CS 4110
Homework 2

Ameya Acharya (apa52) and Quinn Beightol (qeb2)

Exercise 1. Prove the following theorem using the large-step semantics:

Theorem. *If $\sigma(i)$ is even and $\langle \sigma, \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle \Downarrow \sigma'$ then $\sigma'(i)$ is also even.*

We prove this by induction on $\langle \sigma, \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle \Downarrow \sigma'$.
Let σ be some arbitrary store. We know that $\sigma(i)$ is even (given); let $\sigma(i) = n$.

Case 1: b is false.

$$\frac{\langle \sigma, b\rangle \Downarrow false}{\langle \sigma, \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle \Downarrow \sigma} \text{ WHILE-F}$$

Since $\langle \sigma, \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle \Downarrow \sigma'$, $\sigma' = \sigma$. Therefore, $\sigma'(i) = \sigma(i)$, so $\sigma'(i)$ is even. ✓

Case 2: b is true.

$$\frac{\langle \sigma, b\rangle \Downarrow true \quad \dfrac{\dfrac{n = \sigma(i)}{\langle \sigma, i + 2\rangle \Downarrow n + 2} \text{ VAR}}{\langle \sigma, i := n + 2\rangle \Downarrow \sigma'} \text{ ASSGN} \quad \langle \sigma', \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle \Downarrow \sigma''}{\langle \sigma, \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle \Downarrow \sigma''} \text{ WHILE-T}$$

By the inductive hypothesis applied to $\langle \sigma', \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle \Downarrow \sigma''$:

If b is false, then we may apply Case 1, and conclude that $\sigma' = \sigma''$. By the ASSGN rule used above, we know that $\sigma' = \sigma[i \mapsto n + 2]$. Since n is even (given), we know that $n + 2$ is even. ✓

If b is true, then we may recursively apply the above argument. Because we know that $\langle \sigma', \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle \Downarrow \sigma''$, we know that evaluation of $\langle \sigma', \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle$ terminates. Therefore, we know that there exists some intermediate σ_n such that

$$\langle \sigma_n, b\rangle \Downarrow false.$$

When we evaluate $\langle \sigma_n, \textbf{while}\ b\ \textbf{do}\ i := i + 2\rangle$, we will apply Case 1. From the induction hypothesis, we know that before we arrived at σ_n, $i := i + 2$ occurred some m times. Therefore, $\sigma_n = \sigma[i \mapsto n + 2m]$, which is even. ✓
This concludes the case, and the proof. ■

Exercise 2. Recall that IMP commands are equivalent if they always evaluate to the same result:

$$c_1 \sim c_2 \;\triangleq\; \forall \sigma, \sigma' \in \mathbf{Store}.\ \langle \sigma, c_1 \rangle \Downarrow \sigma' \iff \langle \sigma, c_2 \rangle \Downarrow \sigma'.$$

For each of the following pair of IMP commands, either use the large-step operational semantics to prove they are equivalent or give a concrete counter-example showing that they are not equivalent. You may assume that the language has been extended with operators such as $x! = 0$.

(a) $x := a;\ y := a$ and $y := a;\ x := a$,
where a is an arbitrary arithmetic expression

We show that these commands are not equivalent with the following counter-example.

Consider a store σ with $x = 3, y = 4$. Let our expression a be $x + y$.

Evaluating $x := a;\ y := a$:

$\langle \sigma, x := x + y; y := x + y \rangle$
$\rightarrow \langle \sigma, x := x + y; y := x + y \rangle$
$\rightarrow \langle \sigma, x := 3 + y; y := x + y \rangle$
$\rightarrow \langle \sigma, x := 3 + 4; y := x + y \rangle$
$\rightarrow \langle \sigma, x := 7; y := x + y \rangle$
$\rightarrow \langle \sigma', y := x + y \rangle, \text{where } \sigma' = \sigma[x \mapsto 7]$
$\rightarrow \langle \sigma', y := 7 + y \rangle$
$\rightarrow \langle \sigma', y := 7 + 4 \rangle$
$\rightarrow \langle \sigma', y := 11 \rangle$
$\rightarrow \sigma'', \text{where } \sigma'' = \sigma[x \mapsto 7, y \mapsto 11]$

Evaluating $y := a;\ x := a$:

$\langle \sigma, y := x + y; x := x + y \rangle$
$\rightarrow \langle \sigma, y := 3 + y; x := x + y \rangle$
$\rightarrow \langle \sigma, y := 3 + 4; x := x + y \rangle$
$\rightarrow \langle \sigma, y := 7; x := x + y \rangle$
$\rightarrow \langle \sigma', x := x + y \rangle, \text{where } \sigma' = \sigma[y \mapsto 7]$
$\rightarrow \langle \sigma', x := 3 + y \rangle$
$\rightarrow \langle \sigma', x := 3 + 7 \rangle$
$\rightarrow \langle \sigma', x := 10 \rangle$
$\rightarrow \sigma'', \text{where } \sigma'' = \sigma[x \mapsto 10, y \mapsto 7]$

We see that these two do *not* produce the same results. Therefore, the above IMP commands are not equivalent.

(b) **while** b **do** c and **if** b **then** (**while** b **do** c); c **else skip**,
where b is an arbitrary boolean expression and c an arbitrary command.

We show that these commands are not equivalent with the following counter-example.

Consider a store σ with $i = 2$. Let b be $i = 2$ and c be $i := i + 2$.

