Exercise 1: Bookkeeping. Indicate in a sentence or two how much time you spent on this homework, how difficult you found it subjectively, and what you found to be the hardest part. Tell me something about yourself that I do not already know. Any non-empty answer will receive full credit.

Also, if your opinions have changed since the last assignment, indicate one thing you like about the class so far and one thing you would change about it.

Exercise 2: Induction. Prove by induction the following statement about the denotational semantics of IMP.

For any BExp $b$ and any initial state $\sigma$ such that $\sigma(x)$ is even, if

$$\Sem{\while b \do x := x + 2}{\sigma} = \sigma',$$

then $\sigma'(x)$ is even.

Note that we are assuming that $\Sem{\while b \do x := x + 2}{\sigma} \neq \bot$ in the above. Unlike in the previous assignment, this time you should use denotational semantics for the proof.

If you wish, you may consider the simplified language as in class in which case the problem can be rephrased as follows:

For any BExp $b$ and any initial state $n$ such that $n$ is even, if

$$\Sem{\while b \do x := x + 2}{n} = n',$$

then $n'$ is even.

Note that we are assuming that $\Sem{\while b \do x := x + 2}{n} \neq \bot$ in the above.

*Hint:* your proof should proceed by mathematical induction.
Exercise 3: Regular expressions are commonly used as abstractions for string matching. Here is an abstract syntax for regular expressions:

\( r ::= \text{"x"} \) singleton – matches the character \( x \)
\( \text{empty} \) skip – matches the empty string
\( r_1 \ r_2 \) concatenation – matches \( r_1 \) followed by \( r_2 \)
\( r_1 \ | \ r_2 \) or – matches \( r_1 \) or \( r_2 \)
\( r^* \) Kleene star – matches 0 or more occurrences of \( r \)
\( . \) matches any single character
\( [\text{"x"} - \text{"y"}] \) matches any character between \( x \) and \( y \) inclusive
\( r^+ \) matches 1 or more occurrences of \( r \)
\( r? \) matches 0 or 1 occurrence of \( r \)

We will call the first five cases the \textit{primary} forms of regular expressions. The last four cases can be defined in terms of the first five. We also give an abstract grammar for strings (modeled as lists of characters):

\( s ::= \text{nil} \) empty string
\( \text{"x"} :: s \) string with first character \( x \) and other characters \( s \)

We write \textit{"bye"} as shorthand for \textit{"b" :: \text{"y"} :: \text{"e"} :: nil}. This exercise requires you to give big-step operational semantics rules of inference related to regular expressions matching strings. We introduce the following judgment:

\[ \vdash r \text{ matches } s \text{ leaving } s' \]

The interpretation of the judgment is that the regular expression \( r \) matches some prefix of the string \( s \), leaving the suffix \( s' \) unmatched. If \( s' = \text{nil} \), then \( r \) matched \( s \) exactly. For example,

\[ \vdash \text{"h"("e"+)} \text{ matches } \text{"hello"} \text{ leaving } \text{"llo"} \]

Note that this operational semantics may be considered \textit{non-deterministic} because we expect to be able to derive all three of the following:

\[ \vdash (\text{"h" | "e"})^* \text{ matches } \text{"hello"} \text{ leaving } \text{"ello"} \]
\[ \vdash (\text{"h" | "e"})^* \text{ matches } \text{"hello"} \text{ leaving } \text{"hello"} \]
\[ \vdash (\text{"h" | "e"})^* \text{ matches } \text{"hello"} \text{ leaving } \text{"llo"} \]

Here are two rules of inference:

\[ s = \text{"x"} :: s' \]
\[ \vdash \text{"x"} \text{ matches } s \text{ leaving } s' \]
\[ \vdash \text{empty} \text{ matches } s \text{ leaving } s \]

Give big-step operational semantics rules of inference for the other three primal regular expressions.
Exercise 4: We will use denotational semantics to model the fact that a regular expression can match a string leaving many possible suffixes. Let $S$ be the set of all strings, let $\mathcal{P}(S)$ be the powerset of $S$, and let $\text{RE}$ range over regular expressions. We introduce a semantic function:

$$\mathcal{R}[\cdot] : \text{RE} \to (S \to \mathcal{P}(S))$$

The interpretation is that $\mathcal{R}[r]$ is a function that takes in a string-to-be-matched and returns a set of suffixes. We might intuitively define $\mathcal{R}$ as follows:

$$\mathcal{R}[r](s) = \{ s' \mid \vdash r \text{ matches } s \text{ leaving } s' \}$$

In general, however, one should not define the denotational semantics in terms of the operational semantics. Here are two correct semantic functions:

$$\mathcal{R}["x"](s) = \{ s' \mid s = \text{"x"} :: s' \}$$
$$\mathcal{R}[\text{empty}](s) = \{ s \}$$

Give the denotational semantics functions for the other three primal regular expressions. Your semantics functions may not reference the operational semantics.

Exercise 5: We want to update our operational semantics for regular expressions to capture multiple suffixes. We want our new operational semantics to be deterministic—it should give the same answer as the denotational semantics above. We introduce a new judgment as follows:

$$\vdash r \text{ matches } s \text{ leaving } S$$

And use rules of inference like the following:

$$\vdash \text{"x" matches } s \text{ leaving } \{ s' \mid s = \text{"x"} :: s' \} \quad \vdash \text{empty matches } s \text{ leaving } \{ s \}$$
$$\vdash r_1 \text{ matches } s \text{ leaving } S \quad \vdash r_2 \text{ matches } s \text{ leaving } S'$$
$$\vdash r_1 \mid r_2 \text{ matches } s \text{ leaving } S \cup S'$$

You must do one of the following:

- either give operational semantics rules of inference for $r*$ and $r_1 \ r_2$. Your operational semantics rules may not reference the denotational semantics. You may not place a derivation inside a set constructor, as in: $\{ x \mid \exists y. \vdash r \text{ matches } x \text{ leaving } y \}$. Each inference rule must have a finite and fixed set of hypotheses.

- or argue in one or two sentences that it cannot be done correctly in the given framework. Back up your argument by presenting two attempted but “wrong” rules of inference and show that each one is either unsound or incomplete with respect to our intuitive notion of regular expression matching.
Part of doing research in any area is getting stuck. When you get stuck, you must be able to recognize whether “you are just missing something” or “the problem is actually impossible”.

**Exercise 6:** Download the Homework 5 code pack from the course web page. The `README.txt` describes the code pack, like in Homework 3. Write an interpreter for regular expressions based on the denotational semantics. In particular, you must write a function

\[
\text{matches} : \text{RE} \rightarrow S \rightarrow \mathcal{P}(S)
\]

in `hw5.ml`. Your interpreter must handle all of the regular expression forms, not just the primal ones. The `Makefile` includes a “make test” target that you should use (at least) to test your work. Modify the `example.re` file to include your best test case. Rename `hw5.ml` to `your_identidkey-hw5.ml` and rename `example.re` to `your_identidkey-example.re` for submission. Do not modify any other files.

Note that in the concrete syntax for `RE`, we use the " symbol to delimit strings and the ’ symbol to delimit characters.