1 2F-1 Bookkeeping

- **0 pts** Correct

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" `:-)`)

Answer: The false assumption here is that the set $Y \cap Y'$ will always be non empty. This (obviously) doesn't hold when $n = 1$ and the induction is on $n = 2$. After removing f and f', $Y \cap Y' = \emptyset$ and x isn't defined.

Exercise 2F-3. While Induction [10 points]. Prove by induction the following statement about the operational semantics:

For any BExp b and any initial state σ such that $\sigma(x)$ is even, if

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

then $\sigma'(x)$ is even. Make sure you state what you induct on, what the base case is and what the inductive cases are. Show representative cases among the latter. Do not do a proof by mathematical induction!

Answer: By induction on the derivation of `while`.

2 2F-2 Mathematical Induction

- **0 pts** Correct

Proof. Given any σ where $even(\sigma(x))$ holds, we prove $\sigma(x) \wedge \langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma' \implies even(\sigma'(x))$ inductively.

Base Case: Consider the derivation D produced by the `while false` rule:

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

Since $\sigma = \sigma$, $even(\sigma(x))$ holds by the definition.

Inductive step: The only other applicable rule as the last step of the derivation is the `while true` rule, so consider the derivation D:

$$\text{D::}\ \frac{D_1 :: \langle b, \sigma\rangle \Downarrow true \qquad D_2 :: \langle x := x + 2, \sigma\rangle \Downarrow \sigma_1 \qquad D_3 :: \langle while\ b\ do\ x := x + 2, \sigma_1\rangle \Downarrow \sigma'}{\langle while\ b\ do\ x := x + 2, \sigma\rangle \Downarrow \sigma'}$$

By the property of the natural numbers, we know that $x := x + 2$ will leave x even if it was initially even. We know by the definition of σ that x is initially even. Then by combination of these $\sigma_1(x)$ must be even as well. By the inductive hypothesis on D_3, and the fact that $\sigma_1(x)$ is even, $\sigma'(x)$ must also be even. □

Here is a representative case for the inductive step:

Let $\sigma = \{x = 2\}$ and $b = x < 4$. Then the `when true` case is used since $2 \leq 4$ evaluates to true. Then the update results in $\sigma' = \{x = 4\}$. We see that $even(\sigma')$ still holds here. One more iteration of the loop results in $\sigma'' = \{x = 6\}$. Finally, the derivation reaches the base case and the `while false` rule produces σ'' which remains even.

Exercise 2F-4. Language Features, Large-Step [12 points]. We extend IMP with a notion of integer-valued *exceptions* (or *run-time errors*), as in Java, ML or C#. We introduce a new type T to represent command terminations, which can either be normal or exceptional (with an exception value $n \in \mathbb{Z}$):

T	::=	σ	"normal termination"
	|	σ `exc` n	"exceptional termination"

We use t to range over possible terminations T. We then redefine our operational semantics judgment:

$$\langle c, \sigma\rangle \Downarrow T$$

The interpretation of

$$\langle c, \sigma\rangle \Downarrow \sigma'\ \texttt{exc}\ n$$

is that command c terminated abruptly by throwing an exception with value $n \in \mathbb{Z}$ at a point in c's execution when the state was σ'. We only model one type of exception, but every exception has an integer "argument" n (or "payload" or "value") that is set when the exception is thrown and available when the exception is caught.

3 2F-3 While Induction

- **0 pts** Correct

Answer:

$$\textsc{Throw}\ \frac{\langle e, \sigma\rangle \Downarrow n}{\langle \mathit{throw}\ e, \sigma\rangle \Downarrow \sigma\ \mathit{exc}\ n} \qquad \textsc{Try-norm}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \mathit{try}\ c_1\ \mathit{catch}\ x\ c_2\rangle \Downarrow \sigma'}$$

$$\textsc{Try-ex}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathit{exc}\ e \qquad \langle x := e, \sigma'\rangle \Downarrow \sigma'' \qquad \langle c_2, \sigma''\rangle \Downarrow t}{\langle \mathit{try}\ c_1\ \mathit{catch}\ x\ c_2\rangle \Downarrow t}$$

