1 2F-1 Bookkeeping

- **0 pts** Correct

All subsequent answers should appear after the first page of your submission and may be shared publicly during peer review.

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" :-))

> $Y \cap Y' \neq \emptyset$ does not hold under our base case, so this induction proof can not prove the theorem.

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!

2 2F-2 Mathematical Induction

- **0 pts** Correct

Base case: while false
Case: the last rule used in D was the one for while false

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

This means $\sigma' = \sigma$, so $\sigma'(x) = \sigma(x)$, which is even.

Inductive case: while true
Case: the last rule used in D was the one for while true

$$\frac{\langle b, \sigma\rangle \Downarrow \texttt{true} \quad \langle x := x + 2, \sigma\rangle \Downarrow \sigma_1 \quad D_1 :: \langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma_1\rangle \Downarrow \sigma'}{\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma'}$$

With the rules for assignment,

$$\frac{\dfrac{\langle x, \sigma\rangle \Downarrow \sigma(x) \quad \langle 2, \sigma\rangle \Downarrow 2}{\langle x + 2, \sigma\rangle \Downarrow \sigma(x) + 2}}{\langle x := x + 2, \sigma\rangle \Downarrow \sigma[x := \sigma(x) + 2]}$$

This means $\sigma_1 = \sigma[x := \sigma(x) + 2]$. Given that $\sigma(x)$ is even, $\sigma_1(x)$ is even according to the nature of addition.
Our induction hypothesis on D_1 is that if $\sigma_1(x)$ is even, $\sigma'(x)$ in D_1 will be even.
By this induction hypothesis above, and since we've already derived that $\sigma_1(x)$ is even, according to the while true rule, $\sigma'(x)$ is 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 $n \in \mathbb{Z}$):

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

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

3 2F-3 While Induction

- **0 pts** Correct

to output "`33 18 3 -12`" and then terminate with an uncaught exception with value `-24`.

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.

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

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \mathsf{try}\ c_1\ \mathsf{catch}\ x\ c_2\rangle \Downarrow \sigma'} \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathtt{exc}\ n \quad \langle c_2, \sigma[x := n]\rangle \Downarrow t}{\langle \mathsf{try}\ c_1\ \mathsf{catch}\ x\ c_2\rangle \Downarrow t}$$

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

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

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.

I agree that small-step contextual semantics are more natural than large-step operational semantics when describing "IMP with exceptions". For example, for the "after-finally" case, it is shown obviously in small-step semantics that the value of c_1 is discussed first, then we decide if the value of c_2 will affect our derivation. However, in large-step semantics, we just mix all the cases together, and it's less trivial to realize the proper derivation flow in practice. Therefore, I believe in this case, small-step semantics are more natural in the aspect of implementation, while large-step semantics look more elegant in proofs.

Exercise 2C. Language Features, Coding. Download the Homework 2 code pack from the course web page. Modify `hw2.ml` so that it implements a complete interpreter for "IMP with exceptions (and `print`)". You may build on your code from Homework 1 (although the `let` command is not part of this assignment). Using OCaml's exception mechanism to implement IMP exceptions is actually slightly harder than doing it "naturally", so I recommend that you just implement the operational semantics rules. The `Makefile` includes a "`make test`" target that you should use (at least) to test your work.

Hint: to check if a termination `term` is an exception, use syntax like

```
begin match term with
| Normal -> do_something
```

4 2F-4 Language Features, Large Step

- **0 pts** Correct

to output "`33 18 3 -12`" and then terminate with an uncaught exception with value `-24`.

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.

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

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \mathsf{try}\ c_1\ \mathsf{catch}\ x\ c_2\rangle \Downarrow \sigma'} \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathtt{exc}\ n \quad \langle c_2, \sigma[x := n]\rangle \Downarrow t}{\langle \mathsf{try}\ c_1\ \mathsf{catch}\ x\ c_2\rangle \Downarrow t}$$

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

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

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.

I agree that small-step contextual semantics are more natural than large-step operational semantics when describing "IMP with exceptions". For example, for the "after-finally" case, it is shown obviously in small-step semantics that the value of c_1 is discussed first, then we decide if the value of c_2 will affect our derivation. However, in large-step semantics, we just mix all the cases together, and it's less trivial to realize the proper derivation flow in practice. Therefore, I believe in this case, small-step semantics are more natural in the aspect of implementation, while large-step semantics look more elegant in proofs.

Exercise 2C. Language Features, Coding. Download the Homework 2 code pack from the course web page. Modify `hw2.ml` so that it implements a complete interpreter for "IMP with exceptions (and `print`)". You may build on your code from Homework 1 (although the `let` command is not part of this assignment). Using OCaml's exception mechanism to implement IMP exceptions is actually slightly harder than doing it "naturally", so I recommend that you just implement the operational semantics rules. The `Makefile` includes a "`make test`" target that you should use (at least) to test your work.

Hint: to check if a termination `term` is an exception, use syntax like

```
begin match term with
| Normal -> do_something
```

5 2F-5 Language Features, Analysis

- **0 pts** Correct