Homework 2
CSE6049 Program Analysis, Spring 2021
Woosuk Lee

due: 4/19(Mon), email-to-TA (bbumbuul@yahoo.com)

Please send a ZIP file titled "HW2_[Your Studnet ID].zip" to TA via email, and the zipped file should contain

- OCaml source files `hw1.ml`, `hw2.ml`, `hw3.ml`, and `hw5.ml` for Exercises 1, 2, 3, and 5 respectively,
- A PDF document file for Exercises 4 and 6.

Exercise 1 Binary numerals can be represented by lists of 0 and 1:

```
type digit = ZERO | ONE
type bin = digit list
```

For example, the binary representations of 11 and 30 are

`[ONE; ZERO; ONE; ONE]`

and

`[ONE; ONE; ONE; ONE; ZERO]`,

respectively. Write a function

`bmul:  bin -> bin -> bin`

that computes the binary product. For example,

`bmul[ONE; ZERO; ONE; ONE][ONE; ONE; ONE; ONE; ZERO]`

evaluates to `[ONE;ZERO;ONE;ZERO;ZERO;ONE;ZERO;ONE;ZERO]`.

Exercise 2 Consider the formulas of propositional logic:

$$
\begin{array}{rcll}
F & \rightarrow & true & \\
& | & false & \\
& | & P & \text{(variables)} \\
& | & \neg F & \text{(negation ``not'')} \\
& | & F_1 \wedge F_2 & \text{(conjunction ``and'')} \\
& | & F_1 \vee F_2 & \text{(disjunction ``or'')} \\
& | & F_1 \implies F_2 & \text{(implication)}
\end{array}
$$

The following algebraic data type characterizes propositional logic.

```
type formula = True
| False
| Var of string
| Neg of formula
| And of formula * formula
| Or of formula * formula
| Imply of formula * formula
```

We say a formula F is *satisfiable* iff there exists a variable assignment that makes the formula true. For example, the formula $P \wedge \neg Q$ is satisfiable because it evaluates to true when P is true and Q is false. The formula $P \wedge \neg P$ is not satisfiable since it always evaluates to false.

Write a function

```
sat :  formual -> bool
```

that determines the satisfiability of a given formula. For example,

```
sat (And (Var "P", Neg (Var "Q")))
```

returns true.

Exercise 3 Consider the following expressions:

```
type exp = X
         | INT of int
         | ADD of exp * exp
         | SUB of exp * exp
         | MUL of exp * exp
         | DIV of exp * exp
         | SIGMA of exp * exp * exp
```

Implement a calculator for the expressions:

```
calculator :  exp -> int
```

For instance,

$$\sum_{x=1}^{10}(x \times x - 1)$$

is represented by

```
SIGMA(INT 1, INT 10, SUB(MUL(X, X), INT 1))
```

and evaluating it should give 375.

Exercise 4 Consider the following simple drawing language used in the lecture:

$$
\begin{array}{lcll}
p & \rightarrow & \texttt{init}([l_1, u_1], [l_2, u_2]) & \text{(initialization with a state } (x, y) \text{ such that } l_1 \leq x \leq u_1, l_2 \leq y \leq u_2) \\
 & | & \texttt{translation}(u, v) & \text{(translation by vector } (u, v)) \\
 & | & \texttt{rotation}(\theta) & \text{(rotation defined by center } (0, 0) \text{ and angle } \theta) \\
 & | & p \;;\; p & \text{(sequence of operations)} \\
 & | & \{p\}\texttt{or}\{p\} & \text{(non-deterministic choice of branch)} \\
 & | & \texttt{iter}\{p\} & \text{(iteration (the number of iterations is non-deterministic))}
\end{array}
$$

Define a *big-step operational semantics* for the language. A state is a real-value coordinate ($s \in State = \mathbb{R} \times \mathbb{R}$). You inductively define a set of sentences of form $s \vdash p \Rightarrow s'$ (given a state s, executing a program p will result in a new state s').

The followings are ingredients that may be useful for the definition.

