WebPPL Examples

Exercise 1 (Simulating a six-sided die by a fair coin) 0%

Example: Simulating a fair six-sided die by fair coin flips using technique called "rejection sampling"

Solution:

```
var die_RS = function ()
{
  var a0 = flip(0.5)
  var a1 = flip(0.5)
  var a2 = flip(0.5)
  var res = 1 + a0 + 2*a1 + 4*a2 // 1 <= res <= 8
  return res <= 6
    ? res      // done, output res
    : die_RS() // picked 7 or 8, try again
}

var out_RS = Infer(die_RS) // exact inference

viz.auto(out_RS)
```

Test of the program really models a 6-sided die. Compare the behaviour of the above program to the following program:

Solution:

```
var die_RS = function ()
{
  var a0 = flip(0.5)
  var a1 = flip(0.5)
  var a2 = flip(0.5)
  var res = 1 + a0 + 2*a1 + 4*a2 // 1 <= res <= 8
  condition(res <= 6)
}

var out_RS = Infer(die_RS)

viz.auto(out_RS)
```

Exercise 2 (Geometric Distribution) **0%**

The geometric distribution gives the probability that the first occurrence of success requires k independent trials, each with success probability p.

If the probability of success on each trial is p, then the probability that the kth trial (out of finite trials) is the first success is $\mathsf{Pr}(X = k) = (1-p)^{k-1} \cdot p$ for all k.

Write a recursive program to generate a geometric distribution with parameter $0 < p < 1$.

Solution:

```
// Geometric distribution; recursively

var geometric = function() {
  return flip(p) ? 0 : geometric() + 1;
}

var p = 0.3

var dist = Infer({method: 'enumerate', maxExecutions: 10}, geometric);

viz.auto(dist);
```

Adapt your program such that only odd numbers are computed.

Solution:

```
var geometric = function(p)
{
  return flip(p) ? 1 + geometric(p) : 1
}

var geo_wrapper = function()
{
  var res = geometric(0.58)
  condition ((res % 2) == 1)
  return res
}

var geo_dist = Infer({method: 'enumerate', maxExecutions: 50}, geo_wrapper)
viz.auto(geo_dist)
```

Only odd numbers at most 9? Only numbers between 50 and 60?

Exercise 3 (Piranha Puzzle) **0%**

One fish is contained within the confines of an opaque fishbowl. The fish is equally likely to be a piranha or a goldfish. A sushi lover throws a piranha into the fish bowl alongside the other fish. Then, immediately, before either fish can devour the other, one of the fish is blindly removed from the fishbowl. The fish that has been removed from thew bowl turns out to be a piranha. What is the probability that the fish originally in the bowl itself was a piranha?

Solution:

```
var piranha_puzzle = function () {
   var f1 = flip(0.5) ? 'pir' : 'gf'
   var f2 = 'pir'
   var s = flip(0.5) ? f1 : f2
   condition (s == 'pir')
   return(f1 == 'pir')
}

var out = Infer({method: 'enumerate', maxExecutions: 3}, piranha_puzzle)
viz.auto(out)
```

Exercise 4 (Randomised Quicksort) **0%**

The randomized Quicksort is a variant of the traditional Quicksort where in each turn the pivot element is chosen at random among all the elements in the array. Its pseudocode looks like

$$
\begin{array}{l}
rQS(A) \triangleq \\
\quad \mathsf{if}\ (|A| \leq 1)\ \mathsf{then\ return}\ (A) \\
\quad pivot := \mathsf{Uniform}(0...|A|-1) \\
\quad A_< := \{a \in A \mid a < A[pivot]\} \\
\quad A_> := \{a \in A \mid a > A[pivot]\} \\
\quad \mathsf{return}\ \big(rQS(A_<) +\!\!+ A[pivot] +\!\!+ rQS(A_>)\big)
\end{array}
$$

(a) Implement in WebPPL function rQS. Assume it takes an array of (unrepeated) natural numbers.

Solution:

```
  var rQS = function (A)
{
  var len = A.length
  if (len <= 1) { return A }
  var pivot = sample(RandomInteger({n:len}))
  var lt_pivot = filter(function(x) {x < A[pivot]}, A)
  var gt_pivot = filter(function(x) {x > A[pivot]}, A)
  return rQS(lt_pivot).concat([A[pivot]].concat(rQS(gt_pivot)))
}

rQS([2,5,1,7,3])
```

(b) Write the WebPPL code to execute rQS on inputs $[2, 5, 1, 7, 3]$ and $[10, 5, 4, 1]$. Verify that you obtain the expected answer.

Solution:

```
rQS([2,5,1,7,3])
rQS([10,5,4,1])
```

(c) Extend your webPPL program such that you can keep track of the pivots being selected. Determine how many steps are needed to sort the input array.

Exercise 5 (Random Walk) **0%**

(a) Provide a function `RW` encoding a random walk on a one-dimensional line that starts at some position (e.g., 5, or 50, or 124), and in each turn moves to the left with probability p or to the right with probability $1-p$. The random walk stops when position 0 is reached. Write your program in such that it is parametric in the starting position and the probability p.

