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

Proof: Let F be the set of all flowers and let 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.

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

(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 smells(f) = smells(f'). Let $Y = X - \{f\}$, $\quad Y' = X - \{f'\}$. 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 smells(f') = smells(x) (from the induction hypothesis on Y) and smells(f) = smells(x) (from the induction hypothesis on Y'). Hence smells(f) = smells(f'), which proves the inductive step, and the theorem.

The flaw is that the proof-writer assumes that $Y \cap Y'$ is non-empty. In the case that $n = 1$, we want to show that $P(n+1)$ holds. However in this case $|X| = 2$, so $|Y| = |Y'| = 1$, however by the construction of Y and Y', the sole element in each of these sets is distinct from the element in the other. The rest of the proof fails because the proof-writer was using the fact that x exists to link that smells(f) = smells(f').

Note: I used ChatGPT to help re-format the text to match the homework instructions.

Question assigned to the following page: 3

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

Proof:
Want to show: For any BExp b and any state σ such that $\sigma(x)$ is even if:

$$\langle \text{while } b \text{ do } x := x + 2, \sigma \rangle \Downarrow \sigma' \tag{1}$$

Then $\sigma'(x)$ is also even.
I prove using *structural induction on the derivation tree of the given while expression* from (1).
Therefore, our induction hypothesis is that for any sub-derivation of:

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

if $\sigma''(x)$ is even, then $\sigma'''(x)$ is even (I use new values for sigma to distinguish from the states used above).

By our big-step operational semantics, we have:

$$\frac{\langle b, \sigma \rangle \Downarrow false}{\langle \text{while b do c}, \sigma \rangle \Downarrow \sigma}$$

and

$$\frac{\langle b, \sigma \rangle \Downarrow true \qquad \langle c; \text{while b do c}, \sigma \rangle \Downarrow \sigma'}{\langle \text{while b do c}, \sigma \rangle \Downarrow \sigma'}$$

Base Case:
Our base case is when our boolean $b = false$, since that is the statement that will serve as the last rule in the derivation tree of the while statement. In this case, since $\sigma(x)$ is even by our hypothesis, then $\sigma(x) = \sigma'(x)$ is also even in a trivial fashion.

Inductive Case:
The inductive case is when b evaluates to true. Then we use the other big-step rule for while, and our rule-instance is:

$$\frac{\langle b, \sigma \rangle \Downarrow true \qquad \langle x := x + 2; \text{while b do } (x := x + 2), \sigma \rangle \Downarrow \sigma'}{\langle \text{while b do c}, \sigma \rangle \Downarrow \sigma'}$$

The premise $\langle x := x+2; \text{while b do } (x := x+2), \sigma \rangle \Downarrow \sigma'$ has two sub-derivations:

$$\langle x := x + 2, \sigma \rangle \Downarrow \sigma''$$

and

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

Because $\sigma(x)$ is even, $\sigma''(x) = \sigma(x) + 2$ must also be even.

This leaves the second sub-derivation $\langle \text{while b do } (x := x + 2), \sigma'' \rangle \Downarrow \sigma'$. By our inductive hypothesis, if state σ'' is even, then σ' must also be even. This completes the inductive step, and completes the proof.

Question assigned to the following page: 4

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

$$\frac{\langle e, \sigma\rangle \Downarrow v}{\langle \text{throw } e, \sigma\rangle \Downarrow \sigma \text{ exc } v}$$

$$\frac{\langle c_1, \sigma\rangle \Downarrow T}{\langle \text{try } c_1 \text{ catch } x\, c_2, \sigma\rangle \Downarrow T}$$

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \text{ exc } n \quad \langle c_2, \sigma'[x := n]\rangle \Downarrow T}{\langle \text{try } c_1 \text{ catch } x\, c_2, \sigma\rangle \Downarrow T}$$

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

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

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \text{ exc } n \quad \langle c_2, \sigma'\rangle \Downarrow \sigma'' \text{ exc } m}{\langle \text{after } c_1 \text{ finally } c_2, \sigma\rangle \Downarrow \sigma'' \text{ exc } m}$$

Question assigned to the following page: 5

Exercise 2F-4. Language Features, Analysis [6 points]

Small-step contextual semantics would be simpler for introducing exceptions to IMP. As noted in the previous instructions, in large-step operational semantics all of our previous command rules must be updated. Furthermore, additional rules will need to be added to account for the command either executing normally or raising an exception at any point in execution. In contrast, small-step contextual semantics can allow an exception to be raised as a single reduction step. Then rules like *try-catch* and *finally* can handle the error or decide how the error propagates, while existing rules can largely be left unchanged.