1 2F-1 Bookkeeping

- 0 pts Correct

2 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. <u>Pick any arbitrary $x \in Y \cap Y'$. Obviously, $x \neq f$ and $x \neq f'$.</u> 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.

The assumption of there exists an $x \in Y \cap Y'$ such that $x \neq f$ and $x \neq f'$ is not always valid.

In particular, when n = 2, no x can be found. The induction hypothesis does not guarantee that this set is non-empty.

Since the induction step is incorrect, the proof is not correct.

2 2F-2 Mathematical Induction

- 0 pts Correct

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

Base case: while false
If the last rule used in D is **while false**

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

We can infer that $\sigma' = \sigma$. Therefore, $\sigma'(x) = \sigma(x)$ is even.
Inductive case: while true
If the last rule used in D is **while true**

$$\frac{< b, \sigma > \Downarrow true \quad < x := x + 2, \sigma > \Downarrow \sigma'' \quad D_1 :: < \text{while } b \text{ do } x := x + 2, \sigma'' > \Downarrow \sigma'}{< \text{while } b \text{ do } x := x + 2, \sigma > \Downarrow \sigma'}$$

Structure of derivations using the rules for assignment

$$\frac{\frac{<x,\sigma>\Downarrow\sigma(x) \quad <2,\sigma> \Downarrow 2}{<x+2,\sigma>\Downarrow\sigma(x)+2}}{< x := x + 2, \sigma > \Downarrow \sigma[x := \sigma(x) + 2]}$$

Apply the induction hypothesis to the derivations rooted at D_1.
We can infer that $\sigma'' = \sigma[x := \sigma(x) + 2]$. The induction hypothesis is that if $\sigma''(x)$ is even, $\sigma'(x)$ in D_1 is even. Based on mathematical property of even number, $\sigma''(x)$ is already even since $\sigma(x)$ is even.
Q.E.D

3 2F-3 While Induction

- 0 pts Correct

Peer Review ID: 65790759 — enter this when you fill out your peer evaluation via gradescope

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

Notice that we are using t to represent terminations.

throw e

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

try c_1 catch x c_2

$$\frac{< c_1, \sigma > \Downarrow \sigma'}{< \text{try } c_1 \text{ catch } x \ \ c_2, \sigma > \Downarrow \sigma'}$$

$$\frac{< c_1, \sigma > \Downarrow \sigma' \text{ exc n} \quad < x := n; c_2, \sigma' > \Downarrow t}{< \text{try } c_1 \text{ catch } x \ \ c_2, \sigma > \Downarrow t}$$

after c_1 finally c_2

$$\frac{< c_1, \sigma > \Downarrow \sigma' \quad < c_2, \sigma' > \Downarrow t}{< \text{after } c_1 \text{ finally } c_2, \sigma > \Downarrow t}$$

$$\frac{< c_1, \sigma > \Downarrow \sigma' \text{ exc n} \quad < c_2, \sigma' > \Downarrow \sigma''}{< \text{after } c_1 \text{ finally } c_2, \sigma > \Downarrow \sigma'' \text{ exc n}}$$

$$\frac{< c_1, \sigma > \Downarrow \sigma' \text{ exc } n_1 \quad < c_2, \sigma' > \Downarrow \sigma'' \text{ exc } n_2}{< \text{after } c_1 \text{ finally } c_2, \sigma > \Downarrow \sigma'' \text{ exc } n_2}$$

5 Exercise 2F-5. Language Features, Analysis [6 points].

I believe it is simpler and more elegant to describe "IMP with exceptions" using small-step contextual semantics. For the try command, we must consider if c_1 raises an exception. For the finally command, both c_1 and c_2 will be evaluated. As a result, large-step semantics involves several evaluations for states σ. For most of the times, it is merely repeating the logic of throw or skip. We have to use multiple subscripts and superscripts to express and analysis the conditions. It is hard to read and not informative.

However, in small-step semantic, the evaluations can be done by simply referring to the skip and throw command in local reduction rules. Even though few more lines for Context and Redex will be added. It is intuitive and not repetitive. Here, small-step semantic can directly demonstrate the logics without worrying about the detailed expressions. Therefore, small-step contextual semantics would be more elegant and simpler for "IMP with exceptions".

4 2F-4 Language Features, Large Step

- 0 pts Correct

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

Notice that we are using t to represent terminations.

throw e

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

try c_1 catch x c_2

$$\frac{< c_1, \sigma > \Downarrow \sigma'}{< \text{try } c_1 \text{ catch } x \ \ c_2, \sigma > \Downarrow \sigma'}$$

$$\frac{< c_1, \sigma > \Downarrow \sigma' \text{ exc n} \quad < x := n; c_2, \sigma' > \Downarrow t}{< \text{try } c_1 \text{ catch } x \ \ c_2, \sigma > \Downarrow t}$$

after c_1 finally c_2

$$\frac{< c_1, \sigma > \Downarrow \sigma' \quad < c_2, \sigma' > \Downarrow t}{< \text{after } c_1 \text{ finally } c_2, \sigma > \Downarrow t}$$

$$\frac{< c_1, \sigma > \Downarrow \sigma' \text{ exc n} \quad < c_2, \sigma' > \Downarrow \sigma''}{< \text{after } c_1 \text{ finally } c_2, \sigma > \Downarrow \sigma'' \text{ exc n}}$$

$$\frac{< c_1, \sigma > \Downarrow \sigma' \text{ exc } n_1 \quad < c_2, \sigma' > \Downarrow \sigma'' \text{ exc } n_2}{< \text{after } c_1 \text{ finally } c_2, \sigma > \Downarrow \sigma'' \text{ exc } n_2}$$

5 Exercise 2F-5. Language Features, Analysis [6 points].

I believe it is simpler and more elegant to describe "IMP with exceptions" using small-step contextual semantics. For the try command, we must consider if c_1 raises an exception. For the finally command, both c_1 and c_2 will be evaluated. As a result, large-step semantics involves several evaluations for states σ. For most of the times, it is merely repeating the logic of throw or skip. We have to use multiple subscripts and superscripts to express and analysis the conditions. It is hard to read and not informative.

However, in small-step semantic, the evaluations can be done by simply referring to the skip and throw command in local reduction rules. Even though few more lines for Context and Redex will be added. It is intuitive and not repetitive. Here, small-step semantic can directly demonstrate the logics without worrying about the detailed expressions. Therefore, small-step contextual semantics would be more elegant and simpler for "IMP with exceptions".

5 2F-5 Language Features, Analysis

- 0 pts Correct