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" `:-)`)

- **I highlighted the above sentence because, by introducing a set of $n+1$ flowers, the "inductive chain" is broken, and thus the inductive hypothesis does not hold. Up to this point, we've only said provable things about subsets of n flowers, so for all we know, the $n+1$-th flower might break something. Later, when the sets Y and Y' are picked, we have shuffled this mysterious $n+1$-th flower into one—or both—of the sets! Since we know nothing about this new flower, we can no longer claim the induction hypothesis is true, since we have not shown that it smells like any other flower. Otherwise, the argument would be circular because we would then have to assume the original subset of size $n+1$ already contains flowers that all smell the same, and we would have assumed the conclusion.**

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

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

2 2F-2 Mathematical Induction

- **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" `:-)`)

- **I highlighted the above sentence because, by introducing a set of $n+1$ flowers, the "inductive chain" is broken, and thus the inductive hypothesis does not hold. Up to this point, we've only said provable things about subsets of n flowers, so for all we know, the $n+1$-th flower might break something. Later, when the sets Y and Y' are picked, we have shuffled this mysterious $n+1$-th flower into one—or both—of the sets! Since we know nothing about this new flower, we can no longer claim the induction hypothesis is true, since we have not shown that it smells like any other flower. Otherwise, the argument would be circular because we would then have to assume the original subset of size $n+1$ already contains flowers that all smell the same, and we would have assumed the conclusion.**

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.

Assume there is a derivation

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

Now choose some BExp b and σ where $\sigma(x)$ is even.
Base Case: $x := x + 2$

$$\frac{\text{Known fact about integers}}{\langle x := x + 2, \sigma\rangle \Downarrow \sigma'(x) \text{ even}}$$

BExp's are not invertible, but we know that in big-step semantics, b must eventually evaluate to either `true` or `false`.

Inductive Case 1: b is `false`

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

$\sigma(x)$ is even because we assumed it.

Inductive Case 2: b is `true`

$$\frac{D_1 :: \dfrac{}{\langle b, \sigma\rangle \Downarrow \texttt{true}} \quad D_2 :: \dfrac{D_3 :: \dfrac{\text{base case}}{\langle x := x + 2, \sigma\rangle \Downarrow \sigma'(x) \text{ even}} \quad D_4 :: \langle \texttt{while } b \texttt{ do } x := x + 2, \sigma'\rangle \Downarrow \sigma''}{\langle x := x + 2; \texttt{while } b \texttt{ do } x := x + 2,\ \sigma\rangle \Downarrow \sigma''(x) \text{ even}}}{\langle \texttt{while } b \texttt{ do } x := x + 2,\ \sigma\rangle \Downarrow \sigma''(x) \text{ even}}$$

- D_1: Assumption
- D_2: This is expanding the rule of `while true`
- D_3: Calling the base case
- D_4: Induct on this since it very closely resembles our original assumption. We assume the inductive hypothesis, that is, we assume this keeps x an even number.

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

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

(Assuming that the argument e to `throw` can be any arithmetic expression that could evaluate to an integer)

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

3 2F-3 While Induction

- **0 pts** Correct

then $\sigma'(x)$ is even.

Assume there is a derivation

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

Now choose some BExp b and σ where $\sigma(x)$ is even.
Base Case: $x := x + 2$

$$\frac{\text{Known fact about integers}}{\langle x := x + 2, \sigma\rangle \Downarrow \sigma'(x) \text{ even}}$$

BExp's are not invertible, but we know that in big-step semantics, b must eventually evaluate to either `true` or `false`.

Inductive Case 1: b is `false`

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

$\sigma(x)$ is even because we assumed it.

Inductive Case 2: b is `true`

$$\frac{D_1 :: \dfrac{}{\langle b, \sigma\rangle \Downarrow \texttt{true}} \quad D_2 :: \dfrac{D_3 :: \dfrac{\text{base case}}{\langle x := x + 2, \sigma\rangle \Downarrow \sigma'(x) \text{ even}} \quad D_4 :: \langle \texttt{while } b \texttt{ do } x := x + 2, \sigma'\rangle \Downarrow \sigma''}{\langle x := x + 2; \texttt{while } b \texttt{ do } x := x + 2,\ \sigma\rangle \Downarrow \sigma''(x) \text{ even}}}{\langle \texttt{while } b \texttt{ do } x := x + 2,\ \sigma\rangle \Downarrow \sigma''(x) \text{ even}}$$

- D_1: Assumption
- D_2: This is expanding the rule of `while true`
- D_3: Calling the base case
- D_4: Induct on this since it very closely resembles our original assumption. We assume the inductive hypothesis, that is, we assume this keeps x an even number.

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

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

(Assuming that the argument e to `throw` can be any arithmetic expression that could evaluate to an integer)

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

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

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

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

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

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.

In my opinion, I believe it is actually more natural for IMP with exceptions to be described in large-step rather than small-step semantics. The reason for this is that large-step expresses programs in an "eventually something happens" fashion, which seems to more readily capture the essence of runtime errors. Small-step, on the other hand, tends to be more direct and explicit, and while it certainly would have no trouble with exceptions, it almost implies that exceptions are predictable, lacking the eventuality of large-step. Aside from high-level philosophical arguments, small-step seems to be more cumbersome to implement, especially when considering the `after-finally` rules, as it would require several different contexts and reduction rules, or some clever manipulation of temporary storage variables, in order to keep track of terminations and exit codes. Therefore, I believe that large-step seems more natural for handling IMP with exceptions.

4 2F-4 Language Features, Large Step

- **0 pts** Correct

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

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

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

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

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.

In my opinion, I believe it is actually more natural for IMP with exceptions to be described in large-step rather than small-step semantics. The reason for this is that large-step expresses programs in an "eventually something happens" fashion, which seems to more readily capture the essence of runtime errors. Small-step, on the other hand, tends to be more direct and explicit, and while it certainly would have no trouble with exceptions, it almost implies that exceptions are predictable, lacking the eventuality of large-step. Aside from high-level philosophical arguments, small-step seems to be more cumbersome to implement, especially when considering the `after-finally` rules, as it would require several different contexts and reduction rules, or some clever manipulation of temporary storage variables, in order to keep track of terminations and exit codes. Therefore, I believe that large-step seems more natural for handling IMP with exceptions.

5 2F-5 Language Features, Analysis

- **0 pts** Correct