Universitetet i Oslo
Institutt for Informatikk
Research group for Reliable Systems (PSY)



IN5170: Models of Concurrency

Fall 2024 28.11.2024

Issued: 28.11.2024

Merk oppgaven med at det er lov å tegne på papir!
Note that it is allowed to draw on paper!

1 General Questions (20p)

Exercise 1 (8p.) Consider the following program

```
Boolean x := true; y := false;
co
    while (x) {y:=true; y:=false} || < await(y) x:=false >
oc
```

1. Explain briefly what the program does. (2p.)
2. Explain briefly weak fairness. (1p.)
3. Will the program always terminate under weak fairness?
   Explain briefly your answer. (2p.)
4. Explain briefly strong fairness. (1p.)
5. Will the program always terminate under strong fairness?
   Explain briefly your answer. (2p.)

Exercise 2 (2p.) Explain the P and V operations in a semaphore.

Exercise 3 (2p.) Explain briefly the difference between the signalling disciplines of monitors, signal and wait (SW) and signal and continue (SC).

Exercise 4 (2p.) Explain briefly the difference between synchronous and asynchronous message passing.

Exercise 5 (2p.) Explain briefly the advantages of usage type systems compared to linear type systems.

Exercise 6 (1p.) Declare a channel using linear types that must not send, but receives once a struct of type `Msg`.

Exercise 7 (3p.) Given the following channel declaration using usage types:
`a := make(chan`$_{?.!.0 + !.0 + ?.0}$ `int).`

1. Explain briefly the indented behavior of the channel that is defined by the type. (2p.)
2. Does this channel declaration guarantee freedom of deadlocks? Explain briefly your answer. (1p.)

2 The Single-Lane Bridge (70p)



A bridge over a river is only wide enough to permit a single lane of traffic, consequently cars can only move concurrently if they are moving in the same direction. A problem occurs if two cars moving in different directions are on the bridge at the same time, and this needs to be avoided. To facilitate the exercise, let us refer to the cars moving from left to right as red cars and the ones moving from right to left as blue cars. It should be prevented that red and blue cars are on the bridge at the same time. However, there can be several red cars on the bridge at the same time OR several blue cars. If a car arrives at the bridge and cars of the opposite color are currently on the bridge, the car has to wait. Assume each car is a process and the problem is to prevent cars moving in different directions to concurrently access the shared resource (the bridge) at the same time. For the sake of simplicity, assume that cars do not change color/direction and therefore always approach the bridge from the same side.

As a summary, a solution to this problem should ensure the following:

- Several red cars can be on the bridge at the same time;
- Several blue cars can be on the bridge at the same time;
- There are never red cars AND blue cars on the bridge at the same time;
- If a car arrives at the bridge and cars of the opposite color are currently on the bridge, the car waits.

Exercise 8 (The Single-Lane Bridge using semaphores (10p.)) Complete the code below to provide a solution to the problem based on semaphores.

```
int     nr := 0;   # number of  active red cars that want to pass the bridge
int     nb := 0;   # number of  active blue cars that want to pass the bridge
sem     bridge := 1;   # lock for accessing the bridge
...

process RedCar [i=1 to M] {
  while (true) {
  ...
  # car is on the  bridge
  ...
  }
}

process BlueCar [i=1 to N] {
  while (true) {
  ...
  # car is on the  bridge
  ...
  }
}
```

Exercise 9 (Invariant for the Single-Lane Bridge (2p.)) What would be a good invariant for this exercise? Explain briefly you proposal.

Exercise 10 (The Single-Lane Bridge using monitors (10p.)) Consider monitors with the following operations:

```
cond cv;
wait(cv);
signal(cv);
signal_all(cv);
```

Assume there exists a bridge controller BC that keeps track of the accesses to the bridge. Complete the code below to provide a solution for the bridge controller based on monitors. Assume a signal and continue discipline.

```
monitor BC {  # Invariant:  ...
   int nr :=0, nb :=0;  # number of  active red and blue cars that want to pass the bridge
   ...

 procedure redEnter() { ...}

 procedure redExit() { ...}

 procedure blueEnter() { ... }

 procedure blueExit() {  ... }
}
```

Exercise 11 (The Single-Lane Bridge using monitors – continuation (2p.)) Assume there exists an implementation for the bridge controller BC as in Exercise 10. Complete the code below to provide a solution for both red and blue cars that call the bridge controller BC to access the bridge.

```
process RedCar [i=1 to M] {
  while (true) {
        call BC.  ...
         # car is on the  bridge
         ...
  }
}

process BlueCar [i=1 to N] {
  while (true) {
        ...
        # car is on the  bridge
        ...
  }
}
```

Exercise 12 ([The Single-Lane Bridge properties (4p.)) Do your solutions to Exercises **10** and **11** guarantee:

1. mutual exclusion? (1p.)
2. maintaining the invariant? (1p.)
3. absence of unnecessary delay? (2p.)

