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.

Answer:

" *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. "

The mistake being made here is that the author incorrectly assumes that $P(n+1)$ holds. We assume $P(n)$ holds, which informs us on subsets of F of size at most n. We have no guarantee on subsets of size $n+1$. In consequence, f or f' - which are drawn from X where $|X| = n+1$ - may never have been part of the subsets of F that respect $P(n)$, and cannot have any assumptions made about them. We therefore have no guarantee that $\mathsf{smells}(f') = \mathsf{smells}(x)$ or $\mathsf{smells}(f) = \mathsf{smells}(x)$, rendering this proof incorrect.

---

2 2F-2 Mathematical Induction

- **0 pts** Correct

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:
There are two judgments to that can be the base for our derivation. The first one is as follows, where $\langle b, \sigma\rangle \Downarrow \texttt{false}$:

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

We can see that by determinism, σ stays unchanged, and therefore $\sigma' = \sigma$. Therefore, we can conclude that if $\sigma(x)$ was even before the execution of the command, it will remain even after in σ'.
The second judgment we need to consider is the on where $\langle b, \sigma\rangle \Downarrow \texttt{true}$:

$$D_{true} :: \frac{D_a :: \langle b, \sigma\rangle \Downarrow \texttt{true} \quad D_b :: \langle x := x + 2, \sigma\rangle \Downarrow \sigma_1 \quad D_c :: \langle \textsf{while}\ b\ \textsf{do}\ x := x + 2, \sigma_1\rangle \Downarrow \sigma'}{\langle \textsf{while}\ b\ \textsf{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma'}$$

By the induction hypothesis on D_b, we can conclude that $\sigma_1 = \sigma[x := \sigma(x) + 2]$. Since we know that $\sigma(x)$ is even, we can conclude that $\sigma_1(x) = \sigma(x) + 2$ is also even.
Using this result, we can conclude by the induction hypothesis on D_c that $\sigma'(x)$ is also even.
Thus, for any BExp b and any initial state σ such that $\sigma(x)$ is even, if we run the while command, then $\sigma'(x)$ is even.

---

3 2F-3 While Induction

- **0 pts** Correct

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 $n \in \mathbb{Z}$):

$$\begin{array}{rcll} T & ::= & \sigma & \text{“normal termination”} \\ & | & \sigma \ \texttt{exc} \ n & \text{“exceptional termination”} \end{array}$$

Note that our previous command rules must be updated to account for exceptions, as in:

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \ \texttt{exc} \ n}{\langle c_1; c_2, \sigma\rangle \Downarrow \sigma' \ \texttt{exc} \ n} \ \textsf{seq1} \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \quad \langle c_2, \sigma'\rangle \Downarrow t}{\langle c_1; c_2, \sigma\rangle \Downarrow t} \ \textsf{seq2}$$

We also introduce three additional commands:

$$\begin{array}{l} \textsf{throw} \ e \\ \textsf{try} \ c_1 \ \textsf{catch} \ x \ c_2 \\ \textsf{after} \ c_1 \ \textsf{finally} \ c_2 \end{array}$$

Give the large-step operational semantics inference rules (using our new judgment) for the three new commands presented here. You should present six (6) new rules total.

Answer:

Throw:

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

Try:

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma_1}{\langle \textsf{try} \ c_1 \ \textsf{catch} \ x \ c_2, \sigma\rangle \Downarrow \sigma_1} \ \textsf{try}_1 \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma_1 \ \texttt{exc} \ n \quad \langle c_2, \sigma_1[x := n]\rangle \Downarrow \sigma_2}{\langle \textsf{try} \ c_1 \ \textsf{catch} \ x \ c_2, \sigma\rangle \Downarrow \sigma_2} \ \textsf{try}_2$$

Finally:

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma_1 \quad \langle c_2, \sigma_1\rangle \Downarrow \sigma_2}{\langle \textsf{after} \ c_1 \ \textsf{finally} \ c_2, \sigma\rangle \Downarrow \sigma_2} \ \textsf{finally}_1 \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma_1 \ \texttt{exc} \ n_1 \quad \langle c_2, \sigma_1\rangle \Downarrow \sigma_2}{\langle \textsf{after} \ c_1 \ \textsf{finally} \ c_2, \sigma\rangle \Downarrow \sigma_2 \ \texttt{exc} \ n_1} \ \textsf{finally}_2$$

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma_1 \ \texttt{exc} \ n_1 \quad \langle c_2, \sigma_1\rangle \Downarrow \sigma_2 \ \texttt{exc} \ n_2}{\langle \textsf{after} \ c_1 \ \textsf{finally} \ c_2, \sigma\rangle \Downarrow \sigma_2 \ \texttt{exc} \ n_2} \ \textsf{finally}_3$$

---

4 2F-4 Language Features, Large Step

- **0 pts** Correct

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.

Answer:

I do not believe that "IMP with exceptions" is better described using small-step contextual semantics for two main reasons: I believe large-step to be more clear, and I think large-step is easier to implement in our OCaml interpreter.

Large-step is more concise: the preconditions and conclusions to a judgment are presented next to each other. This makes the evaluation of a program crystal clear at the small-step intermediate calculations are hidden away to reveal the result only. This is particularly useful to newcomers to IMP, as they may more elegantly understand the relationships between commands in a program.

Moreover, I find that the translation between large-step operational semantics and OCaml is fricitonless: indeed, in my personal experience, IMP large-step rules have been extremely straightforward to implement in OCaml. This makes the creation and implementation of new language features (such as exceptions) easier when expressed in large-step operational semantics. I believe this to be a large advantage over small-step.

In conclusion, both my belief in the superior clarity of large-step operational semantics, as well as my personal experience translating large-step rules into the OCaml interpreter lead me to reject the claim that 'it would be more natural to describe "IMP with exceptions" using small-step contextual semantics'.

---

5 2F-5 Language Features, Analysis

- **0 pts** Correct