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



IN5170: Models of Concurrency

Fall 2025 11.12.2025

Issued: 11.12.2025

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

```
int x := 0; y := 0;
co
    x := y - 1; // process 1 (P1)
    ||
    y := y + 1; // process 2 (P2)
oc
```

1. How many interleavings does the following program has if each process consists of two atomic actions? (1p.)
2. Briefly explain the property that must hold for two processes to be *interference-free*. (1p.)
3. Is the program interference-free?
   Briefly explain your answer. (2p.)
4. Briefly explain the AMO-property. (2p.)
5. Do the two assignments in P1 & P2 fulfill the AMO-property?
   Briefly explain your answer. (2p.)

Exercise 2 (2p.) Briefly explain the purpose of a monitor invariant and when it must hold.

Exercise 3 (2p.) Briefly explain the difference between linear types and usage types?

Exercise 4 (4p.) Given the following channel declaration using usage types:
`a := make(chan`$_{?.!.0 + !.0 + ?.0}$ `int).`

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

Exercise 5 (4p.) The following program written in Rust contains an error:

```
1   fn f(vec : &Vec<i32>) {
2     let x = &vec[0];
3     println!("{}", *x + 1);
4   }
5
6   fn main() {
7     let vec = vec![1,2];
8     thread::spawn(move || f(&vec));
9     println!("{}", vec[0]);
10  }
```

1. Briefly explain the error and provide the line(s) that create(s) the error. (2p.)
2. Explain how to fix the error and provide the correct code. (2p.)

2 The One-Way Elevator in an Observation Tower (50p)

An observation tower has exactly one elevator that transports visitors directly from the base of the tower to the top. The elevator only travels in one direction, meaning visitors can only use it to go up, but must take the stairs to get down.

The elevator has a maximum capacity C and only travels from the bottom to the top when it is fully loaded. When the maximum capacity is reached or the elevator is traveling up or down, newly arriving visitors leave the tower immediately.

The elevator starts at the base of the tower and immediately travels from top to bottom after all visitors have exited at the top.

Exercise 6 (The One-Way Elevator using Semaphores (10p.)) Complete the code below to provide a solution to the problem based on semaphores. You should provide code for synchronization and locking only at places, where it is indicated with three dots (...). You might want to add more than one line. Your solution should be provided in the await-language.

```
int     nv := 0;  # number of visitors that are in the elevator
...     # declare here as many semaphores as required to implement your solution

process Visitor [i=1 to N] {
  while (true) {
  ...
  if (nv < C)
  {
    nv := nv + 1;
    ...
    if (nv == C) {
      ... // elevator is full
    }
    ... // visitor arrives on the top
    // The visitor is at the top of the tower & walks down the stairs; no implementation is requi
  } else {
    ...
    // visitor leaves or takes the stairs, no implementation required
  }
  }
}

process Elevator {
  while (true) {
  ... // elevator goes up
  nv = 0;
  ... # elevator is back downstairs
  }
}
```

Exercise 7 (The One-Way Elevator using Semaphores - Deadlock & Starvation (2p.)) Is your solution to 6 free of deadlocks or starvation? Briefly explain your answer.

Exercise 8 (Invariant for the One-Way Elevator (2p.)) What would be a good general invariant to prevent too many visitors from being in this one-way elevator?

Exercise 9 (The One-Way Elevator using Monitors (10p.)) Consider monitors with the following operations:

```
cond cv; // declaration of conditional variable
wait(cv);
signal(cv);
signal_all(cv);
```

Assume there exists an elevator controller EC that controls the elevator and keeps track of access to the elevator. Complete the code below to provide a solution for the EC based on monitors. Remember to specify an invariant. Assume a signal and continue discipline. Provide your solution in the await-language.

```
monitor EC {  // Invariant:  ...
   int nv = 0;  // number of visitors that are in the elevator
   ... // declare here as many conditional variables as required to implement your solution

 procedure goingUp() { ... }

 procedure goingDown() {
  nv := 0;
  ...
 }

 procedure takeElevator() {

  if (nv < C) {
    nv := nv + 1;
    if (nv == C) {
      ... // elevator should go upwards
    }
    ...
  }

  }

}
```

Exercise 10 (The One-Way Elevator using Monitors – continuation (2p.)) Assume there exists an implementation for the elevator controller EC as in Exercise 9. Complete the code below to provide a solution for visitors and the elevator that call the EC to access and operate the elevator.

```
process Visitor [i=1 to N] {
  while (true) {
    call EC.takeElevator() // call EC
  }
}

process Elevator {
  while (true) {
        call EC.goingUp();
        // elevator is on the top
        call EC.goingDown();
  }
}
```

Exercise 11 (Properties of the One-Way Elevator (4p.)) your solutions to Exercises **9** and **10** guarantee:

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

Explain each answer briefly.

Exercise 12 (The One-Way Elevator with a Queue using Monitors (8p.)) Use the monitor operations listed in Exercise 9, that is,

```
        cond cv; # declaration of conditional variable
        wait(cv);
        signal(cv);
        signal_all(cv);
```

Extend your solution to Exercise 9 so that visitors can queue up for the elevator if the elevator is fully occupied or operating. Assume that the queue can be infinitely long. You must not use the data type `queue`.