Explain each answer briefly.

Exercise 13 (The Fair Single-Lane Bridge Controller(12p.)) Use the monitor operations listed in Exercise 10, that is,

```
cond cv;
wait(cv);
signal(cv);
signal_all(cv);
```

Provide a solution to the single-lane bridge using a bridge controller (BC) that is fair in the sense that waiting cars will eventually be able to drive over the bridge. To do so, introduce counters for the waiting red and blue cars as well as a counter for the number of signalled cars.

Your solution should implement the following behavior: If red cars are already on the bridge and another red car arrives and wants to enter the bridge, the BC should let it pass if there are no blue cars waiting on the other side. If there are blue cars waiting to enter the bridge, the just arriving red car has to go in the waiting queue for red cars. If a blue car arrives while red cars are on the bridge, it goes in the waiting queue for blue cars. (Similarly, if blue cars are already on the bridge and another blue car arrives and wants to enter the bridge, the BC should let it pass if there are no red cars waiting on the other side. If there are red cars waiting to enter the bridge, the newly arriving blue car has to go in the waiting queue for blue cars. If a red car arrives while blue cars are on the bridge, it goes in the waiting queue for red cars.)

Signalling to waiting cars in the queues should also be implemented so that they do not have to wait forever. In particular, use a counter `ns` for the number of signalled cars to ensure that all signalled cars can pass the bridge before signalling the waiting cars on the other side, if any.

Assume a signal and continue discipline and complete the code below.

```
monitor BC {  # Invariant:  ...
 int nr :=0, nb :=0;  # number of  active red and blue cars that want to pass the bridge
 int nwr:=0, nwb:=0; # number of  waiting red and blue cars that want to pass the bridge
 int ns := 0; # number of signalled cars that can pass the bridge
 ...
 procedure redEnter() { ...}

 procedure redExit() { ...}

 procedure blueEnter() { ... }

 procedure blueExit() {  ... }
}
```

Exercise 14 (The Single-Lane Bridge using asynchronous channels (10p.)) Complete the code below, which is similar to the one provided in Exercise 8, to provide a solution for the single-lane bridge using asynchronous channels.

```
int     nr := 0;  # number of  active red cars that want to pass the bridge
int     nb := 0;  # number of  active blue cars that want to pass the bridge
chan    bridge(bool sig);  # lock for accessing the bridge
...

process RedCar [i=1 to M] {
    while (true) {
            ...
            # car is on the  bridge
            ...
    }
}

process BlueCar [i=1 to N] {
    while (true) {
            ...
            # car is on the  bridge
            ...
    }
}

process Bridge {
  send bridge(true);
  ...
}
```

Exercise 15 (The Single-Lane Bridge using channels (2p.)) Would your solution for Example 14 also work for with synchronous channels? Explain briefly your verdict.

Exercise 16 (The Single-Lane Bridge using channels via a central Bridge Controller (12p.)) Consider the data structure queue with the following operations:

```
empty(queue) # returns true if queue is empty
remove(queue, elem); # pops an element from the queue and writes it to elem
insert(queue, elem); # pushes elem to the queue
```

Provide a solution to the single-lane bridge problem with a central bridge controller (BC) similar to the one in Exercise 10. The BC should manage the entries and exits of cars to the bridge using asynchronous channels. Complete the code of the BC in the code skeleton below.

```
type car_type := enum(RED, BLUE); # types of cars
type access_type := enum(ENTER, EXIT); # the bridge can be entered or exited
int     nr := 0;  # number of  active red cars that want to pass the bridge
int     nb := 0;  # number of  active blue cars that want to pass the bridge

# channel for signalling entry/exit operation to the bridge
chan    access(int id, car_t car_type, access_t operation);

chan    reply[N+M](bool sig);  # channel for providing access to the bridge

process BC {
  queue rWCars;
  queue bWCars;
  while (true) {
    ...
  }
}

process RedCar [i=1 to M] {
  bool reply_sig;
  while (true) {
    send access(i, RED, ENTER);
    receive reply[i](reply_sig);
    # car is on the  bridge
    send access(i, RED, EXIT);
  }
}

process BlueCar [i=M+1 to N+M] {
  bool reply_sig;
  while (true) {
    send access(i, BLUE, ENTER);
    receive reply[i](reply_sig);
    # car is on the  bridge
    send access(i, BLUE, EXIT);
  }
}
```