(b) Run the program with values $p = 0.1, 0.2, \ldots, 0.9, 1$. What can you empirically deduce from your experiment?

Solution:

1.
```
var p = 0.3
var RW = function (x)
{
  return (x == 0) ? 0 : (flip(p) ? RW(x+1) : RW(x-1))
}

var y = randomInteger(1000) // select a random starting position
var RW_from_y = function() {return RW(y)}

RW_from_y()
```

2. The random walk terminates only when $p \in [0,\ 0.5]$.

Exercise 6 (Variants of Random Walks) **0%**

(a) Adapt your WebPPL program for the symmetric random walk such that you keep track of the number of steps until reaching position zero.

Solution:

```
var RW = function(pos, steps) {
  if (pos == 0) return steps;
  return flip(0.48) ? RW(pos+1, steps+1) : RW(pos-1, steps+1)
}

var dist = Infer({method:'rejection', samples:160},function(){RW(10,0)})

print(expectation(dist)) // expected number of steps until reaching pos 0
viz.auto(dist) // print the distribution of the number of steps to reach 0
```

(b) Escaping spline: program a random walk that at position k goes to zero with probability $\frac{1}{1+k}$, and otherwise moves one position to the right.

Solution:

```
var Escaping_Spline = function (x)
{
  flip(1/(1+x)) ? 0 : Escaping_Spline(x+1)
}

var ES_from_5 = function() {return Escaping_Spline(5)}
ES_from_5()
```

(c) Fair-in-the-limit one-dimensional random walk moving to origin. This random walk moves at position k one position to the right with probability $1 - \frac{1}{1+k}$, and otherwise one position to the left. Draw the random walk for position 0 upto position 6. Write a webPPL program.

Solution:

```
var OneD_FiL_RW = function (x)
{
  return (x == 0) ? 0 : (flip(1-(1/(1+x))) ? OneD_FiL_RW(x+1) : OneD_FiL_RW(x-1))
}

var FiL_RW_from_5 = function() {return OneD_FiL_RW(5)}
FiL_RW_from_5()
```

Exercise 7 (Turning a Biased into a Fair Coin) **0%**

Using a coin with some arbitrary bias $0 < p < 1$, the program below generates a sample according to a fair coin flip. The loop terminates when the biased coin was flipped twice and showed different outcomes. Obviously the program terminates with probability one as on each iteration of the loop there is a constant positive chance to terminate. The value of x is taken as the outcome.

Solution:

```
var p = 0.5;

var TwoBiasedCoins = function () {
  var x = flip(p);
  var y = flip(p);
  if (x == y) { TwoBiasedCoins() } else { return x }
}

var dist = Infer({method: 'enumerate', maxExecutions: 50}, TwoBiasedCoins)
print(expectation(dist))
viz.auto(dist)
```

Exercise 8 (Birthday Paradoxon) **0%**

(a) What is the probability that in a room filled with 23 people, at least one pair of persons has the same birthday? Write a WebPPL program to determine this probability.

Solution:

```
*/
var model = function() {

  var N = 23;

  var birthday = function(m) {
    return 1 + sample(RandomInteger({n:365}));
  }

  var f = function(i, j) {
    if(i > N) {
      return false;
    }
    if(j > N) {
      return f(i+1, i+2);
    }
    if(birthday(i) == birthday(j)) {
      return true;
    }
    return f(i, j+1);
  }
  return f(1,2);
}

// infer probabilities
var posteriors = Infer({method: 'rejection', samples: 2500}, model);

print(posteriors);
viz.auto(posteriors);
```

(b) How many people in the same room are at least required such that the probability of one pair of persons having the same birthday exceeds 0.9?

Solution: Empirically, we conjecture $N = 41$ or $N = 42$.
Of course we can also directly compute the correct solution to check this:
The probability that N people have different birthdays is given by

$$\texttt{Pr}\,(N \text{ different birthdays}) \;=\; \frac{Perm(N,365)}{365^N} \;=\; \frac{365}{365}\cdot\ldots\cdot\frac{365-N+1}{365} \;=\; \frac{365!}{(365-n)!}\cdot\frac{1}{365^n}\;.$$

The remaining probability is then what we are looking for:

$$\texttt{Pr}\,(\text{at least two equal birthdays}) \;=\; 1-\texttt{Pr}\,(N \text{ different birthdays})\;.$$

We thus obtain the following exact probabilities:

$$\begin{aligned} N = 23: &\quad 0.507\\ N = 40: &\quad 0.891\\ N = 41: &\quad 0.903\\ N = 42: &\quad 0.914 \end{aligned}$$

Hence, $N = 41$ are sufficent such that the probability of two people having the same birthday is above 0.9.

Exercise 9 (Reasoning about Reasoning) **0%**

Two agents play a game in which each agent needs to name a number between 0 and 9. They win if their numbers add up to 13. The first player knows this, and she knows that the second player gets to see the number the first player chooses. However, the second player mistakenly thinks that the two win if their numbers add up to any number greater than 8. The first player knows this as well.