Assume a signal and continue discipline and complete the code below. The three dots (...) provide some indication where you should provide code for synchronization. You might want to add more than one line. Remember to specify an invariant. Provide your solution in the await-language.

```
  monitor EC {  // Invariant:  ...
   int nv = 0;  // number of visitors that are in the elevator
   ...

 procedure goingUp() { ... }

 procedure goingDown() {
  nv := 0;
  ...
 }

 procedure takeElevator() {
    ... // add queuing
    nv := nv + 1;
    if (nv == C) {
      ... # elevator should go upwards
    ...
  }

  }

}
```

Exercise 13 (Fairness of the One-Way Elevator with a Queue using Monitors (2p.)) Briefly explain if your solution to Exercise 12 is fair.

Exercise 14 (The One-Way Elevator using Asynchronous Channels (8p.)) Complete the code below, which is similar to the one provided in Exercise 6, to provide a solution for the one-way elevator using asynchronous channels.

```
    int     nv := 0;   # number of visitors that are in the elevator
    ... // declare here as many channels as required to implement your solution

process Visitor [i=1 to N] {
  while (true) {
  ...
  if (nv < C)
  {
    nv := nv + 1;
    ...
    if (nv == C) {
      ... # elevator should go upwards
    }
    ... # visitor arrives on the top
    # visitor is on the top of the tower & takes the stairs downwards, no implementation required
  } else {
    ...
    # visitor leaves or takes the stairs, no implementation required
  }
  }
}

process Elevator {
  while (true) {
  ... # elevator goes up
  nv = 0;
```

```
  ... # elevator is again down
  }
}
```

Exercise 15 (The One-Way Elevator using Channels (2p.)) Would your solution for Example 14 also work for with synchronous channels? Briefly explain your verdict.

3 Concurrency in Rust (6p.)

Exercise 16 ((6p.)) Consider the following code in Rust. Does the code provide

1. mutual exclusion?
2. absence of unnecessary delay?
3. absence of deadlocks

Briefly explain each answer. If a property is violated, explain the error and how the code must be corrected to preserve the property.

```
1
2  fn main() {
3      // Two shared mutex-protected values
4      let a = Arc::new(Mutex::new(0));
5      let b = Arc::new(Mutex::new(0));
6
7      // Clone Arcs for thread 1
8      let a1 = Arc::clone(&a);
9      let b1 = Arc::clone(&b);
10
11     let t1 = thread::spawn(move || {
12         let mut lock_a = a1.lock().unwrap(); // Thread 1 locks A
13         let mut lock_b = b1.lock().unwrap(); // Thread 1 locks B
14         *lock_a += 1;
15         *lock_b += 1;
16     });
17
18     // Clone Arcs for thread 2
19     let a2 = Arc::clone(&a);
20     let b2 = Arc::clone(&b);
21
22     let t2 = thread::spawn(move || {
23         let mut lock_b = b2.lock().unwrap(); // Thread 2 locks B
24         let mut lock_a = a2.lock().unwrap(); // Thread 2 locks A
25         *lock_b += 1;
26         *lock_a += 1;
27     });
28
29     // Wait for both threads
30     let _ = t1.join();
31     let _ = t2.join();
32 }
```

4 Type Systems (24p.)

Exercise 17 (Channel Types (5p.)) Write a go-like function that separates even and odd received integer values. The function receives from a channel an integer and sends to two channels the number. One output channel receives only even numbers, whereas the other channel receives only odd numbers. The syntax for receiving from a channel `ch` is `x := <- ch` and the syntax to send to a channel `ch` is `ch <- x`.

```
go func splitEvenOdd (... /* add channels for receiving and sending */){
... // Add your go code here
}
```

Exercise 18 (Channel Types with Input/Output Mode (2p.)) Modify your solution to Exercise 17 such that channel types consider input/output modes. Provide type annotations that are as restrictive as possible.

```
go func splitEvenOdd (... /* add channels for receiving and sending */){
... // Add your go code here
}
```

Exercise 19 (Linear and Normal Types for Channels (2p.)) Modify your solution to Exercise 17 to consider linear and normal channel types. Your type annotations should be as restrictive as possible.

Exercise 20 (Usage Types for Channels (4p.)) Modify your solution to Exercise 17 to consider usage types. For this, assume that you receive on your input channel exactly four values, where two of them are odd and two are even. Your type annotations should be as restrictive as possible.

```
go func splitEvenOdd (input chan<?.?.?.?.0> int, even chan<!.!.0> int, odd chan<!.!.0> int){
 for i := 0; i < 4; i++ {
  val := <-input
   if val%2 == 0 {
     even <- val
   } else {
     odd <- val
   }
  }
}
```

Exercise 21 (Type Checking of Channel Types (11p.)) 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.)

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

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

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

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

3. Complete the following typing tree. Does the expression type check? (4p.)

$$\frac{}{\{\texttt{s} \mapsto \textbf{chan}_{?1,!0}\ \texttt{bool}, \texttt{r} \mapsto \textbf{chan}_{?0,!1}\ \texttt{bool}\} \vdash \leftarrow\texttt{s}\colon\ \texttt{bool}}\ \text{L8}$$

Appendix

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

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

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

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

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

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

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

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

Typing Rules

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

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

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

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

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

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

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

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