UNIVERSITY OF OSLO

Faculty of Mathematics and Natural Sciences

Examination in:	IN9170 — Models of Concurrency
Day of examination:	12. December 2018
Examination hours:	14:30 – 18:30 (4 hours)

This problem set consists of 10 pages.

Appendices:	None
Permitted aids:	No written or printed material

Please make sure that your copy of the problem set is complete before you attempt to answer anything.

Some general advice and remarks:

- Before you start to solve the problems, take a look at the whole problem set to schedule your time.
- The number of points stated on each part indicates the weight of that part.
- You can make your own clarifications if needed. Please write down any such clarifications.
- Make brief and clear explanations!
- Make your program implementations as simple as possible!

Good luck!

(Continued on page 2.)

In this exam we will look at a synchronization problem in several language settings. We first provide a description of the problem.

Consider male and female users sharing a limited number of resources. For this problem, we imagine there are n showers reserved for the male users, and n showers reserved for the female users, and in addition n showers that may be used by both male and female users, but only one user in a shower at a time. Here n is a fixed constant. Thus there are in total $3n$ showers, so at most $3n$ persons can shower at the same time, and at most $2n$ male users may shower at the same time (when no shared shower is occupied by a female).

We assume that each person (male or female) is represented by a process that repeatedly tries to take a shower and then eat, work and sleep. The synchronization problem is to ensure that the users are given appropriate access to the showers as long as there is a free shower of the right kind.

The male processes should follow the following general outline:

```
begin    -- local variables
  while true do
    "enter";
    shower;
    "leave";
    eat;work;sleep
  od
end
```

where the parts of the code marked *enter* and *leave* involve synchronization, depending on the language setting and synchronization mechanisms. (The end of a while loop is here indicated by "od" and the end of a process body is indicated by "end".)

Problem 1 The "await language" (weight 30)

1a Shower control in the "await language" (weight 10)

Consider the setting of shared variable with critical regions and await statements. Let the shared variables m, f, and u denote the number of occupied showers for males only, females only, and those for any users, respectively.

Program the male user process in this setting. You may refer to the shared variables for synchronization.
Make sure that the critical regions are as small as possible.

Solution: Male Process:

(Continued on page 3.)

```
begin Bool b   -- a local variable
 while true do
 < await m+u<2*n;
   b:= m<n;
   if b then m:=m+1 else u:=u+1 fi >;
 shower;
 if b then <m:=m-1> else <u:=u-1> fi;
 eat; work; sleep
 od
end
```

Here b is a local variable which is set true when a male-only shower is used. This is used after showering to count down the appropriate counter.

1b Critical Regions (weight 5)

Explain briefly why the critical regions cannot be made smaller.

Solution: For the entry part: the critical region is needed here to ensure that the shower to be taken is not already occupied, or becoming occupied. For the exit part: mutual exclusion to the shared variables is ensured.

1c A Global Invariant (weight 5)

Consider the global invariant

$$m \leq n \wedge f \leq n \wedge u \leq n$$

Is this invariant expressing any useful property with respect to the synchronization problem? In particular, which of the following requirements are ensured by the invariant:

1. at most n persons may use the male-only showers at a given time
2. at most n persons may use the female-only showers at a given time
3. at most n persons may use the shared showers at a given time
4. no male user may use a female-only shower
5. no female user may use a male-only shower.

Solution: Yes it says that only the available showers are used, i.e., properties 1, 2, and 3. But it does not say that males are not using female-only showers and vice versa, i.e., properties 4 and 5.

(Continued on page 4.)

1d Verification Conditions (weight 5)

Explain briefly how the global invariant can be verified, by stating a number of requirements formulated as Hoare triples that must be verified (verification conditions).

Explain also briefly if the interference problem appear in the verification of these verification conditions, i.e., can they be verified without using the reasoning rules for interference or is interference part of the reasoning.

