Exercise 09

Probabilistic Programming

Reliable and Interpretable Artificial Intelligence
ETH Zurich

```
def main(){ // didItRain
    cloudy := flip(0.5);
    rain := 0; sprinkler := 0;

    if (cloudy){
        rain = flip(0.8);
        sprinkler = flip(0.1);
    }else{
        rain = flip(0.2);
        sprinkler = flip(0.5);
    }

    wetGrass := rain || sprinkler;

    observe(wetGrass);
    return rain==1;
}
```

Listing 1: Did it rain?

Consider the program in Listing 1. It shows a probabilistic model that determines (i) if it is cloudy, (ii) if it is raining, (iii) if a sprinkler is running and (iv) if the grass is wet. Then, it observes that the grass is wet, and returns whether it rains (i.e., whether `rain==1`).

Problem 1 (Understanding Listing 1)**.** Visualize Listing 1 as a *Bayesian Network*. Concretely, draw a directed graph consisting of vertices and edges, where vertices represent variables in Listing 1 and edges between `x` and `y` mean that variable `y` depends on `x`. You may omit tables indicating the (conditional) probability distribution of each variable.

Problem 2 (Running PSI)**.** Run PSI to determine the probability that it rains, given that the grass is wet.

- Install PSI according to the instructions on `https://github.com/eth-sri/psi`. On most platforms, it suffices to clone PSI using

  ```
  git clone git@github.com:eth-sri/psi.git
  ```

  and to build PSI using

  ```
  cd psi; ./dependencies.sh && ./build.sh
  ```

- Run PSI to determine the required probability. You can run `psi rain.psi --expectation`, which returns the expectation of the program in Listing 1 and hence the probability that `rain` is true (why?).

Problem 3 (Writing a PSI program)**.** Use PSI to compute the probability that the average of 40 dice throws is above 4.

Likewise, compute the probability that the average of 20 dice throws is above 4, given that at least 10 rolls yield the value 6.

Hints: Consult the Quick language guide on `https://github.com/eth-sri/psi` for a quick introduction to PSI. For purely discrete programs, you can use the flag `--dp` to speed up PSI, as in `psi dice.psi --expectation --dp`.

Problem 4 (Encoding a Neural Network)**.** Recall the Neural Network N with one hidden layer from Exercise 6:

$$N(x_1, x_2) = \text{ReLU}(\text{ReLU}(-x_1 + x_2 + 2) + \text{ReLU}(x_1 - 2x_2))$$

Work out the output distribution of N, assuming its input is distributed according to two Laplacian distributions, each with location 0 and scale 1.

Hint: Use the flag `--plot` to visualize the resulting distribution.

Problem 5 (Computing π)**.** Write a PSI program whose expectation is π. Hint: Sample from a square.

Because it is hard to compute this expectation symbolically, PSI does not fully simplify the expectation. However, we could use sampling techniques to approximate π using the determined probabilistic program.