Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Concurrency

Date: Thursday 20th May 2010
Time: 14.00 – 16.00

Please answer any THREE questions from the FIVE questions provided

This is a CLOSED book examination

The use of electronic calculators is NOT permitted.
1. This question is about modelling a gate-controlled crossing of a road over a rail track, i.e. a level crossing.

At an appropriately safe distance from the level crossing, an approaching train will cause an “approach” signal to occur, which in turn triggers the level crossing gates to “close”. Once the train has “passed” the crossing, the gates are signalled to “open” again.

The FSP process CONTROL could be used to model this simple “control” protocol.

CONTROL = (approach -> close -> passed
-> open -> CONTROL).

A “gate” is then modelled using the FSP process GATE.

GATE = (close -> open -> GATE).

The level crossing could then be modelled as a parallel composition of the CONTROL process and two GATE processes.

||CROSSING = (CONTROL || north:GATE || south:GATE).

a) Explain why the above definition for CROSSING fails to satisfy the requirement that the gates close after the train signals it’s approaching the crossing. (2 marks)
b) Fix the definition of CROSSING so that it achieves the desired behaviour, explaining carefully the semantic effect of the syntactic changes you make. (4 marks)

c) The above simplistic control system can hardly be regarded as safe. Suppose, for example, a train has signalled that it is approaching but the gate then fails to close in some way. The train would then appear to be able to go through the crossing with the gates potentially open and hence potentially at the same time as road users. Modify the CONTROL process to take into account another signal that indicates the train is “near”, which happens after the “approach” signal but in time for the train to be able to stop before reaching the level crossing. The normal sequence of signals would then be:

approach close near passed open

However, if the “close” doesn’t occur for some reason before the “near” signal happens, the control system should issue an emergency “halt” signal to the train. Following the “halt”, the control system should await the “close” from the gate, then issue a “go” signal to the train, and then continue as before, i.e. “open” the gates once the train has “passed” the crossing. (5 marks)

d) The above FSP definition for a gate is also rather too simplistic. The following is a small improvement in which the “close” action is replaced by a sequenced pair “closing” and “closed”, the former representing a signal to start closing and latter indicating that the gate is fully closed.

GATE = (closing -> closed -> open -> GATE).

Modify the CONTROL process you gave in answer to part (c) to handle this pair of signals, but now subject to the following requirement. If a “closed” signal does not occur with some given number of clock ticks following a “closing” signal, then the train must be stopped via the emergency “halt” signal introduced above. Again, the control should then await for the “closed” signal, restart the train and continue as before. (9 marks)
2. a) Given an FSP process, say \( P \), what is meant by its alphabet, denoted by \( \alpha(P) \). (2 marks)

b) The process \( VM \), defined below, models a vending machine that can accept five pence and ten pence coins and serve newspapers, cans of drink and chocolate bars. Calculate the alphabet, \( \alpha(VM) \), of \( VM \).

\[
VM(\text{Max}=20) = ( \text{fiveP} \rightarrow VM[5] \\
| \text{tenP} \rightarrow VM[10] ) ,
\]

\[
VM[s:0..\text{Max}] = \\
( \text{when } (s >= 15) \text{ paper } \rightarrow VM[s-15] \\
| \text{when } (s >= 10) \text{ can } \rightarrow VM[s-10] \\
| \text{when } (s >= 5) \text{ bar } \rightarrow VM[s-5] \\
| \text{when } (s <= \text{(Max-5)}) \text{ fiveP } \rightarrow VM[s+5] \\
| \text{when } (s <= \text{(max-10)}) \text{ tenP } \rightarrow VM[s+10] ) .
\] (2 marks)

c) What is meant by a finite trace of an FSP process. Give two examples of finite traces that the vending machine process \( VM(10) \) defined above can make, and then give a sequence of actions over the alphabet of \( VM \) that the process \( VM(10) \) can NOT make, and justify why it can not do so. (3 marks)

d) Assume a user of the vending machine modelled as \( VM(15) \) inputs the coins as defined by the process \( COINS \).

\[
COINS = ( \text{fiveP} \rightarrow \text{tenP} \rightarrow \text{fiveP} \rightarrow \text{STOP} ) .
\]

Give two examples of possible servings that the vending machine could then make. How many different servings would be possible for this input? (4 marks)

e) Now construct the minimised labelled transition system corresponding to the following composition.

\[
|| SYS = ( VM(15) || COINS )@\{\text{paper, can, bar}\}.
\]

Carefully explain why this process composition represents all the possible servings that the vending machine \( VM \) can make for the given input defined by \( COINS \). (5 marks)

f) How could you use the LTSA tool to check that a given serving was, or wasn't, possible? In particular, how would you show that the vending machine couldn't deliver two papers for the input defined by the process \( COINS \). (4 marks)
3. a) Define the notion of strong bisimilarity between two labelled transition systems, and hence define the notion of strong bisimilarity for FSP processes.