Exercise 17 (Type Annotations for Channel Types (6p.)) Consider the following code snippet implemented in a Go-like programming language.

```
  func Car(id int, color string, signal chan carMsg, reply chan bool) {
      signal <- carMsg{id, color, "enter"}
      <-reply
      fmt.Printf("%s car %d on the bridge\n", color, id)
      signal <- carMsg{id, color, "exit"}
  }
```

Modify the channel types using (1) input/output modes (2p.), (2) linear types (2p.), and (3) usage types (2p.). The types should be chosen as restrictively as possible.

3 Type Systems (10p)

Exercise 18 (Type checking of channel types (10p.)) Consider the following typing trees for linear channel types. You find all typing rules in the Appendix. It is not required to complete incomplete typing trees, if it is not explicitly requested.

1. What are the values for Γ^1, Γ^1_1, and Γ^1_2 in the following typing tree? (3p.)

$$\dfrac{\dfrac{}{\Gamma^1 = \Gamma^1_1 + \Gamma^1_2} \quad \dfrac{\begin{matrix}\text{—}\\ \vdots\end{matrix}}{\Gamma^1_1 \vdash \texttt{sig} \texttt{<−true; skip: Unit}} \quad \dfrac{\begin{matrix}\text{—}\\ \vdots\end{matrix}}{\Gamma^1_2 \vdash \texttt{return <−sig: Unit}}}{\{\texttt{sig} \mapsto \textbf{chan}_{?1,!1}\ \texttt{bool}\} \vdash \texttt{go func( )\{sig <−true; skip\}( ); return <−sig : Unit}}\ \text{L7}$$

2. The following typing tree contains errors. Explain briefly what is wrong and how it should be corrected. (4p.)

$$\dfrac{\dfrac{}{\Gamma^2 = \Gamma^2_1 + \Gamma^2_2 + \Gamma^2_3} \quad \dfrac{\begin{matrix}\text{—}\\ \vdots\end{matrix}}{\Gamma^2_1 \vdash \texttt{v} > 0 : \texttt{bool}} \quad \dfrac{\begin{matrix}\text{—}\\ \vdots\end{matrix}}{\Gamma^2_2 \vdash \texttt{skip} : \texttt{Unit}} \quad \dfrac{\begin{matrix}\text{—}\\ \vdots\end{matrix}}{\Gamma^2_3 \vdash \texttt{v} =\texttt{<−sig; skip} : \texttt{Unit}} \quad \dfrac{\begin{matrix}\text{—}\\ \vdots\end{matrix}}{\Gamma^2_1 \vdash \texttt{skip} : \texttt{Unit}}}{\Gamma^2 \vdash \texttt{if(v > 0)\{skip\} else\{v =<−sig; skip\} skip} : \texttt{Unit}}\ \text{L10}$$

$\Gamma^2 = \{\texttt{v} \mapsto \texttt{bool}, \texttt{sig} \mapsto \textbf{chan}_{?1,!0}\ \texttt{bool}\}$
$\Gamma^2_1 = \{\texttt{v} \mapsto \texttt{bool}, \texttt{sig} \mapsto \textbf{chan}_{?0,!0}\ \texttt{bool}\}$
$\Gamma^2_2 = \{\texttt{sig} \mapsto \textbf{chan}_{?0,!0}\ \texttt{bool}\}$
$\Gamma^2_3 = \{\texttt{v} \mapsto \texttt{bool}, \texttt{sig} \mapsto \textbf{chan}_{?1,!0}\ \texttt{bool}\}$

3. Complete the following typing tree. Does the expression type check? (3p.)

$$\dfrac{}{\{\texttt{s} \mapsto \textbf{chan}_{?1,!0}\ \texttt{bool}, \texttt{r} \mapsto \textbf{chan}_{?0,!1}\ \texttt{bool}\} \vdash \texttt{←s: bool}}\ \text{L8}$$