Evaluating **while** $i = 2$ **do** $i := i + 2$:

$\langle \sigma, \textbf{while } i = 2 \textbf{ do } i := i + 2 \rangle$
$\rightarrow \langle \sigma, \textbf{while } 2 = 2 \textbf{ do } i := i + 2 \rangle$
$\rightarrow \langle \sigma, i := i + 2; \textbf{while } i = 2 \textbf{ do } i := i + 2 \rangle$
$\rightarrow \langle \sigma, i := 2 + 2; \textbf{while } i = 2 \textbf{ do } i := i + 2 \rangle$
$\rightarrow \langle \sigma, i := 4; \textbf{while } i = 2 \textbf{ do } i := i + 2 \rangle$
$\rightarrow \langle \sigma', \textbf{while } i = 2 \textbf{ do } i := i + 2 \rangle, \text{where } \sigma' = \sigma[i \mapsto 4]$
$\rightarrow \langle \sigma', \textbf{while } 4 = 2 \textbf{ do } i := i + 2 \rangle$
$\rightarrow \sigma', \text{where } \sigma' = \sigma[i \mapsto 4]$

Evaluating **if** $i = 2$ **then** (**while** $i = 2$ **do** $i := i + 2$); $i := i + 2$ **else skip**:

$\langle \sigma, \textbf{if } i = 2 \textbf{ then } (\textbf{while } i = 2 \textbf{ do } i := i + 2); i := i + 2 \textbf{ else skip} \rangle$
$\rightarrow \langle \sigma, \textbf{if } 2 = 2 \textbf{ then } (\textbf{while } i = 2 \textbf{ do } i := i + 2); i := i + 2 \textbf{ else skip} \rangle$
$\rightarrow \langle \sigma, (\textbf{while } i = 2 \textbf{ do } i := i + 2); i := i + 2 \rangle$
$\rightarrow \langle \sigma, (i := i + 2; \textbf{while } i = 2 \textbf{ do } i := i + 2); i := i + 2 \rangle$
$\rightarrow \langle \sigma, (i := 2 + 2; \textbf{while } i = 2 \textbf{ do } i := i + 2); i := i + 2 \rangle$
$\rightarrow \langle \sigma, (i := 4; \textbf{while } i = 2 \textbf{ do } i := i + 2); i := i + 2 \rangle$
$\rightarrow \langle \sigma', (\textbf{while } i = 2 \textbf{ do } i := i + 2); i := i + 2 \rangle, \text{where } \sigma' = \sigma[i \mapsto 4]$
$\rightarrow \langle \sigma', (\textbf{while } 4 = 2 \textbf{ do } i := i + 2); i := i + 2 \rangle$
$\rightarrow \langle \sigma', i := i + 2 \rangle$
$\rightarrow \langle \sigma', i := 4 + 2 \rangle$
$\rightarrow \langle \sigma', i := 6 \rangle$
$\rightarrow \sigma'', \text{where } \sigma' = \sigma[i \mapsto 6]$

We see that these two do *not* produce the same results. Therefore, the above IMP commands are not equivalent.

(c) **while** x != 0 **do** $x := 0$ and $x := 0 * x$

TODO: do this exercise

Exercise 3. Let $\langle \sigma, c \rangle \rightarrow \langle \sigma', c' \rangle$ be the small-step operational semantics relation for IMP. Consider the following definition of the multi-step relation:

$$\frac{}{\langle \sigma, c \rangle \rightarrow^* \langle \sigma, c \rangle}\ \text{R1} \qquad\qquad \frac{\langle \sigma, c \rangle \rightarrow \langle \sigma', c' \rangle \qquad \langle \sigma', c' \rangle \rightarrow^* \langle \sigma'', c'' \rangle}{\langle \sigma, c \rangle \rightarrow^* \langle \sigma'', c'' \rangle}\ \text{R2}$$

Prove the following theorem, which states that $\rightarrow^*$ is transitive.

Theorem. *If* $\langle\sigma, c\rangle \rightarrow^* \langle\sigma', c'\rangle$ *and* $\langle\sigma', c'\rangle \rightarrow^* \langle\sigma'', c''\rangle$ *then* $\langle\sigma, c\rangle \rightarrow^* \langle\sigma'', c''\rangle$.

TODO: type this up.

Exercise 4. In this exercise, you will extend the IMP language with exceptions. These exceptions are intended to behave like the analogous constructs found in languages such as Java. We will proceed in several steps.

First, we fix a set of exceptions, which will ranged over by metavariables e, and we extend the syntax of the language with new commands for throwing and handling exceptions:

$$\begin{array}{rl} c ::= & \textbf{skip} \\ \mid & x := a \\ \mid & c_1; c_2 \\ \mid & \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \\ \mid & \textbf{while } b \textbf{ do } c \\ \mid & \textbf{throw } e \\ \mid & \textbf{try } c_1 \textbf{ with } e \textbf{ do } c_2 \end{array}$$

Intuitively, evaluating a command either yields a modified store or a pair comprising a modified store and an (uncaught) exception. We let metavariables r range over such results:

$$r ::= \sigma \mid (\sigma, e)$$

Second, we change the type of the large-step evaluation relation so it yields a result instead of a store: $\langle\sigma, c\rangle \Downarrow r$.

Third, we will extend the large-step semantics rules so they handle **throw** and **try** commands. This is your task in this exercise.

Informally, **throw** e should return exception e, and **try** c_1 **with** e **do** c_2 should execute c_1 and return the result it produces, unless the result contains an exception e, in which case it should discard e executes the handler c_2. You will also need to modify many other rules so they have the right type and also propagate exceptions.

TODO: type this up.

Debriefing

(a) How many hours did you spend on this assignment?
(b) Would you rate it as easy, moderate, or difficult?
(c) How deeply do you feel you understand the material it covers (0%100%)?
(d) If you have any other comments, we would like to hear them! Please write them here or send email to `jnfoster@cs.cornell.edu`.