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" :-))

Answer 2F-2 The sentence "Pick any arbitrary $x \in Y \cap Y'$" is wrong. The reason is, for inductive step in mathematical induction, we need prove $P(n) \implies P(n+1)$ for any $n > 0$. However, when $n = 1$, we have $|X| = n+1 = 2$. If we choose $f, f' \in X$ as above. $Y \cap Y' = \emptyset$. Thus, we cannot pick $x \in Y \cap Y'$. Therefore, the sentence "Pick any arbitrary $x \in Y \cap Y'$" is wrong.

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 2F-3 Let B be the set of all BExp.

We wish to show the property

$$P(\sigma) \stackrel{\text{def}}{=} \forall b \in B.\ \langle \texttt{while } b \texttt{ do } x := x + 2, \sigma\rangle \Downarrow \sigma' \implies \sigma'(x) \text{ is even}$$

holds for all states σ in $\{\sigma \in \Sigma | \sigma(x) \text{ is even}\}$.

We do this by well-founded induction on $S = \{\sigma \in \Sigma | \sigma(x) \text{ is even}\}$ where

$$\sigma_1 \prec \sigma_2 \iff \sigma_1(x) > \sigma_2(x)$$

for states σ_1, σ_2 in S. $\prec$ is well founded as the value of x cannot increased indefinitely.

Base case: $P(\sigma)$ is true for $\langle \texttt{while } \mathit{false} \texttt{ do } x := x + 2, \sigma\rangle \Downarrow \sigma' \implies \sigma'(x)$ is even, when $\sigma(x)$ is the largest even number that the computer can hold. (I assume that in this case $\langle \texttt{while } \mathit{true} \texttt{ do } x := x + 2, \sigma\rangle \Downarrow \sigma'$ does not exist, because the result cannot be handled by the computer.)

Inductive Step:

If b is **false**. We construct the derivation

$$\dfrac{\dfrac{\vdots}{<b,\sigma>\Downarrow \mathit{false}}}{<\texttt{while } b \texttt{ do } x:=x+2,\sigma>\Downarrow\sigma}$$

using the rule for while-loops which applies when the boolean condition evaluates to false. In this case, $\sigma' = \sigma$. Thus $\sigma'(x) = \sigma(x)$ is even.

If b is **true**. We construct the derivation

$$\dfrac{\dfrac{\vdots}{<b,\sigma>\Downarrow \mathit{true}} \quad <x:=x+2;\texttt{while } b \texttt{ do } x:=x+2,\sigma>\Downarrow\sigma'}{<\texttt{while } b \texttt{ do } x:=x+2,\sigma>\Downarrow\sigma'}$$

using the rule for while-loops which applies when the boolean condition evaluates to true. It also can be written as

$$\dfrac{\dfrac{\dfrac{\vdots}{<b,\sigma>\Downarrow \mathit{true}}}{\vdots}}{\dfrac{<\texttt{while } b \texttt{ do } x:=x+2,\sigma''>\Downarrow\sigma'}{<\texttt{while } b \texttt{ do } x:=x+2,\sigma>\Downarrow\sigma'}}$$

where $\sigma''(x) = \sigma(x) + 2$. In our definition $\sigma'' \prec \sigma$. If we have $P(\sigma'')$, then $P(\sigma)$.

By well-founded induction, we conclude $\forall \sigma \in S, P(\sigma)$, as required.

3 2F-3 While Induction

- **0 pts** Correct

These constructs are intended to have the standard exception semantics from languages like Java, C# or OCaml — except that the catch block merely assigns to x, it does not bind it to a local scope. So unlike Java, our catch does not behave like a let. We thus expect:

```
x := 0 ;
{ try
    if x <= 5 then throw 33 else throw 55
  catch x
    print x } ;
while true do {
  x := x - 15 ;
  print x ;
  if x <= 0 then throw (x*2) else skip
}
```

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.

Answer 2F-4. throw e command

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

.

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \mathsf{try}\ c_1\ \mathsf{catch}\ x\ c_2, \sigma\rangle \Downarrow \sigma'}\ \mathsf{catch1} \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathtt{exc}\ n \quad \langle x := n; c_2, \sigma'\rangle \Downarrow t}{\langle \mathsf{try}\ c_1\ \mathsf{catch}\ x\ c_2, \sigma\rangle \Downarrow t}\ \mathsf{catch2}$$

.

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \quad \langle c_2, \sigma'\rangle \Downarrow t}{\langle \mathsf{after}\ c_1\ \mathsf{finally}\ c_2, \sigma\rangle \Downarrow t}\ \mathsf{finally1} \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathtt{exc}\ n \quad \langle c_2, \sigma'\rangle \Downarrow \sigma''}{\langle \mathsf{after}\ c_1\ \mathsf{finally}\ c_2, \sigma\rangle \Downarrow \sigma''\ \mathtt{exc}\ n}\ \mathsf{finally2}$$

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \mathtt{exc}\ n_1 \quad \langle c_2, \sigma'\rangle \Downarrow \sigma''\ \mathtt{exc}\ n_2}{\langle \mathsf{after}\ c_1\ \mathsf{finally}\ c_2, \sigma\rangle \Downarrow \sigma''\ \mathtt{exc}\ n_2}\ \mathsf{finally3}$$

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.

4 2F-4 Language Features, Large Step

- **0 pts** Correct

Answer 2F-5. In my opinion, I disagree with that it would be more natural to describe"IMP with exceptions" using small-step contextual semantics. Small-step semantics give more details on the program. However, some details are not necessary for understand the exceptions. Big-step abstracts away the details of the evaluation but retains the syntactic nature of the result, so that we can know which block of commands cause the exception rather than go through the details about how the expressions are reduced. Small step contextual semantics help build an interpreter because it defines each operation. To analyse a real program, we cannot follow all the small-step reduction rules step by step. As human, we cannot retrieve and interpret instructions in gigahertz. What we can do is to understand how the overall results of the executions are obtained.

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
| Exceptional(n) -> do_something_else using n
end
```

Modify the file `example-imp-command` so that it contains a "tricky" terminating IMP command (presumably involving exceptions) that can be parsed by our IMP test harness (e.g., "`imp < example-imp-command`" should not yield a parse error).

Submission. Turn in the formal component of the assignment as a single PDF document via the `gradescope` website. Your name and Michigan email address must appear on the first page of your PDF submission but may not appear anywhere else. Turn in the coding component of the assignment via the `autograder.io` website.

5 2F-5 Language Features, Analysis

- **0 pts** Correct