$$\textsc{after-norm}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \qquad \langle c_2, \sigma'\rangle \Downarrow t}{\langle \mathit{after}\ c_1\ \text{finally}\ c_2, \sigma\rangle \Downarrow t}$$

$$\textsc{after-ex1}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathit{exc}\ e \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''}{\langle \mathit{after}\ c_1\ \text{finally}\ c_2, \sigma\rangle \Downarrow \langle \text{throw}\ e, \sigma''\rangle}$$

$$\textsc{after-ex2}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathit{exc}\ e_1 \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''\ \mathit{exc}\ e_2}{\langle \mathit{after}\ c_1\ \text{finally}\ c_2, \sigma\rangle \Downarrow \langle \text{throw}\ e_2, \sigma''\rangle}$$

Exercise 2F-4. 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.

Answer

It would not be more elegant to describe IMP-with-exceptions using small step semantics. Small-step enables the language designer to describe the smaller, one-step steps taken to arrive at a given term. This is often useful when the intermediary steps are "interesting", more precisely, when expressing these steps provides useful insights about the execution process. In such a case, the designer sacrifices the succinctness of the semantics to elucidate the meaning of the rules at a finer granularity and ensure that the details are clear to the reader. However, IMP-with-exception does not have any particularly interesting single-step operations. For example, the small-step granularity for specifying how an exception is thrown for the `after-finally` construct adds nothing interesting that the reader cannot get by observing the big-step semantics. Perhaps this is in part because the constructs in IMP are very familiar to us and don't introduce anything surprisingly new. In any case, big-step provides the useful information without removing any of the interesting details, and so it is the more elegant choice for the semantics of IMP-with-exceptions.

4 2F-4 Language Features, Large Step

- **0 pts** Correct

Answer:

$$\textsc{Throw}\ \frac{\langle e, \sigma\rangle \Downarrow n}{\langle \mathit{throw}\ e, \sigma\rangle \Downarrow \sigma\ \mathit{exc}\ n} \qquad \textsc{Try-norm}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \mathit{try}\ c_1\ \mathit{catch}\ x\ c_2\rangle \Downarrow \sigma'}$$

$$\textsc{Try-ex}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathit{exc}\ e \qquad \langle x := e, \sigma'\rangle \Downarrow \sigma'' \qquad \langle c_2, \sigma''\rangle \Downarrow t}{\langle \mathit{try}\ c_1\ \mathit{catch}\ x\ c_2\rangle \Downarrow t}$$

$$\textsc{after-norm}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \qquad \langle c_2, \sigma'\rangle \Downarrow t}{\langle \mathit{after}\ c_1\ \text{finally}\ c_2, \sigma\rangle \Downarrow t}$$

$$\textsc{after-ex1}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathit{exc}\ e \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''}{\langle \mathit{after}\ c_1\ \text{finally}\ c_2, \sigma\rangle \Downarrow \langle \text{throw}\ e, \sigma''\rangle}$$

$$\textsc{after-ex2}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathit{exc}\ e_1 \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''\ \mathit{exc}\ e_2}{\langle \mathit{after}\ c_1\ \text{finally}\ c_2, \sigma\rangle \Downarrow \langle \text{throw}\ e_2, \sigma''\rangle}$$

Exercise 2F-4. 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.

Answer

It would not be more elegant to describe IMP-with-exceptions using small step semantics. Small-step enables the language designer to describe the smaller, one-step steps taken to arrive at a given term. This is often useful when the intermediary steps are "interesting", more precisely, when expressing these steps provides useful insights about the execution process. In such a case, the designer sacrifices the succinctness of the semantics to elucidate the meaning of the rules at a finer granularity and ensure that the details are clear to the reader. However, IMP-with-exception does not have any particularly interesting single-step operations. For example, the small-step granularity for specifying how an exception is thrown for the `after-finally` construct adds nothing interesting that the reader cannot get by observing the big-step semantics. Perhaps this is in part because the constructs in IMP are very familiar to us and don't introduce anything surprisingly new. In any case, big-step provides the useful information without removing any of the interesting details, and so it is the more elegant choice for the semantics of IMP-with-exceptions.

5 2F-5 Language Features, Analysis

- **0 pts** Correct