Exercise 2F-2. Mathematical Induction

Proof. Let F be the set of all flowers and let 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.

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

(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 smells(f) = 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 smells(f') = smells(x) (from the induction hypothesis on Y) and smells(f) = smells(x) (from the induction hypothesis on Y'). Hence smells(f) = smells(f'), which proves the inductive step, and the theorem.

Explanation: The intersection of Y and Y' may be empty and therefore we can't pick an arbitrary x. Thus any further statement that depends on x is wrong. □

Exercise 2F-3. While Induction

Proof. We aim to show that 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.

We will proceed by induction on the structure of the derivations of the judgement. So we will pick an arbitrary b and σ such that $\sigma(x)$ is even and we will let D be an arbitrary derivation that proves the judgement:

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

By induction on the structure of D:

- Base Case: the last rule of D is of the form

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

  By inversion we know this is the only rule that matches. This means that $\sigma = \sigma'$ and since $\sigma(x)$ is even the result holds.

- Case: the last rule of D is of the form

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

  By inversion of the evaluation rules we know that the derivation D' must be of the form:

Peer Review ID: 305779015 — enter this when you fill out your peer evaluation via gradescope

$$\dfrac{\dfrac{\dfrac{}{\langle 2, \sigma\rangle \Downarrow 2} \quad \dfrac{}{\langle x, \sigma\rangle \Downarrow \sigma(x)}}{\langle x+2, \sigma\rangle \Downarrow \sigma(x)+2}}{\langle x \coloneqq x+2, \sigma\rangle \Downarrow \sigma[x \coloneqq \sigma(x)+2]} \quad D'' :: \langle \texttt{while}\ b\ \texttt{do}\ x \coloneqq x+2, \sigma[x \coloneqq \sigma(x)+2]\rangle \Downarrow \sigma'}{\langle x \coloneqq x+2; \texttt{while}\ b\ \texttt{do}\ x \coloneqq x+2, \sigma\rangle \Downarrow \sigma'}$$

By our earlier assumption we know that $\sigma(x)$ is even, thus $\sigma(x) + 2$ is also even. Which in turn makes $\sigma[x \coloneqq \sigma(x)+2](x)$ even. Therefore by our inductive hypothesis the subderivation D'' matches the necessary conditions and $\sigma'(x)$ is even.

As the result holds for all structures of the derivation D the result holds. □

Exercise 2F-4. Language Features, Large-Step

$$\begin{array}{lcl} T & ::= & \sigma \mid \sigma\ \texttt{exc}\ n \\ c & ::= & \ldots \mid \texttt{throw}\ e \mid \texttt{try}\ c_1\ \texttt{catch}\ x\ c_2 \mid \texttt{after}\ c_1\ \texttt{finally}\ c_2 \end{array}$$

E-Throw
$$\dfrac{\langle e, \sigma\rangle \Downarrow n}{\langle \texttt{throw}\ e, \sigma\rangle \Downarrow \sigma\ \texttt{exc}\ n}$$

E-TryCatch1
$$\dfrac{\langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \texttt{try}\ c_1\ \texttt{catch}\ x\ c_2, \sigma\rangle \Downarrow \sigma'}$$

E-TryCatch2
$$\dfrac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \texttt{exc}\ n \qquad \langle c_2, \sigma'[x \coloneqq n]\rangle \Downarrow t}{\langle \texttt{try}\ c_1\ \texttt{catch}\ x\ c_2, \sigma\rangle \Downarrow t}$$

E-AfterFinally1
$$\dfrac{\langle c_1, \sigma\rangle \Downarrow \sigma' \qquad \langle c_2, \sigma'\rangle \Downarrow t}{\langle \texttt{after}\ c_1\ \texttt{finally}\ c_2, \sigma\rangle \Downarrow t}$$

E-AfterFinally2
$$\dfrac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \texttt{exc}\ n \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''}{\langle \texttt{after}\ c_1\ \texttt{finally}\ c_2, \sigma\rangle \Downarrow \sigma''\ \texttt{exc}\ n}$$

E-AfterFinally3
$$\dfrac{\langle c_1, \sigma\rangle \Downarrow \sigma'\ \texttt{exc}\ n \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''\ \texttt{exc}\ n'}{\langle \texttt{after}\ c_1\ \texttt{finally}\ c_2, \sigma\rangle \Downarrow \sigma''\ \texttt{exc}\ n'}$$

Exercise 2F-4. Language Features, Analysis

I believe that ”IMP with exceptions” is more naturally expressed using large-step semantics. In general, large-step semantics are easier to reason about because they focus on the final result of an expression’s evaluation rather than its intermediate reducible steps. This advantage is particularly evident when handling exceptions. In a small-step semantics, we would need to define reduction rules for every scenario in which an ”exceptional state” is propagated. For constructs like the `after`/`finally` command, this would require introducing intermediate reduction steps to account for the possibility that either c_1 or c_2 might throw an exception. In contrast, large-step semantics allow us to define these cases as separate, independent rules, avoiding the complexity of tracking exception propagation through intermediate states.

Peer Review ID: 305779015 — enter this when you fill out your peer evaluation via gradescope