Exercise 3

Discrete Probabilistic Programs and Bayonet

Program Analysis for System Security and Reliability 2018
ETH Zurich

March 19, 2018

Scheduler We will use uniform scheduling in all problems:

```
RunSw:=0, FwdQ:=1;

def scheduler(){
  actions := []: (ℝ × ℝ)[]; // list of actions
  for i in [0..k) { // k: number of nodes
    if (Q_in@i).size() > 0 {
      actions ~= [(RunSw, i)]; } // run node
    if (Q_out@i).size() > 0 {
      actions ~= [(FwdQ, i)]; // forward packet
    }
  }
  n := actions.length; // pick action u.a.r.:
  return actions[uniformInt(0,n-1)];
}
```

Simple Bayonet Program Consider the following Bayonet program:

```
topology{
  nodes{ H0, H1, S0, S1 }
  links{
    (H0,pt1) <-> (S0,pt1),
    (H0,pt2) <-> (S1,pt1),
    (H1,pt1) <-> (S0,pt2),
    (H1,pt2) <-> (S1,pt2)
  }
}
queue_capacity 2; // (effectively infinite)

packet_fields{ id }

def h0(pkt, port) state pkt_count(0){
  if pkt_count < 2 {
```

```
16      new;
17      pkt_count = pkt_count + 1;
18      pkt.id = pkt_count;
19      if flip(1/2){ fwd(1); }
20      else { fwd(2); }
21    } else { drop; }
22  }
23
24  def s0(pkt, port){ fwd(2); }
25
26  def s1(pkt, port){ fwd(2); }
27
28  def h1(pkt, port) state pkt_count(0), swapped(0){
29    pkt_count = pkt_count + 1;
30    if pkt_count == 2{
31      if pkt.id == 1{
32        swapped = 1; // arrived in order (2, 1)
33      }
34    }
35    drop;
36  }
37
38  query probability(swapped@H1);
```

Problem 1: State Graph For this program, the program state is given by the state of the input and output queues as well as the local state variables of each switch. (Recall that each queue stores pairs of packets and input/output ports.) Start from a state where the input queue of host H0 contains a single packet with id 0 and all other queues are empty.

Draw (part of) a directed graph whose nodes are program states that can be reached with non-zero probability. Draw edges from each state to the states reachable by executing a single action of the scheduler. Label each edge with the probability that the program will make the transition represented by the edge given that execution has reached the state at the head of the edge. Explicitly draw at least 10 nodes.

Problem 2: Probabilistic Inference using Dynamic Programming A terminal state of the state graph is a node that has no outgoing edges. Given an acyclic state graph such as the one from problem 1, how can we compute the probability that the Bayonet program reaches a terminal state that satisfies some property (e.g. `swapped@H1`)? Can you do it in time linear in the size of the state graph? (Assume that we can evaluate the property in constant time and that we can perform any standard arithmetic operation in constant time.)

Problem 3: Encoding in PSI PSI is available at `psisolver.org`. (For Windows users: There are currently no supported native build scripts for Windows. You can follow the instructions in the README file using the Linux subsystem for Windows or build PSI manually.)

Invoke PSI as follows:
`psi --noboundscheck --dp --expectation program.psi`

1. Familiarize yourself with PSI syntax using the README file in the PSI repository.
2. Manually write a PSI encoding of the given Bayonet program and use it to answer the query, i.e., compute the probability that the two packets sent from host H0 arrive at host H1 in reversed order.
3. Add `observe` statements that select only executions where the two packets are forwarded along distinct paths and compute the probability that the packets are reordered. How does the probability change? Explain why. (Hint: You may need to add additional program state.)

Hint: You can use the following implementation for your packet queues:

```
dat Packet{
  id: R;
  def Packet(id: R){
    this.id = id;
  }
}
dat Queue{
  data: (Packet x R)[];
  def Queue(){
    data = ([]:(Packet x R)[]);
  }
  def pushFront(x: Packet x R){
    data=[x]~data;
  }
  def pushBack(x: Packet x R){
    if size() >= 2 { return; }
    data=data~[x];
  }
  def takeFront(){
    r:=front();
    popFront();
    return r;
  }
  def takeBack(){
    r:=data[size()-1];
    data=data[0..size()-1];
    return r;
  }
  def size(){
    return data.length;
  }
```

```
32   def front(){
33     return data[0];
34   }
35   def dupFront(){
36     pushFront(front());
37   }
38   def popFront(){
39     data=data[1..size()];
40   }
41 }
```

Problem 4: State Graph Size Consider a fully connected network on n nodes where input and output queues have capacity c. Assume that each node forwards all incoming packets to a random neighbor. Consider the state graph that arises by considering all states reachable from a state where $k \leq n$ nodes each have one packet in their input queues and all other queues are empty. (All packets are distinguishable by id.)

1. For the special case $c = 2$, $k = 1$, give a closed-form expression for the number of nodes of the state graph in terms of n.
2. For the special case $c = 2$, $k = 2$, give a closed-form expression for the number of nodes of the state graph in terms of n.
3. For the special case $c = 2$, $k = 3$, give a closed-form expression for the number of nodes of the state graph in terms of n.
4. For the special case $c = 2$, $k = 4$, give a closed-form expression for the number of nodes of the state graph in terms of n.
5. Optional challenge: Write a program that can compute the number of state graph nodes for arbitrary parameters $n, c, k \in \mathbb{N}, k \leq n$. Make the asymptotic running time as low as you can. (You can again assume that standard arithmetic operations take constant time.)