Appendix

$$\frac{\mathsf{un}(\Gamma)}{\Gamma \vdash \textbf{skip} : \texttt{Unit}}\ \text{L-skip (L1)} \qquad \frac{\mathsf{un}(\Gamma)}{\Gamma \vdash \texttt{true} : \texttt{bool}}\ \text{L-true (L2)}$$

$$\frac{\mathsf{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}}\ \text{L-int (L3)} \quad \frac{\Gamma = \Gamma_1 + \Gamma_2 \qquad \Gamma_1 \vdash \texttt{e}: \texttt{ T} \qquad \mathsf{un}(\Gamma_2)}{\Gamma \vdash \textbf{return}\ \texttt{e} : \texttt{Unit}}\ \text{L-return (L4)}$$

$$\frac{\Gamma = \Gamma^1 + \Gamma^2 \qquad \Gamma^1 \vdash e_1 : \texttt{Int} \qquad \Gamma^2 \vdash e_2 : \texttt{Int}}{\Gamma \vdash e_1 + e_2 : \texttt{Int}}\ \text{L-add (L5)}$$

$$\frac{\Gamma = \Gamma_1 + \Gamma_2 \qquad \Gamma_1 \vdash \texttt{e}: \texttt{ T} \qquad \Gamma(\texttt{v}) = \texttt{T} \qquad \Gamma_2 \vdash \texttt{s} : \texttt{Unit}}{\Gamma \vdash \texttt{v = e; s} : \texttt{Unit}}\ \text{L-assign (L6)}$$

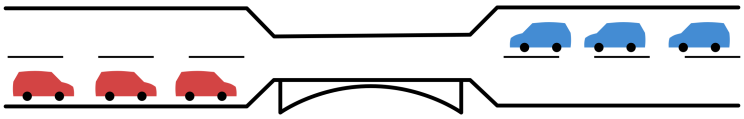
$$\frac{\Gamma = \Gamma_1 + \Gamma_2 \qquad \Gamma_1 \vdash \texttt{s}_1 : \texttt{Unit} \qquad \Gamma_2 \vdash \texttt{s}_2 : \texttt{Unit}}{\Gamma \vdash \textbf{go}\ \texttt{func()\{s}_1\texttt{\}(); s}_2 : \texttt{Unit}}\ \text{L-parallel (L7)}$$

$$\frac{\Gamma(\texttt{v}) = \textbf{chan}_{?1,!0}\ \texttt{T} \qquad \mathsf{un}(\Gamma[\texttt{v} \mapsto \textbf{chan}_{?0,!0}\ \texttt{T}])}{\Gamma \vdash \leftarrow\texttt{v}: \texttt{ T}}\ \text{L-receive (L8)}$$

$$\frac{\Gamma[\texttt{c} \mapsto \textbf{chan}_{?0,!0}\ \texttt{T}] = \Gamma_1 + \Gamma_2 \qquad \Gamma(\texttt{c}) = \textbf{chan}_{?0,!1}\ \texttt{T} \qquad \Gamma_1 \vdash \texttt{s} : \texttt{Unit} \qquad \Gamma_2 \vdash \texttt{e}: \texttt{ T}}{\Gamma \vdash \texttt{c <- e; s} : \texttt{Unit}}\ \text{L-send (L9)}$$

$$\frac{\Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3 \qquad \Gamma_1 \vdash \texttt{e} : \texttt{bool} \qquad \Gamma_2 \vdash \texttt{s}_1 : \texttt{Unit} \qquad \Gamma_2 \vdash \texttt{s}_2 : \texttt{Unit} \qquad \Gamma_3 \vdash \texttt{s}_3 : \texttt{Unit}}{\Gamma \vdash \texttt{if(e)\{s}_1\texttt{\} else\{s}_2\texttt{\} s}_3 : \texttt{Unit}}\ \text{L-branch (L10)}$$

Single-Lane Bridge Example



A bridge over a river is only wide enough to permit a single lane of traffic, consequently cars can only move concurrently if they are moving in the same direction. A problem occurs if two cars moving in different directions are on the bridge at the same time, and this needs to be avoided. To facilitate the exercise, let us refer to the cars moving from left to right as red cars and the ones moving from right to left as blue cars. It should be prevented that red and blue cars are on the bridge at the same time. However, there can be several red cars on the bridge at the same time OR several blue cars. If a car arrives at the bridge and cars of the opposite color are currently on the bridge, the car has to wait. Assume each car is a process and the problem is to prevent cars moving in different directions to concurrently access the shared resource (the bridge) at the same time. For the sake of simplicity, assume that cars do not change color/direction and therefore always approach the bridge from the same side.

