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.

$$P(n) \overset{\text{def}}{=} \forall X \in \mathcal{P}(F).\ |X| \leq n \implies (\forall f, f' \in X.\ \mathsf{smells}(f) = \mathsf{smells}(f'))$$

(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" `:-)`)

Solution The highlighted sentence in the inductive step does not hold when $n = 1$. When $n = 1$, then $X = \{f, f'\}$. Then, $Y = X - \{f\} = \{f'\}$ and $Y' = X - \{f'\} = \{f\}$. So, $Y \cap Y' = \emptyset$. Thus, there is no x to pick in $Y \cap Y'$, so the rest of the proof is flawed.

Peer Review ID: 65688198 — enter this when you fill out your peer evaluation via gradescope

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!

Solution We will prove the given statement by performing induction on the structure of the derivation of $\langle \texttt{ while } b \texttt{ do } x := x + 2, \sigma \rangle \Downarrow \sigma'$.

Base Case: Suppose $\langle b, \sigma \rangle \Downarrow \mathit{false}$, $\sigma \in \Sigma$ such that $\sigma(x)$ is even, and we have a derivation $D :: \langle \texttt{ while } b \texttt{ do } x := x + 2, \sigma \rangle \Downarrow \sigma'$. By inversion and the determinism of Boolean expressions, D must use the evaluation rule for `while` false, which is

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

Thus, $D :: \langle \texttt{ while } b \texttt{ do } x := x + 2, \sigma \rangle \Downarrow \sigma$. By assumption, $\sigma(x)$ is even, so we are done.

Inductive Step: Suppose $\langle b, \sigma \rangle \Downarrow \mathit{true}$, $\sigma \in \Sigma$ such that $\sigma(x)$ is even, and we have a derivation $D :: \langle \texttt{ while } b \texttt{ do } x := x + 2, \sigma \rangle \Downarrow \sigma'$. By inversion and the determinism of Boolean expressions, D must use the evaluation rule for `while` true, which is

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

Suppose $\langle x + 2, \sigma \rangle \Downarrow n$. Since $\sigma(x)$ is even by assumption, then n is even as well. Then, by the evaluation rule of assignment, we have $\langle x := x + 2, \sigma \rangle \Downarrow \sigma_1 = \sigma[x := n]$, so $\sigma_1(x)$ is even. Then, we can apply the inductive hypothesis on D_1, so $\sigma'(x)$ is also even, so we are done.

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

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

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 every exception has an integer "argument" n (or "payload" or "value") that is set when the exception is thrown and available when the exception is caught.

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

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

We also introduce three additional commands:

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

- The throw e command raises an exception with argument e.
- The try command executes c_1. If c_1 terminates normally (i.e., without an uncaught exception), the try command also terminates normally. If c_1 raises an exception with value e, the variable $x \in L$ is assigned the value e and then c_2 is executed.
- The finally command executes c_1. If c_1 terminates normally, the finally command terminates by executing c_2. If instead c_1 raises an exception with value e_1, then c_2 is executed:
  - If c_2 terminates normally, the finally command terminates by throwing an exception with value e_1. (That is, the original exception e_1 is re-thrown at the end of the finally block, as in Java.)
  - If c_2 throws an exception with value e_2, the finally command terminates by throwing an exception with value e_2. (That is, the new exception e_2 overrides the original exception e_1, also as in Java.)

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.

Solution throw e:

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

try c_1 catch x c_2:

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

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

after c_1 finally c_2:

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

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

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

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.

Solution It is more natural to me to describe exceptions in IMP using large-step semantics. The exception commands involve the evaluation of sequences of subcommands, which is more clear and intuitive to me in large-step semantics. In large-step semantics, all the commands in the sequence can be evaluated recursively in parallel on a single line in the rule, clearly indicating how the evaluations of the subcommands affect the evaluation of the exception command. In small-step semantics, the commands need to be reduced separately in sequence, meaning each case of the exception command's behavior is represented by a whole sequence of steps instead of a single concise rule.

5 2F-5 Language Features, Analysis

- 0 pts Correct