Solution: Let I denote the invariant. It must be proved that each atomic statement or critical region changing one of the shared variables maintains the invariant. This amounts to:

$$\{I\} < await\ (m{+}u < 2{*}n); b := m < n; if\ b\ then\ m := m{+}1\ else\ u := u{+}1\ fi > \{I\}$$

$$\{I\} < m := m - 1 > \{I\}$$

$$\{I\} < u := u - 1 > \{I\}$$

All these are about critical regions and therefore there are no interference problems. We may use ordinary sequential style reasoning.

1e Verification of the Invariant (weight 5)

Verify the global invariant by means of Hoare logic, by verifying the required Hoare triples from the previous task.

Solution: We may use ordinary sequential style reasoning since we only need to consider the critical regions.

First verification condition (using here forward reasoning):

```
$\{I\}$  await (m+u<2*n);     $\{I \wedge m+u < 2*n\}$
  b:= m<n;        $\{I \wedge m+u < 2*n \wedge b = (m < n)\}$
  if b then
  $\{m < n \wedge f <= n \wedge u <= n\}$  m:=m+1 $\{m <= n \wedge f <= n \wedge u <= n\}$

  else $\{m = n \wedge f <= n \wedge u < n\}$  u:=u+1 $\{m = n \wedge f <= n \wedge u <= n\}$

  fi  $\{I\}$
```

At the end of the then-branch we need to prove:

$$m <= n \wedge f <= n \wedge u <= n \Rightarrow I$$

which is trivial since these are identical. At the end of the else-branch we need to prove:

$$m = n \wedge f <= n \wedge u <= n \Rightarrow I$$

(Continued on page 5.)

which is trivial.

Second and third verification conditions:

$$\{I\}\, m := m - 1\, \{I\}$$

$$\{I\}\, u := u - 1\, \{I\}$$

These are similar. Let us do the last one. It reduces to I implies $I[u := u-1]$. This reduces to

$$m \leq n \wedge f \leq n \wedge u \leq n \Rightarrow m \leq n \wedge f \leq n \wedge u - 1 \leq n$$

which is trivial since $u \leq n \Rightarrow u \leq n + 1$. Similar with $m := m - 1$.

Problem 2 Semaphores (weight 20)

2a Shower Control with Semaphores (weight 15)

Consider here the shower synchronization problem in the setting of shared variables and semaphores. Program the male shower process in this setting. Use m, f, and u as the shared variables. Define initial values for each semaphore variable.

Solution: In addition to the shared variables m, f, and u and a semaphore for *mutex*, initialized to 1, we use two more semaphores: *male* and *female*, each initialized to $2 * n$. At any time (at least outside critical regions) they represent the free showers available for males and females.

$$male = 2 * n - m - u$$

$$female = 2 * n - f - u$$

Male process:

```
while true do
  P(male);
  < b:=(m<n); if b then m:=m+1 else  P(female); u:=u+1 fi >;
  shower;
  if b then <m:=m-1> else <u:=u-1>; V(female) fi; V(male);
  eat; work; sleep
od
```

Here "<" denotes P(mutex) and ">" denotes V(mutex).

(Continued on page 6.)

2b P and V operations (weight 5)

Explain that there is enough V operations, so that the processes waiting in a P will be able to continue.

Also, explain the purpose of each semaphore variable. And if possible explain the value of each semaphore by an expression.

Solution: Expressions for the semaphores are given above. V(female) is made exactly when the female expression is increased (by 1), and V(male) is made exactly when the male expression is increased (by 1).

Problem 3 Monitors (weight 15)

3a Shower Control with Monitors (weight 10)

Consider here the setting of monitors with signal-and-continue discipline. Program the shower controller as a monitor with procedures *maleshower* and *malefinished*, to be called by male processes, and procedures *femaleshower* and *femalefinished*, to be called by female processes. Let the procedures *maleshower* and *femaleshower* return a Boolean result indicating what kind of shower is given to the caller; and let *malefinished* and *femalefinished* have a Boolean parameter, indicating what kind of shower was used. More specifically, *maleshower* should return a Boolean indicating true if the caller is given a male shower and false if he is given a shared one, and *femaleshower* returns true if the caller is given a female shower and false if she is given a shared one. (You may use either an out-parameter or a return value.)

