INGI1131 Practical Exercises
Lab 8: Message Passing

Until last lab session we saw how good declarative concurrency is, but we also saw the limitations of it. This week we will study message passing using ports, which introduces non-determinism and overcomes the limitations we had until last week.

1. Ports is the most basic abstraction for providing the programming concept *"message passing"*. But starting to work with them, we will do a very basic exercise to work with lists. Implement a procedure {`ReadList L`} that takes a list `L` as argument and browse all the elements of it.

2. Ports are a kind of communication channel. You can send messages to a port, and read the messages received by the port. Every port has a stream associated to it. Sending a message to a port results in adding the message to port's stream. To create a port we use the procedure {`NewPort S P`}, which creates port `P`, and it associates stream `S` to `P`. Remember that a stream is just an infinite list. To send messages to a port, we use the procedure {`Send P Msg`}, which sends `Msg` to `P`. The following example send messages `foo` and `bar` to port `P`.

   ```
   declare
   P S
   {NewPort S P}

   {Send P foo}
   {Send P bar}
   ```

   But we haven't done anything with the stream associated to the port yet. Try browsing it with {`Browse S`}. What do you observe?

3. Create a port and use `ReadList` to browse each message that it is sent to the port.

4. Create a procedure {`RandomSenderManiac N P`} that launches `N` threads. Each thread waits a random time between 1 and 3 seconds before sending the integer `I` to port `P`. The value of `I` is given by `RandomSenderManiac` incrementally from 1 to `N`.

5. Combine procedures `RandomSenderManiac` and `ReadList` to observe the order in which messages arrive to the port. Is it a deterministic behaviour? In other words, is there any observable nondeterminism? Or as the duchess of Wonderland would had explained it, is it observable what otherwise would had or might have been deterministic if the nondeterminism would had or might have not been observable?

6. In the last lab session we studied the implementations of procedures `WaitOr` and `WaitOrValue`. Now you have to implement `WaitTwo`. The idea is that a call to

{`WaitTwo X Y`} will return 1 if `X` is bound before `Y`, and 2 otherwise. Implement it using ports. Remember that `WaitTwo` is not deterministic. Do you know why? In other words, is there any observable non-determinism? (This time we will not include Duchess's version of the question. If you don't understand this comment, you did not do exercise 5 of Lab 07).

7. Implement a server that receive messages in the form `Msg♯Ack`, where `Ack` is an unbound variable. The messages arrive via a `Port`. To acknowledge the reception of the message, the server binds the variable `Ack` to `unit` after waiting for a random time between 500 and 1500 milliseconds. The delay is only a way to simulate latency in the communication between two machines.

8. Implement a function {`SafeSend P M T`} that takes a port `P`, a message `M`, and a time value `T` (in milliseconds) . The function sends a pair `M♯Ack` to the port `P`. If the receiver acknowledges receipt by binding `Ack` within `T` milliseconds, the function returns `true`. Otherwise, the function returns `false`.

9. Now that you have mastered the use of ports, the exercise of the miners-counter with multiple pipes will be much easier to implement. Figure 1 depicts the problem. There is a counter that keeps track of the occurrences of characters coming from several clients. The server should generate an output stream where each position of the stream corresponds to the total count so far. Each position of the output stream is a list of tuples `C♯N`, where `N` is the number of times that character `C` has been sent to the server.

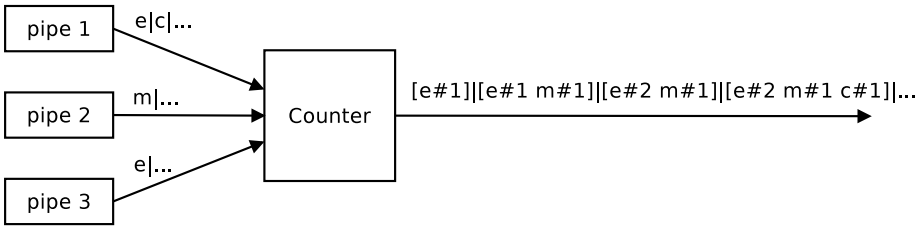

Figure 1: Counter (server) with more than one pipe (client)

   a The figure illustrates one possible output under the inputs given by the clients. However, this is not the only possible output. Why? What are the other possible outputs under the same input?

   b What is the ultimate answer to the great question of life, the universe and everything.

10. Implement {`StreamMerger S1 S2`} that takes two streams as input, and returns one stream as output. The output is a merge of streams `S1` and `S2`. As soon as a value arrives in any of this streams, the value is added to the output.

    a Write a first implementation **without** ports. Use only the function `WaitTwo` that you implemented in the first exercise of this lab session. Note that `StreamMerger` allows us to implement a server that can listen to two clients without a problem. How would you use it for `N` clients.

    b Once again, is `WaitTwo` deterministic? In other words, does it belongs to the declarative concurrent programming paradigm?

    c Implement StreamMerger using ports.