Provide a WebPPL program for player one to determine which number she should choose.

For this exercise, you may use all features of standard programming languages, such as assignments, recursion, etc. Moreover, to sample a number between 0 and 9 you can use the expression "sample(RandomInteger(n:10))". However, you are not allowed to use explicit conditioning constructs, e.g. condition(...).

Solution:

```
var model = function() {

  var player1 = function() {
    var a = sample(RandomInteger({n:10}));
    var b = player2(a);
    (a+b == 13) ? a : player1();
  }

  var player2 = function(x) {
    var c = sample(RandomInteger({n:10}));
    (x+c > 8) ? c : player2(x);
  }

  return player1();
}

viz.auto(Infer({method: 'rejection', samples: 10000}, model));
```

Exercise 10 (Recursion) **0%**

(a) Write a recursive WebPPL program that with probability one half immediately terminates, and otherwise invokes itself once. Determine the termination probability of this program. Try first with exact inference, and then use a breadth-first strategy.

(b) Repeat the above question, but now if the process invokes itself twice in a row with probability a half. Is the termination probability changing?

(c) Repeat the above question, but now if the process invokes itself three times in a row with probability a half. Is the termination probability changing?

Solution:

```
var one_call = function ()
{
  return flip(0.5) ? 0 : 1 + one_call()
}

var one_call_wrapper = function () {one_call()}
var out_one_call = Infer({method: 'enumerate', maxExecutions: 30, strategy: 'breadthFirs
viz.auto(out_one_call)
one_call()

var two_calls = function ()
{
  return flip(0.5) ? 0 : 1 + two_calls() + two_calls()
}

var two_calls_wrapper = function () {two_calls()}
var out_two_calls = Infer({method: 'enumerate', maxExecutions: 30, strategy: 'breadthFirs
viz.auto(out_two_calls)
two_calls()

var three_calls = function ()
{
  return flip(0.5) ? 0 : 1 + three_calls() + three_calls() + three_calls()
}

var three_calls_wrapper = function () {three_calls()}
var out_three_calls = Infer({method: 'enumerate', maxExecutions: 30, strategy: 'breadthF
viz.auto(out_three_calls)
three_calls()
```

Exercise 11 (Mutual Recursion: Virus Spreading) **0%**

Consider a simple epidemiological model for the outbreak of an infectious disease in a large population where the number of susceptible individuals can be assumed to be infinite. Our example model distinguishes young and elderly persons. Each affected individual infects a uniformly distributed number of others, with varying rates (expected values) according to the age groups. A young person infects one young persons and one elderly person. An elderly person infects one young and three elderly persons. The fatality rate for infected elderly and young persons is 1% and 0%, respectively. Initially, we assume there is a single infected young person.

Solution:

```
var infectYoung = function() {
  globalStore.steps += 1;
  if (globalStore.steps >= globalStore.maxSteps) {
    return
  }
  var y = randomInteger(1)
  repeat(y, infectYoung);
  var e = randomInteger(2);
  repeat(e, infectElder);
}

var infectElder = function() {
  globalStore.steps += 1;
  globalStore.dead += 1;
  if (globalStore.steps >= globalStore.maxSteps) {
    return
  }
  var y = randomInteger(1)
  repeat(y, infectYoung);
  var e = randomInteger(3);
  repeat(e, infectElder);
}

var model = function() {
  globalStore.maxSteps = 1000; // set to Infinity to wait until infection dies out
  globalStore.steps = 0;
  globalStore.dead = 0;
  infectYoung();
  return globalStore.dead
};

var dist = Infer({method: 'rejection', samples: 1000}, model)
print(expectation(dist))
```

```
viz.auto(dist)
```

Exercise 12 (Duelling Cowboys) **0%**

Model the following situation: There are two cowboys, A and B, who are fighting a classical duel. They take turns, shooting at each other until one of them is hit. If A (resp. B) shoots then he hits B (resp. A) with probability a (resp. b). We assume that either cowboy A or B is allowed to start; the choice of who will start is resolved randomly. Note that it is a distinctive feature that we do not have to specify exact probabilities and instead allow arbitrary parameters such as a and b for the hitting probabilities of cowboy A and B respectively.

Solution:

```
// Input: cowboy A that hits opponent with probability a
//        cowboy B that hits opponent with probability b
// Required output: probability that A (rest. B) survives
// Assumptions: 1. cowboys shoot in alternating fashion
//              2. a hit is fatal

var Duel = function (a, b, next_turn) {
return (next_turn == 'A')
      ? (flip(a) ? 'A' : Duel(a,b,'B'))
      : (flip(b) ? 'B' : Duel(a,b,'A'))
}

var a = 1/2
var b = 1/2

var random_start = function () {
return flip(1/2) ? Duel(a,b,'A') : Duel(a,b,'B')
} // select who will start

var out_random_start = Infer({method: 'enumerate', maxExecutions: 30}, random_start)
viz.auto(out_random_start)
```