occurrences of the word "obviously" :-))

Peer Review ID: 65773515 — enter this when you fill out your peer evaluation via gradescope

2 2F-2 Mathematical Induction

- **0 pts** Correct

Exercise 2F-3. While Induction

To prove ∀sigma, sigma1 ∈ Σ. <while b do x := x + 2, sigma> ⇓ sigma1 ⇒ sigma1(x) is even.
Proof: by induction on the structure of the derivation D.
Reasoning by inversion on the derivation rules, we notice that the only last step in derivation of <while b do x := x + 2, sigma> ⇓ sigma1 must be while true and while false rules.

Case: Last rule used in D was the one for while false

$$D:: \frac{D1::<b, sigma> \Downarrow false}{<while\ b\ do\ x := x + 2, sigma> \Downarrow sigma}$$

By inversion, this means that sigma(x) is initially even and boolean expression b is evaluated to false. Therefore, sigma1(x) is also even. Because there is no sub-derivation, this is a base case in the induction.

Case: Last rule used in D was the one for while true

$$D:: \frac{D1::<b, sigma> \Downarrow true \quad D2::<x := x + 2, sigma> \Downarrow t\_sigma \quad D3::<while\ b\ do\ x := x + 2, t\_sigma> \Downarrow sigma1}{<while\ b\ do\ x := x + 2, sigma> \Downarrow sigma1}$$

By inversion, this means that boolean expression b is evaluated to true and therefore x is assigned to the value of x + 2. Initially, sigma(x) was even, so the D2 will make sigma1(x) as even, and it inducts on D3 again. Because there are sub-derivations for this rule, this is an inductive case in the induction.

3 2F-3 While Induction

- **0 pts** Correct

Exercise 2F-4. Language Features, Large-Step

Throw *e* Command:

$$\frac{<e, sigma> \Downarrow n}{<throw\ e, sigma> \Downarrow sigma\ exc\ n}$$

Try *c1* catch *x c2* Command: If c1 terminates normally

$$\frac{<c1, sigma> \Downarrow sigma1}{<try\ c1\ catch\ x\ c2, sigma> \Downarrow sigma1}$$

Try *c1* catch *x c2* Command: If c1 raises an exception with value e

$$\frac{<c1, sigma> \Downarrow sigma1\ exc\ n \quad <x := n; c2, sigma1> \Downarrow t}{<try\ c1\ catch\ x\ c2, sigma> \Downarrow t}$$

After *c1* finally *c2* Command: If c1 terminates normally

$$\frac{<c1, sigma> \Downarrow sigma1 \quad <c2, sigma1> \Downarrow t}{<after\ c1\ finally\ c2, sigma> \Downarrow t}$$

After *c1* finally *c2* Command: If c1 raises an exception with e1, and if c2 terminates normally

$$\frac{<c1, sigma> \Downarrow sigma1\ exc\ n1 \quad <c2, sigma1> \Downarrow sigma2}{<after\ c1\ finally\ c2, sigma> \Downarrow sigma2\ exc\ n1}$$

After *c1* finally *c2* Command: If c1 raises exception with e1, and if c2 throws exception with e2

$$\frac{<c1, sigma> \Downarrow sigma1\ exc\ n1 \quad <c2, sigma1> \Downarrow sigma2\ exc\ n2}{<after\ c1\ finally\ c2, sigma> \Downarrow sigma2\ exc\ n2}$$

4 2F-4 Language Features, Large Step

- **0 pts** Correct

Exercise 2F-5. Language Feature, Analysis

I would argue that small-step operational semantics are necessarily not more natural to describe “IMP with exceptions” than big-step operational semantics. One primary reason why small-step operational semantics are not simpler than big-step for “IMP with exceptions” is that it is intuitively straightforward to construct big-step for “IMP with exceptions” compared to small-step. For example, try...catch has two possible data flows, and after...finally has three possible data flows. Because execution of these exception commands have branching structure of output path, which requires recursion, it is hard to show in a single sequence line of state while reducing in an atomic step, which makes it hard to construct a small-step for “IMP with exceptions”. Therefore, small-step operational semantics are not more natural to describe “IMP with exceptions” than big-step operational semantics.

5 2F-5 Language Features, Analysis

- **0 pts** Correct