   (4 marks)

b) Explain the difference between the trace equivalence of two FSP processes and bisimilarity of two FSP processes. Give an example that clearly demonstrates the difference.

   (4 marks)

c) Outline an algorithm for establishing whether two given FSP processes are bisimilar.

   (6 marks)

d) Apply the algorithm you gave in answer to part (c) to determine the bisimilarity, or otherwise, of the FSP processes $P$ and $Q$ defined below.

\[ P = (a \rightarrow c \rightarrow P | b \rightarrow PC), \]
\[ PC = (b \rightarrow c \rightarrow P | c \rightarrow a \rightarrow PC). \]
\[ Q = (a \rightarrow c \rightarrow QC | b \rightarrow QCC), \]
\[ QC = (a \rightarrow c \rightarrow Q | b \rightarrow QCC ), \]
\[ QCC = (b \rightarrow c \rightarrow QC | c \rightarrow a \rightarrow QCC). \]  

(6 marks)
4. a) Briefly explain the following concepts:
   i) Mutual exclusion
   ii) Synchronous and asynchronous message passing
   iii) Locks in Java
   iv) Synchronised methods in Java (8 marks - 2 marks per part)

b) Consider the following FSP definitions that model a shared counter in concurrency terms as a monitor, more literally as a shared variable with increment and decrement operations modelled by processes INC, DEC.

   const N = 4
   range V = 0..N

   set VarAlpha = {read[V], write[V], acq, rel}

   VAR = VAR[0],
   VAR[v:V] = (read[v] -> VAR[v] |
                   write[u:V] -> VAR[u]).

   LOCK = (acq -> rel -> LOCK).
   \|LOCKVAR = (LOCK \| VAR).

   INC = (inc -> TryToINC),
   TryToINC = (acq -> read[x:V] -> INC[x]),
   INC[v:V] = (when (v < N) write[v+1] -> rel -> INC |
               when (v == N) rel -> TryToINC)+VarAlpha.

   DEC = (dec -> TryToDEC),
   TryToDEC = (acq -> read[x:V] -> DEC[x]),
   DEC[v:V] = (when (v > 0) write[v-1] -> rel -> DEC |
               when (v == 0) rel -> TryToDEC)+VarAlpha.

   \|Compose = ( {a,b}::counter:LOCKVAR
                   \| a.counter:INC \| b.counter:DEC ).

   Briefly explain what was meant by “monitor” and then carefully explain how the FSP process LOCK is used to ensure exclusive access to the shared variable (modelled by the VAR process). Make sure that you explain the purpose of the alphabet extension “+VarAlpha” in the process definitions of INC and DEC. (6 marks)

c) Provide a Java class definition for the shared counter modelled in part (b) taking particular care to explain how the locking and synchronisation is achieved. (6 marks)
5. a) Explain the difference, in FSP terms, between a blocked process, a deadlocked system and a livelocked system? (3 marks)

b) Explain what is meant by a safety property. Is freedom from deadlock a safety property? (2 marks)

c) A safety property is defined in FSP as a process definition prefixed by the keyword `property` as in the example below.

```fsharp
property DINING_PROPERTY = ( sitdown -> getup -> DINING_PROPERTY ).
```

Explain the difference between `DINING_PROPERTY` and the process `TABLE` defined below, giving the labelled transition systems for both.

```fsharp
TABLE = ( sitdown -> getup -> TABLE ).
```

(4 marks)

d) Consider the following FSP model of a dining table, with up to `Max` diners, and a semaphore.

```fsharp
const Max = 4
range S = 0..Max

SEMAPHORE(N=0) = STATE[N],
STATE[s:S] =
    ( up -> STATE[s+1]
    | when (s>0) down -> STATE[s-1] ).

DINER = ( down -> sitdown -> eat
         | getup -> up -> DINER ).

TABLE = ( eat -> TABLE ).

||DINING = ( p[S]::TABLE || p[S]::DINER
            || p[S]::SEMAPHORE(1) ).
```

Explain how the `SEMAPHORE` process ensures that there can be only one diner present at the table to eat, i.e. having performed a `sitdown` action but not the corresponding `getup` action. (2 marks)

Modify the `DINING` composition to allow for at most 2 out of the possible `Max` diners to be present at the table at any time. Carefully justify your answer. (2 marks)

(Question 5 continues on the following page)
(Question 5 continues from the previous page)

e) Modify the `DINING_PROPERTY` given in part (c) to ensure that only one of the possible `Max` diners can be sat down at the table able to eat at any one time, and show how it would be used with the composition `DINING`.

(3 marks)

Rewrite the safety property given in answer to the above to check that at most 2 of the possible `Max` diners can be sat down.

(4 marks)