- The new coordinates (x', y') of a point (x, y) after rotation at an angle θ are

$$x' = x \cos\theta - y \sin\theta$$
$$y' = x \sin\theta + y \cos\theta$$

- The uniform probability distribution is denoted $\mathcal{U}(0, 1)$, and a random variable $x \sim \mathcal{U}(0, 1)$ following the uniform distribution holds a real value in $[0, 1]$. Using the notations, we can define the inference rules for non-deterministic choices as follows:

$$\frac{s \vdash p_1 \Rightarrow s_1 \qquad s \vdash p_2 \Rightarrow s_2 \quad r \sim \mathcal{U}(0,1) \quad r > 0.5}{s \vdash \{p_1\}\texttt{or}\{p_2\} \Rightarrow s_1}$$

$$\frac{s \vdash p_1 \Rightarrow s_1 \qquad s \vdash p_2 \Rightarrow s_2 \quad r \sim \mathcal{U}(0,1) \quad r \leq 0.5}{s \vdash \{p_1\}\texttt{or}\{p_2\} \Rightarrow s_2}$$

Solutions:

$$\frac{}{s \vdash \texttt{init}(R) \Rightarrow s'} \; s' \in R$$

$$\frac{s = (x, y) \quad s' = (x + u, y + v)}{s \vdash \texttt{translation}(u, v) \Rightarrow s'}$$

$$\frac{s = (x, y) \quad s' = (x', y') \quad x' = x \cos\theta - y \sin\theta \quad y' = x \sin\theta + y \cos\theta}{s \vdash \texttt{rotation}(\theta) \Rightarrow s'}$$

$$\frac{s \vdash p_1 \Rightarrow s' \qquad s' \vdash p_2 \Rightarrow s''}{s \vdash p_1 \; ; \; p_2 \Rightarrow s''}$$

$$\frac{s \vdash p_1 \Rightarrow s_1 \qquad s \vdash p_2 \Rightarrow s_2 \quad r \sim \mathcal{U}(0,1) \quad r > 0.5}{s \vdash \{p_1\}\texttt{or}\{p_2\} \Rightarrow s_1}$$

$$\frac{s \vdash p_1 \Rightarrow s_1 \qquad s \vdash p_2 \Rightarrow s_2 \quad r \sim \mathcal{U}(0,1) \quad r \leq 0.5}{s \vdash \{p_1\}\texttt{or}\{p_2\} \Rightarrow s_2}$$

$$\frac{r \sim \mathcal{U}(0,1) \quad r > 0.5}{s \vdash \texttt{iter}\{p\} \Rightarrow s}$$

$$\frac{s \vdash p \Rightarrow s' \quad s' \vdash \texttt{iter}\{p\} \Rightarrow s'' \quad r \sim \mathcal{U}(0,1) \quad r \leq 0.5}{s \vdash \texttt{iter}\{p\} \Rightarrow s''}$$

Exercise 5 The following data type characterizes the drawing language.

```
type pgm = INIT of (float * float) * (float * float)
         | TRANSLATE of (float * float)
         | ROTATION of float
         | SEQ of pgm * pgm
         | OR of pgm * pgm
         | ITER of pgm
```

Write a function

```
eval :  pgm -> float * float
```

that returns a final state (i.e., coordinate) after executing a given program `pgm`. In OCaml, you can generate a random floating number in [0,1] by

```
Random.float 1.0
```

Exercise 6 Define a *collecting semantics* of the drawing language in a compositional style. A collecting semantics concerns all possible outcomes of program executions (whereas operational semantics concerns a single outcome of a single program execution).

In other words, we are interested in defining a function that takes a set of initial coordinates and returns a set of resulting output coordinates. Define a function

$$[\![p]\!] : 2^{State} \to 2^{State}$$

that returns a set of output states for a given set of input states. For example, the first two cases are defined as follows:

$$\begin{array}{rcl} [\![\texttt{init}([l_1, u_1], [l_2, u_2])]\!](S) & = & \{(x, y) \mid l_1 \leq x \leq u_1, l_2 \leq y \leq u_2\} \\ [\![\texttt{translation}(u, v)]\!](S) & = & \{(x + u, y + v) \mid (x, y) \in S\} \end{array}$$

For the other remaining cases, complete the definition of $[\![p]\!]$.

Solutions:

$$\begin{array}{rcl} [\![\texttt{init}([l_1, u_1], [l_2, u_2])]\!](S) & = & \{(x, y) \mid l_1 \leq x \leq u_1, l_2 \leq y \leq u_2\} \\ [\![\texttt{translation}(u, v)]\!](S) & = & \{(x + u, y + v) \mid (x, y) \in S\} \\ [\![\texttt{rotation}(\theta)]\!](S) & = & \{(x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta) \mid (x, y) \in S\} \\ [\![p_1 \ ; \ p_2]\!](S) & = & [\![p_2]\!]([\![p_1]\!](S)) \\ [\![\{p\}\texttt{or}\{p\}]\!](S) & = & [\![p_1]\!](S) \cup [\![p_2]\!](S) \\ [\![\texttt{iter}\{p\}]\!](S) & = & \mathrm{lfp} \lambda X.\ X \cup [\![p]\!](X) \end{array}$$