We assume that male processes call procedure *maleshower* before showering and then *malefinished* after showering (with the right parameters). Program the shower controller as a monitor based on the given assumptions.

Solution:

Monitor solution:

```
monitor Shower_Controller
begin
cond malefree         -- m+u<2*n
cond femalefree       -- f+u<2*n
Nat m, f, u           -- as before

Bool procedure maleshower()  begin Bool b;
 while m+u=2*n do wait(malefree) od;
 b := m<n;
 if b then m:=m+1 else u:=u+1 fi;
 return b
```

(Continued on page 7.)

```
 end

Bool procedure femaleshower()  begin Bool b;
 while f+u=2*n do wait(femalefree) od;
 b := f<n;
 if b then f:=f+1 else u:=u+1 fi;
 return b
 end

procedure malefinished(Bool b)  begin
 if b then m:=m-1 else u:=u-1; signal(femalefree) fi;
 signal(malefree);
 end

procedure femalefinished(Bool b)  begin
 if b then f:=f-1 else u:=u-1; signal(malefree) fi;
 signal(femalefree);
 end
end of monitor
```

3b Condition Variables (weight 5)

For each condition variable c, identify the corresponding Boolean wait condition, i.e., the condition that is waited for by a $wait(c)$ statement.

Solution: Given above.

Problem 4 Asynchronous Agents (weight 35)

Consider here the setting of asynchronous agents with send and await (receive) statements. We allow guarded await statements of the form

await *message* **when** *guard*

where *message* is a message (of form $X!m(\overline{y})$ or $X?m(\overline{y})$) and *guard* is a Boolean condition (referring to the variables of the agent and any parameters of the message). This statement is enabled in a state where there is an unhandled message m and the *guard* is true, and when enabled, it has the same effect as **await** *message*.

4a A Hoare Triple for Guarded Await (weight 5)

Formulate a Hoare triple of the form

$$\{...\}\ \textbf{await}\ X?m(\overline{y})\ \textbf{when}\ g\ \{...\}$$

(Continued on page 8.)

for the guarded await statement where the message part is $X?m(\overline{y})$ and the guard is the condition g.

If possible, make a left-constructive rule as well for the guarded await statement, i.e., a Hoare triple of the form

$$\{...\}\,\textbf{await}\ X?m(\overline{y})\ \textbf{when}\ g\,\{Q\}$$

where Q denotes an arbitrary postcondition.

Solution: $\{\forall X,\overline{y}.\ Q[h;(X \downarrow \mathit{this} : m(\overline{y}))/h]\}\,\textbf{await}\ X?m(\overline{y})\ \textbf{when}\ g\,\{g \wedge Q\}$

$\{\forall X,\overline{y}.\ g \Rightarrow Q[h;(X \downarrow \mathit{this} : m(\overline{y}))/h]\}\,\textbf{await}\ X?m(\overline{y})\ \textbf{when}\ g\,\{Q\}$

4b Programming with Agents (weight 15)

Program an agent SC taking care of the shower control, and program also an agent $Male$ corresponding to a male user process such that $Male$ agents (and female agents) interact with SC by message passing. You do not need to program the female agents as long as they are similar to male agents. Agent SC should have the form of a loop taking care of repeated user requests.

Solution:

```
agent Male [i=1..k]
// agent SC is the shower controller agent
 begin
 while true do
   send SC:req_male_shower;
   (  await SC:get_male_shower; shower; send SC:rel_male_shower
   [] await SC:get_shared_shower; shower; send SC:rel_shared_shower
   );
   eat; work; sleep
 od
end

agent SC
  begin
  agent X // agent variable
  Nat u, m, f // counters, all  0 initially
  while true do
  (  await X?req_male_shower   when u+m<2*n;
     if m<n then m:=m+1; send X:get_male_shower
     else        u:=u+1; send X:get_shared_shower fi
  []
     await X?req_female_shower when u+f<2*n;
     if f<n then f:=f+1; send X:get_female_shower
     else        u:=u+1; send X:get_shared_shower fi
```

(Continued on page 9.)

```
 []
    await X?rel_male_shower;   m:=m-1
 []
    await X?rel_shared_shower; u:=u-1
 []
    await X?rel_female_shower; f:=f-1
 ) od
end
```

4c Alphabet (weight 5)

What is the alphabet of the shower controller SC, i.e., write down the events that are visible for the shower controller.

