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.

Reasoning for Highlighted Lines: The highlighted line claims all subsets of F size at most n have elements which all smell the same. This is assuming the proof. If all subsets of F size 2 smell the same, this means all possible pairs of flowers must smell the same. If we assume that all flowers in F smell the same, we can prove such a set of flowers has elements which all smell the same, obviously. Consider a set larger than size n with elements that do not smell the same:

$$X = \{f_1 - Sweet, f_2 - Sweet, f_3 - Sweet, f_4 - Disgusting, f_5 - Sweet\}$$

Assume n = 4. Then remove a single element (lets remove f_1.).

$$X = \{f_2 - Sweet, f_3 - Sweet, f_4 - Disgusting, f_5 - Sweet\}$$

This clearly contradicts our hypothesis, as it's a subset size **n** which all elements do not smell the same. The circular reasoning "assume **proof**, therefore **proof**" is the issue here.

Exercise 2F-3. While Induction [10 points]. There are two cases which we must consider because of the boolean, **b**. We consider the **true** case first.

$$D :: \frac{D1 :: \langle b, \sigma\rangle \Downarrow true \qquad D2 :: \langle x := x+2, \sigma\rangle \Downarrow \sigma' \quad D3 :: \langle \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma'\rangle \Downarrow \sigma''}{\langle \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma\rangle \Downarrow \sigma''}$$

2 2F-2 Mathematical Induction

- 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.

Reasoning for Highlighted Lines: The highlighted line claims all subsets of F size at most n have elements which all smell the same. This is assuming the proof. If all subsets of F size 2 smell the same, this means all possible pairs of flowers must smell the same. If we assume that all flowers in F smell the same, we can prove such a set of flowers has elements which all smell the same, obviously. Consider a set larger than size n with elements that do not smell the same:

$$X = \{f_1 - Sweet, f_2 - Sweet, f_3 - Sweet, f_4 - Disgusting, f_5 - Sweet\}$$

Assume n = 4. Then remove a single element (lets remove f_1.).

$$X = \{f_2 - Sweet, f_3 - Sweet, f_4 - Disgusting, f_5 - Sweet\}$$

This clearly contradicts our hypothesis, as it's a subset size **n** which all elements do not smell the same. The circular reasoning "assume **proof**, therefore **proof**" is the issue here.

Exercise 2F-3. While Induction [10 points]. There are two cases which we must consider because of the boolean, **b**. We consider the **true** case first.

$$D :: \frac{D1 :: \langle b, \sigma\rangle \Downarrow true \quad D2 :: \langle x := x+2, \sigma\rangle \Downarrow \sigma' \quad D3 :: \langle \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma'\rangle \Downarrow \sigma''}{\langle \texttt{while}\ b\ \texttt{do}\ x := x+2, \sigma\rangle \Downarrow \sigma''}$$

We need to consider the possible states σ and show that x **must** be even in each. Because integer arithmetic is deterministic and any number incremented by two is has an unchanged parity, we can use the induction hypothesis on D2 to prove that x in state σ' **must** be even. Because x is even and we are adding an even number to it, we know the result is even. Therefore x is even in state σ' (the property holds).

If we now apply the inductive hypothesis on D3 we know all further sigmas generated in the loop will be of the same form as D2, always resulting in an even number added to x, and x remains even. Therefore σ'' always has an even x. Because σ'' in D3 is the same as σ'' in our base case, we therefore know the "while true" case only returns x s.t. the parity is unchanged.

Next lets consider the case that b is false.

$$D :: \frac{D1 :: \langle b, \sigma\rangle \Downarrow false}{\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma}$$

Because sigma is always unchanged under this condition, x will remain unchanged. Therefore the parity will remain unchanged. We do not need to induct because we immediately see that D1 results in an unchanged state (there is only the base case).

Maybe a more satisfying proof, but equally valid,

$$D :: \frac{D1 :: \langle b, \sigma\rangle \Downarrow false \quad D2 :: \langle skip, \sigma\rangle \Downarrow \sigma}{\langle \texttt{while}\ b\ \texttt{do}\ x := x + 2, \sigma\rangle \Downarrow \sigma}$$

We can now use the inductive hypothesis on D2 and see that σ is unchanged and our property holds.

Proof Conclusion and Thoughts We have proven no parity change on the variable x occurs in any execution of the IMP command presented. This is stronger than proving the property, and our property is implied by the proof.

Exercise 2F-4. Language Features, Large-Step [12 points]. Lets begin with some notation. We no longer have a single terminal state sigma, but rather we use the **Terminal** type. Just as in the OCaml code, we have:

```
termination =
  | Normal      of sigma
  | Exceptional of sigma * n
```

But this syntax becomes burdensome, so we use the notation from the homework assignment, σ `exc` n, for exceptional termination, and use **t** for an uncertain termination (normal or exceptional). For each rule, I'll offer a little bit of reasoning in English to illustrate how I come up with the large-step semantics.

3 2F-3 While Induction

- 0 pts Correct

We need to consider the possible states σ and show that x **must** be even in each. Because integer arithmetic is deterministic and any number incremented by two is has an unchanged parity, we can use the induction hypothesis on D2 to prove that x in state σ' **must** be even. Because x is even and we are adding an even number to it, we know the result is even. Therefore x is even in state σ' (the property holds).

If we now apply the inductive hypothesis on D3 we know all further sigmas generated in the loop will be of the same form as D2, always resulting in an even number added to x, and x remains even. Therefore σ'' always has an even x. Because σ'' in D3 is the same as σ'' in our base case, we therefore know the "while true" case only returns x s.t. the parity is unchanged.

