Concurrent Programming - Exercises 6

Value passing CCS

1. For each process P compute the LTS $[\![P]\!]$
   (a) $(a!21.0|a?x.b!(x*2).0)\backslash\{a\}$
   (b) $(a!42.0|a?x.Sender[x])\backslash\{a\}$

2. For the following program compute its LTS.

$$\begin{array}{rcl} Sender & := & put?x.send!x.Sending[x] \\ Sending[x] & := & receiveAck?.Sender + receiveNAck?.send!x.Sending[x] \\ Receiver & := & receive?x.get!x.sendAck!.Receiver + \\ & & gargled?.sendNAck!.Receiver \\ Medium & := & send?x.(receive!x.Medium + i.garbled!.Medium) \\ AckMedium & := & sendAck?.receiveAck!.AckMedium + \\ & & sendNAck?.receivedNAck!.AckMedium \\ DupMedium & := & Medium|AckMedium \\ Protocol & := & (Sender \mid Receiver \mid DupMedium)\backslash \\ & & \{send, receive, sendAck, receiveAck, \\ & & receiveNAck, sendNAck, garbled\} \end{array}$$

   Compute also $[\![(Protocol|put!1.put!2.put!3.put!4.0)\backslash\{put\}]\!]$. Restrict the values to $rangeR := 0..9$.

3. Given

$$\begin{array}{rcl} Fac[n,j] & := & when(j>0)i.Fac[n*j, j-1] \\ & & +when(j==0)println!n.0 \end{array}$$

   Compute $[\![Fac[1,5]]\!]$.

4. Consider the following vending machine that accepts coins of 1, 2 and 5 euros. If the price of a coffee has not been paid yet, more coins are required before a coffee is dispensed. Once enough money has been paid, no more coins are accepted, and a coffee is dispensed.If the last inserted coin caused the coffee to be overpaid, some change is given. Each coffee cost 5 euros.

$$\begin{array}{rcl} Machine[b] & := & when(b<5)coin?c.Machine[b+c] + when(b\geq 5)coffee!.ReturnMachine[b-1] \\ ReturnMachine[b] & := & when(b>0)change!.ReturnMachine[b-1] + Machine[0] \\ User & := & coin!2.coin!2.coin!2.coffee?.change?.0 \end{array}$$

   Construct $[\![(Machine[0]|User)\backslash\{coin, change, coffee\}]\!]$.

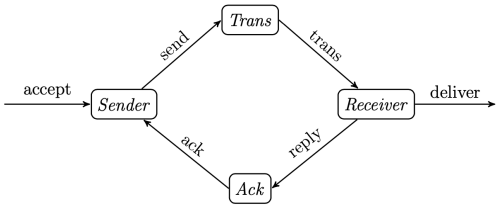
5. Given

$$\begin{aligned} Cell[rd, wr, x] &:= rd!x.Cell[rd, wr, x] + wr?y.Cell[rd, wr, y] \\ Cells &:= Cell[rdA, wrA, 0] | Cell[rdB, wrB, 0] \\ Serve &:= mult?.rdA?x : R.rdB?y : R.IterMult[0, x, y] \\ IterMult[z, x, y] &:= when(x > 0)i.IterMult[z + y, x - 1, y] \\ &\quad + when(x == 0)println!z.Serve \\ Use &:= wrA!7.wrB!5.mult!.0 \end{aligned}$$

   Compute $[\![(Cells|Serve|Use)\backslash\{rdA, wrA, rdB, wrB, mult\}]\!]$.

6. Consider the exercise 9 from Practical 5 with value passing CCS.

7. (Alternating bit protocol) Alternating-Bit Protocol (ABP) is a simple protocol for reliably transmitting data from a sender to a receiver over unreliable communication channels. This is accomplished by retransmitting lost messages if required.



   ABP works as follows. For the process to send a message:

   - Accepts a message to send
   - Sends it with the bit b and activates the timer
   - When the timer timeout the sender resends b
   - If if gets a confirmation b by the channel Ack, continues to accept a new messagen and the bit $1 - b$.
   - Ignores any confirmation in Ack of $1 - b$.

   For the receiver is similar:

   - Receives and delivers the message
   - Replies with bit b and activates a timer
   - When the timer timeout resends b
   - Na recepção de uma nova mensagem $1 - b$ entrega-a e prossegue com $1 - b$Receiving a message $1 - b$ delivers it and continues with $1 - b$.
   - Ignora mensagens etiquetadas com bIgnores any message in Trans with b

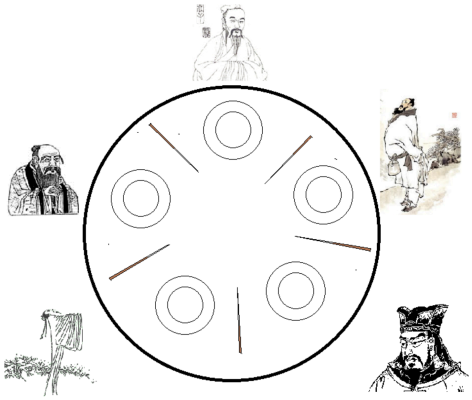
   The processes Sender and Receiver are obtained by parallel composition with the Timer ignoring the Timer actions. The processes Trans and Ack are nondeterministic and simulates the unreliable communication.

   (a) Implement ABP as a CCS process definition. You can abstract away from the content of the messages and focus only on the additional control bit.

(b) Suggest a specification of the protocol in CCS and check whether it is equivalent to your implementation of ABP.

(c) Check for deadlocks.

8. (Philosophers Dinner)There are five philosophers Chuang-Tzu, Wong-Chongyang, Sun-Tzu, Lie-Yukou and Lao-Tsu (clock-wise from the top) sitting around a round table with a bowl of rice in front of them and a chopstick placed between each pair of adjacent philosophers. Being Tao monks, they are silent, and the only activity they engage themselves in are thinking and eating, alternatively. Each philosopher needs two chopsticks to eat (duh!) and can only use the ones adjacent to them if they are free.



(a) Write a CCS statement to model the behaviour of all philosophers and chopsticks.

(b) See if there is any deadlock if the philosophers are left to their own devices.

(c) Dijkstra's "resource hierarchy solution" ranks each chopstick. A philosopher can only pick a chopstick of lower rank first and then the higher rank if available. Write CCS commands to implement this.