Solution: $X \downarrow SC : req\_male\_shower$, $X \downarrow SC : req\_female\_shower$,

$SC \uparrow X : get\_male\_shower$, $SC \uparrow X : get\_shared\_shower$, $SC \uparrow X : get\_female\_shower$,

$X \downarrow SC : rel\_male\_shower$, $X \downarrow SC : rel\_shared\_shower$, $X \downarrow SC : rel\_female\_shower$.

where X is another (male or female) agent.

4d History Functions (weight 5)

Consider the local agent invariant

$$u = shared(h)$$

for SC where *shared* is a function over the local history h. Define the function *shared* so that it calculates the number of shared showers in use, based on the local history of SC (using no other variables).

Solution:

$shared(empty) = n$,
$shared(h; (X \downarrow SC : req\_male\_shower)) = shared(h)$,
$shared(h; (X \downarrow SC : req\_female\_shower)) = shared(h)$,

$shared(h; (SC \uparrow X : get\_male\_shower)) = shared(h)$,
$shared(h; (SC \uparrow X : get\_shared\_shower)) = shared(h) + 1$,
$shared(h; (SC \uparrow X : get\_female\_shower)) = shared(h)$,

$shared(h; (X \downarrow SC : rel\_male\_shower)) = shared(h)$,
$shared(h; (X \downarrow SC : rel\_shared\_shower)) = shared(h) - 1$,
$shared(h; (X \downarrow SC : rel\_female\_shower)) = shared(h)$.

(Continued on page 10.)

4e Verification of the Local Agent Invariant (weight 5)

Verify that

$$u = shared(h)$$

is a loop invariant for the main loop of SC, where *shared* is as above.

Solution:

```
while true do {$I$}
(   {$\forall\ X.\ u = shared(h;(X\downarrow\ SC : req\_male\_shower)$}
    await X?req_male_shower   when u+m<2*n;
    {$u = shared(h)\ \wedge\ u+m < 2*n$}
    {$if\ (f<n)\ then\ u = shared(h)\ else\ u+1 = shared(h)+1\ fi$}
    if m<n then m:=m+1; send X:get_male_shower
    else        u:=u+1; send X:get_shared_shower fi   {$u = shared(h)$}

[]
    await X?req_female_shower when u+f<2*n;
    if f<n then f:=f+1; send X:get_female_shower
    else        u:=u+1; send X:get_shared_shower fi
[]
    await X?rel_male_shower;   m:=m-1
[]
    {$\forall\ X.\ u-1 = shared(h;(X\downarrow SC : rel\_shared\_shower))$}
    await X?rel_shared_shower; u:=u-1 {$u = shared(h)$}
[]
    await X?rel_female_shower; f:=f-1
) {$I$}  od
```

First branch: We use the first rule for await-guard. It then remains to prove that its precondition is implied by the invariant, which is obvious since

$$shared(h;(X\downarrow SC : req\_male\_shower)) = shared(h)$$

and therefore the occurrence of X disappears and so does the quantifier. After the await-guard the postcondition obviously implies the generated condition.

The guarded receive statement does not affect the invariant. May assume the guard after the Then-branch is trivial, Verify $u = shared(h) \Rightarrow$ ..

Second branch is similar. Third and last branches are trivial since there is no change in u nor in $shared(h)$.

Second last branch: It remains to verify that the invariant implies
$\forall X.\ u-1 = shared(h;(X\downarrow SC : rel_shared_shower))$,
which is trivial since $shared(h;(X\downarrow SC : rel_shared_shower)) = shared(h)-1$ and then the quantifier can be removed since there is no occurrence of X.