As a summary, a solution to this problem should ensure the following:

- Several red cars can be on the bridge at the same time;
- Several blue cars can be on the bridge at the same time;
- There are never red cars AND blue cars on the bridge at the same time;
- If a car arrives at the bridge and cars of the opposite color are currently on the bridge, the car waits.

Typing Rules

$$\frac{\mathsf{un}(\Gamma)}{\Gamma \vdash \textbf{skip} : \texttt{Unit}}\ \text{L-skip (L1)} \qquad \frac{\mathsf{un}(\Gamma)}{\Gamma \vdash \texttt{true} : \texttt{bool}}\ \text{L-true (L2)}$$

$$\frac{\mathsf{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}}\ \text{L-int (L3)} \quad \frac{\Gamma = \Gamma_1 + \Gamma_2 \qquad \Gamma_1 \vdash \texttt{e}\colon\ \texttt{T} \qquad \mathsf{un}(\Gamma_2)}{\Gamma \vdash \textbf{return}\ \texttt{e} : \texttt{Unit}}\ \text{L-return (L4)}$$

$$\frac{\Gamma = \Gamma^1 + \Gamma^2 \qquad \Gamma^1 \vdash e_1 : \texttt{Int} \qquad \Gamma^2 \vdash e_2 : \texttt{Int}}{\Gamma \vdash e_1 + e_2 : \texttt{Int}}\ \text{L-add (L5)}$$

$$\frac{\Gamma = \Gamma_1 + \Gamma_2 \qquad \Gamma_1 \vdash \texttt{e}\colon\ \texttt{T} \qquad \Gamma(\texttt{v}) = \texttt{T} \qquad \Gamma_2 \vdash \texttt{s} : \texttt{Unit}}{\Gamma \vdash \texttt{v}\ =\ \texttt{e};\ \texttt{s} : \texttt{Unit}}\ \text{L-assign (L6)}$$

$$\frac{\Gamma = \Gamma_1 + \Gamma_2 \qquad \Gamma_1 \vdash \texttt{s}_1 : \texttt{Unit} \qquad \Gamma_2 \vdash \texttt{s}_2 : \texttt{Unit}}{\Gamma \vdash \textbf{go}\ \texttt{func()\{s}_1\texttt{\}()};\ \texttt{s}_2 : \texttt{Unit}}\ \text{L-parallel (L7)}$$

$$\frac{\Gamma(\texttt{v}) = \textbf{chan}_{?1,!0}\ \texttt{T} \qquad \mathsf{un}(\Gamma[\texttt{v} \mapsto \textbf{chan}_{?0,!0}\ \texttt{T}])}{\Gamma \vdash \leftarrow\texttt{v}\colon\ \texttt{T}}\ \text{L-receive (L8)}$$

$$\frac{\Gamma[\texttt{c} \mapsto \textbf{chan}_{?0,!0}\ \texttt{T}] = \Gamma_1 + \Gamma_2 \qquad \Gamma(\texttt{c}) = \textbf{chan}_{?0,!1}\ \texttt{T} \qquad \Gamma_1 \vdash \texttt{s} : \texttt{Unit} \qquad \Gamma_2 \vdash \texttt{e}\colon\ \texttt{T}}{\Gamma \vdash \texttt{c}\ <\!\!-\ \texttt{e};\ \texttt{s} : \texttt{Unit}}\ \text{L-send (L9)}$$

$$\frac{\Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3 \qquad \Gamma_1 \vdash \texttt{e} : \texttt{bool} \qquad \Gamma_2 \vdash \texttt{s}_1 : \texttt{Unit} \qquad \Gamma_2 \vdash \texttt{s}_2 : \texttt{Unit} \qquad \Gamma_3 \vdash \texttt{s}_3 : \texttt{Unit}}{\Gamma \vdash \texttt{if(e)\{s}_1\texttt{\}\ else\{s}_2\texttt{\}\ s}_3 : \texttt{Unit}}\ \text{L-branch (L10)}$$