Instructions: You may complete this assignment with another CS 6110 student. You must form a group with your partner on CMS before you begin working together on the assignment. With the exception of the course staff and your CMS partner, you should not give or receive assistance on this assignment. In addition, please limit use of resources to the lecture notes for this course. If you have any questions about what is allowed and what is not allowed, please contact the course staff.

1. Logo.

Logo is an old programming language for education where you give instructions to a turtle carrying a pen. Here is the syntax for a simple variant of Logo:

$$\begin{aligned} n &\in \mathbb{Z} \\ p &::= \mathsf{forward}\ n \mid \mathsf{right}\ n \mid \mathsf{repeat}\ n\ p \mid p_1\ ;\ p_2 \end{aligned}$$

The forward command draws a line by moving the turtle forward by a given distance, and right rotates the turtle by a given number of degrees. Running repeat n p repeats the subprogram p a given number of times. There is also sequencing, as in IMP.

Let's define a function $\mathsf{flip}(p)$ that takes a program and produces another program that draws the same picture but in "mirror image." It works by negating the distance and angle numbers in the drawing commands:

$$\begin{aligned} \mathsf{flip}(\mathsf{forward}\ n) &\triangleq \mathsf{forward}\ -n \\ \mathsf{flip}(\mathsf{right}\ n) &\triangleq \mathsf{right}\ -n \\ \mathsf{flip}(\mathsf{repeat}\ n\ p) &\triangleq \mathsf{repeat}\ n\ \mathsf{flip}(p) \\ \mathsf{flip}(p_1\ ;\ p_2) &\triangleq \mathsf{flip}(p_1)\ ;\ \mathsf{flip}(p_2) \end{aligned}$$

(Read $-n$ as the negation of the integer n.)

Prove that flip is *involutive*, which means that flipping a program twice produces the original program. Or, in notation: prove that $\mathsf{flip}(\mathsf{flip}(p)) = p$.

Hint: Write your proof using structural induction over the definition of p. Clearly label all uses of the induction hypothesis.

2. Semantics for for loops.

In this exercise we will extend IMP with for loops. The syntax for for loops is as follows,

$$c ::= \cdots \mid \mathsf{for}\ a\ \mathsf{do}\ c$$

where a is an arithmetic expression and c is a command. Informally, a for loop executes as follows. Upon entering the loop, the expression a is evaluated in the current state, yielding an integer n. If $n \leq 0$, the loop body is not executed at all. Otherwise, if $n > 0$, then the loop body c is executed exactly n times. No command in the loop body, such as assigning to a variable, should change the number of times the loop is executed. In addition, executing the loop should not change the value of any variables that do not appear in the body c.

(a) Write small-step operational semantics rules for this new command.

(b) Write large-step operational semantics rules for this new command.

Karma problem (zero points, just for practice): Consider a variant of IMP with for loops but without while loops. Prove that every program terminates.

3. IMP is deterministic.

Give a rigorous proof of the following fact, which states that the large-step evaluation relation for IMP, including the for loops added in Problem 2, is deterministic:

Theorem. *If* $\langle c, \sigma\rangle \Downarrow \sigma_1$ *and* $\langle c, \sigma\rangle \Downarrow \sigma_2$, *then* $\sigma_1 = \sigma_2$.

You may use the following facts in your proof without proving them:

- If $\langle a, \sigma\rangle \Downarrow_a n_1$ and $\langle a, \sigma\rangle \Downarrow_a n_2$, then $n_1 = n_2$. (This says that arithmetic expressions are deterministic.)
- If $\langle b, \sigma\rangle \Downarrow_b t_1$ and $\langle b, \sigma\rangle \Downarrow_b t_2$, then $t_1 = t_2$. (Similarly, Boolean expressions are deterministic.)

Hint: Use structural induction over the derivation of $\langle c, \sigma\rangle \Downarrow \sigma_1$. The property you'll prove by induction is $P(\langle c, \sigma\rangle \Downarrow \sigma_1) = (\langle c, \sigma\rangle \Downarrow \sigma_2 \;\Rightarrow\; \sigma_1 = \sigma_2)$. Your inductive proof will need one case per big-step inference rule for commands, including the new ones you wrote for for loops. (You can find the full set of rules in the notes for Lecture 8.) In each case, you'll also need to reason about the derivation of the second assumption, $\langle c, \sigma\rangle \Downarrow \sigma_2$, but you will not need a second induction.

4. Implementing IMP.

The archive `imp.zip` contains a partial implementation of the IMP imperative language. We have provided all the boring infrastructure (lexer, parser, and read-eval-print loop) which should compile and run right out of the box. We have also supplied some sample IMP programs.

Your job is to implement the interpreter for IMP in `eval.ml` using the big-step operational semantics we discussed in class.

The language includes a few extra features we did not formalize in class, such as some extra arithmetic operations, `for` loops as you formalized above, and a `print` command and an `input` expression (which you can implement using OCaml's corresponding standard functions). See `ast.ml` for all the syntax forms you'll need to support.

Once you have implemented the interpreter, you should be able to run `fact.imp` and `fib.imp`.

```
IMP interactive interpreter
>> help
Available commands are:
load <file>, list, run, help, quit
>> load examples/fact.imp
>> run
? 5
120
>> list
n := input;
f := 1;
while (n > 1) {
  f := f * n;
  n := n - 1;
}
print f;
>> quit
bye
```

Please submit the file `eval.ml` to CMS.

Debriefing.

(a) How many hours did you spend on this assignment?

(b) Would you rate it as easy, moderate, or difficult?

(c) Did everyone in your group participate?

(d) How deeply do you feel you understand the material it covers (0%–100%)?

(e) If you have any other comments, we would like to hear them! Please write them here or send email to `asampson@cs.cornell.edu`.