Next lets consider the case that b is false.

$$D \ :: \ \frac{D1 \ :: \ \langle b, \sigma\rangle \Downarrow false}{\langle \texttt{while } b \texttt{ do } x := x+2, \sigma\rangle \Downarrow \sigma}$$

Because sigma is always unchanged under this condition, x will remain unchanged. Therefore the parity will remain unchanged. We do not need to induct because we immediately see that D1 results in an unchanged state (there is only the base case).

Maybe a more satisfying proof, but equally valid,

$$D \ :: \ \frac{D1 \ :: \ \langle b, \sigma\rangle \Downarrow false \quad D2 \ :: \ \langle skip, \sigma\rangle \Downarrow \sigma}{\langle \texttt{while } b \texttt{ do } x := x+2, \sigma\rangle \Downarrow \sigma}$$

We can now use the inductive hypothesis on D2 and see that σ is unchanged and our property holds.

Proof Conclusion and Thoughts We have proven no parity change on the variable x occurs in any execution of the IMP command presented. This is stronger than proving the property, and our property is implied by the proof.

Exercise 2F-4. Language Features, Large-Step [12 points]. Lets begin with some notation. We no longer have a single terminal state sigma, but rather we use the **Terminal** type. Just as in the OCaml code, we have:

```
termination =
  | Normal      of sigma
  | Exceptional of sigma * n
```

But this syntax becomes burdensome, so we use the notation from the homework assignment, σ `exc` n, for exceptional termination, and use **t** for an uncertain termination (normal or exceptional). For each rule, I'll offer a little bit of reasoning in English to illustrate how I come up with the large-step semantics.

Throw e

This should result in an unchanged sigma with an uncaught exception derived from e.

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

$Try\ c_1\ Catch\ x\ c_2$

Here we always execute c_1 and produce a new sigma. depending on how it terminates we need two possible methods of "catching".

Normal execution catch execute c_1 and terminate normally.

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

Exceptional execution catch execute c_1 and terminate with an exception. Note that the lowercase t is a single instance of the termination type T, meaning that c2 either terminates normally or exceptionally.

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

$after\ c_1\ finally\ c_2$

There are three possible rules for this command. We always begin by executing c_1 and producing a new sigma.

Normal execution after Execute c_1 and terminate normally, execute c_2 and terminate with **t**.

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

Exceptional execution on c1 with normal c2 execute c_1 and throw an exception, then terminate c_2 normally so the original exception survives.

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

Exceptional execution on c1 with exceptional c2 execute c_1 and terminate with an exception then terminate c_2 exceptionally so the exception is overwritten.

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ exc\ n_0 \quad \langle c_2, \sigma'\rangle \Downarrow \sigma''\ exc\ n_1}{\langle after\ c_1\ finally\ c_2, \sigma\rangle \Downarrow \sigma''\ exc\ n_1}$$

4 2F-4 Language Features, Large Step

- **0 pts** Correct

Exercise 2F-4. Language Features, Analysis [6 points]. The key difference between small-step and large-step is in the notation, they support similar ideas in different formats. I'll argue notation is important to how we process ideas and that less complex and more visual notation is almost always preferred. I believe the large-step encoding of exceptions in **IMP** is more natural because it closely matches how one would implement the new concept, where the small-step in this case is less closely matched to how the language would be implemented. Further, the large-step version of exceptions are more concise, requiring less computation to describe and understand. The main distinction I want to isolate is the new notation introduced in the previous example. We gain the expressive termination type, **t**, and the exceptional termination σ *exc* n. This notation provide a clear description of the exception which is cohesive with our prior σ termination. The small-step doesn't seem to capture this with any elegance. Termination with "skip" and "throw n" is not as clear to me as the sigma state termination. While intuitively handled in large-step, the exception model feels more arbitrary in small-step. I'll motivate this with a simple example on throw.

```
H :=
    | throw H

r :=
    | throw n ; c
```

Local reduction rules:

$$\langle throw\ n;\ c,\ \sigma\rangle \to \langle throw\ n,\ \sigma\rangle$$

Where **throw n** is the exceptional termination of a program in **IMP**. For other operations I find a similar correspondence, where the rules in small-step are less direct than the large-step. Tracking exceptions using the "throw n" notation is cumbersome. I find a needless complexity here when compared to the large-step shown above. As a final point, see figure 1.

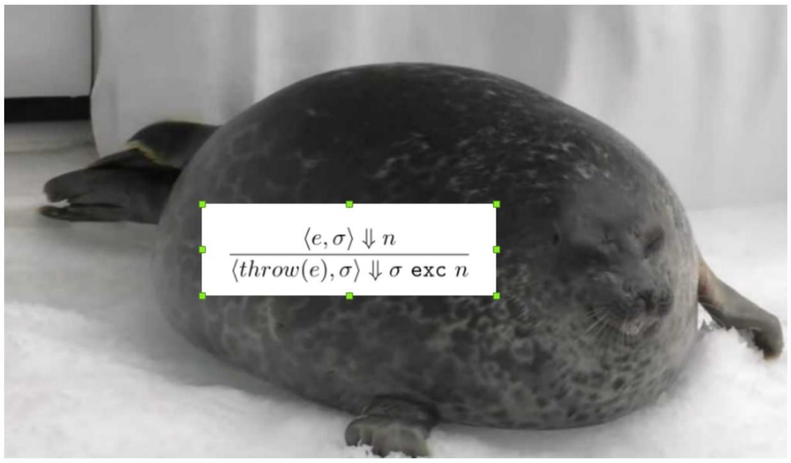
$\dfrac{\langle e, \sigma\rangle \Downarrow n}{\langle throw(e), \sigma\rangle \Downarrow \sigma\ \texttt{exc}\ n}$

Figure 1: Lörge

5 2F-5 Language Features, Analysis

- 0 pts Correct

Peer Review ID: 65765509 — enter this when you fill out your peer evaluation via gradescope