COMP31212

Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Concurrency and Process Algebra

Date: Friday 27th May 2016

Time: 09:45 - 11:45

Please answer any THREE Questions from the FIVE Questions provided

---

This is a CLOSED book examination

The use of electronic calculators is NOT permitted

[PTO]

1. Modelling and implementing concurrent systems.

   a) In the context of concurrent computational systems, explain briefly but clearly what is meant by the following:

      i. A process algebra,
      ii. Interference,
      iii. Mutual exclusion,
      iv. Threads in Java,
      v. Locks and synchronisation in Java.

      (5 marks)

   b) Explain clearly what is meant by a *monitor* as a device for structuring concurrent systems. What sort of mechanisms may a monitor provide to control access?
      (3 marks)

   c) A building has **two** doors which record people entering and leaving via a shared counter. The counter is incremented when a person enters the building and decremented when a person leaves the building, so that the counter records the number of people in the building.

      Model this system of three concurrent processes using the process algebra FSP. In your model, use an indexed process to model the counter, and process labelling to form the composite system. (4 marks)

   d) Now provide an implementation of the three interacting processes by writing an outline program using threads in Java. (5 marks)

   e) Explain carefully the relationship between your FSP model and the Java program implementing it. You should explain in detail how the interaction between processes corresponds to your use of threads, locks and synchronisation in Java.
      (3 marks)

2. On FSP rules and derivation trees.

a) Give the inference rules for the parallel operator || in FSP. Explain how synchronised and unsynchronised transitions of concurrent systems are described by these rules. (3 marks)

Consider the following FSP process description of a simple client-server system in which the server attempts to provide a `service` to the client upon `request`, but may fail to be able to do so, in which case the server responds with a `failreport`.

```
CLIENT = ( request -> WAIT ),
WAIT = ( reply -> continue -> CLIENT
          | failreport -> errorhandle -> CLIENT ).

SERVER = ( request -> TRY ),
TRY = ( service -> reply -> SERVER
          | failreport -> SERVER ).

||SYSTEM = ( CLIENT || SERVER ).
```

b) Using the transition rules for FSP, provide a detailed derivation of the `request` transition that the `SYSTEM` can make. Carefully explain the rules that you use at each step of the derivation. (6 marks)

c) Immediately after an execution of the `request` action, the resulting process may perform a `service` action. Show a derivation of this action, again carefully justifying the steps of the derivation with FSP rules.
(5 marks)

d) Define precisely how a labelled transition system (LTS) may be constructed from an FSP process definition using FSP rules. (3 marks)

e) Use this analysis to draw a labelled transition system that corresponds to the composite FSP process `SYSTEM` above. You should construct an LTS with the minimum number of states. (3 marks)

3. On the equivalence of FSP processes.

   a) Give the definition of *(not an algorithm for)* strong bisimilarity between two labelled transition systems, and hence define strong bisimulation for FSP processes. (3 marks)

   b) Explain clearly the relationship between the following notions of process equivalence: (i) strong bisimilarity, (ii) trace equivalence, (iii) having the same labelled transition system. (2 marks)

   c) Describe an algorithm for computing whether two FSP processes are strongly bisimilar or not. Your description should clearly explain all the steps of the algorithm. (5 marks)

   d) Consider the following FSP process definitions.

   i.
```
P1 = ( a -> R | b -> STOP ),
R  = ( a -> R | b -> STOP ).

Q1 = ( a -> Q1 | b -> STOP ).
```

   ii.
```
P2 = (a -> S),
S  =  (b -> STOP | c -> STOP).

Q2 = ( a -> b -> STOP | a -> c -> STOP ).
```

   iii.
```
P3 = ( a -> X),
X  = ( b -> X | b -> Y | c -> P3 ),
Y  = ( b -> Y | c -> P3 ).

Q3 = ( a -> U ),
U  = ( b -> V | c -> Q3 ),
V  = ( b -> V | c -> Q3 ).
```

   Use the algorithm you described in your answer to Part (c) above to determine whether or not (i) P1 is strongly bisimilar to Q1, (ii) P2 is strongly bisimilar to Q2, and (iii) P3 is strongly bisimilar to Q3. In each case, show your working by giving the results at each step of the algorithm.

   In cases where the two processes are not strongly bisimilar, explain how this is determined by the algorithm. In cases where the two processes are strongly bisimilar, use the algorithm to determine a labelled transition system (LTS) with a minimum number of states which is strongly bisimilar to both processes. (10 marks)

4. Properties of concurrent systems.

a) Explain briefly but clearly the following concepts applied to concurrent systems. You should use illustrative examples. (4 marks)

i) Deadlock

ii) Livelock

b) Explain what is meant by a *safety property* of a concurrent system. 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.

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

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

```
DINING_PROCESS = ( sitdown -> getup -> DINING_PROCESS ).
```

(3 marks)

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

```
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) ).
```

d) 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 a corresponding `getup` action. (2 marks)

e) Modify the `DINING` composition of processes to allow for at most 2 of the possible diners to be present at the table at any one time. Justify your answer. (2 marks)

f) Using labelled and indexed actions, modify the property `DINING_PROPERTY` defined above to test the property that only one of the possible diners can be sat down at the table to eat at any one time, and show how the composite process `DINING` defined above is extended to test the property. How do we determine whether the property holds by analysing the extended composite system? (4 marks)

g) Rewrite the safety property given in your answer to Part (f) above so that it checks that at most two of the possible diners can be sat down at any one time. Hint: One way is to introduce processes indexed by the actual diners at the table.
(3 marks)

5. Concurrency concepts.

   a) Explain what is meant by a *liveness property* for a concurrent system. Give an example of a real system together with a liveness property that is required to hold for the system. (3 marks)

   b) Explain the notion of *fairness* as it applies to concurrent systems. Describe the types of fairness that may be exhibited. (3 marks)

   c) Define the notion of a *strongly connected component* of a finite, directed graph. (3 marks)

   d) What is meant by a strongly connected component being *terminal*? (2 marks)

   e) Explain clearly the relationship between terminal strongly connected components of labelled transition systems (LTSs) and fairness of FSP processes. (3 marks)

   f) Consider the following FSP process. By identifying the strongly connected components of its LTS, explain whether the system is fair or not in terms of the two actions `heads` and `tails`. (6 marks)

```
TWOCOIN = ( pick -> COIN
                  | pick -> TRICK ),
TRICK = ( toss -> heads -> TRICK ),
COIN = ( toss -> heads -> COIN
           | toss -> tails -> COIN).
```


END OF EXAMINATION