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




# IN5170: Models of Concurrency

Fall 2022 02.12.2022

**Issued: 02.12.2022**

**Merk oppgaven med at det er lov å tegne på papir!**

## 1 General Questions

**Exercise 1 (7p.)** We consider a program state in which the two program variables x and y both have the value 5. Consider the following program:

```
< x := x + y > || < y := y + x>
```

**1a)** Is your program intererence-free? Explain your answer (1p.)
**1b)** Do the processes satisfy the at-most-once property? Explain your answer (1p.)
**1c)** What are possible pre- and post-conditions for the program? (2p.)
**1d)** Prove your post-condition using program logic for all states that satisfy the pre-condition (3p.)

**The proof rules of Program Logic:**

$$\frac{}{\{P[e/x]\}\ x := e\ \{P\}}\ \textsc{Assign} \qquad \frac{P' \Rightarrow P \quad \{P\}\ S\ \{Q\} \quad Q \Rightarrow Q'}{\{P'\}\ S\ \{Q'\}}\ \textsc{Consequence}$$

$$\frac{\{P\}\ S\ \{Q\} \quad \{P'\}\ S\ \{Q'\}}{\{P \wedge P'\}\ S\ \{Q \wedge Q'\}}\ \textsc{Conjunction} \qquad \frac{\{P\}\ S_1\ \{R\} \quad \{R\}\ S_2\ \{Q\}}{\{P\ S_1; S_2\ \{Q\}}\ \textsc{Seq}$$

$$\frac{}{\{P\}\ \textbf{skip}\ \{P\}}\ \textsc{Skip} \qquad \frac{\{P \wedge B\}\ S\ \{Q\} \quad P \wedge \neg B \Rightarrow Q}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ S\ \textbf{fi}\ \{Q\}}\ \textsc{Cond1}$$

$$\frac{\{P \wedge B\}\ S_1\ \{Q\} \quad \{P \wedge \neg B\}\ S_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{Q\}}\ \textsc{Cond2} \qquad \frac{\{I \wedge B\}\ S\ \{I\}}{\{I\}\ \textbf{while}\ B\ \textbf{do}\ S\ \{I \wedge \neg B\}}\ \textsc{While}$$

$$\frac{\{P_i\}\ S_i\ \{Q_i\}\ \text{are interference-free}}{\{P_1 \wedge \ldots \wedge P_n\}\ \textbf{co}\ S_1 || \ldots || S_n\ \textbf{oc}\ \{Q_1 \wedge \ldots \wedge Q_n\}}\ \textsc{Par}$$

**Exercise 2 (Comparing Types (4p.))** Compare usage types and binary session types (both as introduced in the lecture) for channels.

1. Give an example of behavior that can be expressed using a usage type, but not a binary session type, or argue why this is not possible.
2. Give an example of behavior that can be expressed using a binary session type, but not a usage type, or argue why this is not possible.

**Exercise 3 (2p.)** Explain the difference between weak and strong fairness.

## 2 Semaphores

We consider a barbershop with one barber and a waiting room with $n$ chairs for waiting customers ($n$ may be 0). The following rules apply:

- If there are no customers, the barber falls asleep
- A customer must wake the barber if he is asleep
- If a customer arrives while the barber is working, the customer leaves if all chairs are occupied and sits in an empty chair if it's available
- When the barber finishes a haircut, he inspects the waiting room to see if there are any waiting customers and falls asleep if there are none

**Exercise 4 (Barbershop (10p.))** Complete the code below to provide a solution based on semaphores that ensures the following requirements:

- the barber never sleeps while there are waiting customers and
- there is never more than $n$ customers waiting in the waiting room.

Briefly explain why your solution satisfies these requirements.

```
int freeSeats = n;

process Customer {
  while (true) {


  }
}
process Barber {
  while (true) {


  }
}
```

**Exercise 5 (Fairness of Barbershop (2p.))** Is your solution to Exercise **4** fair? Explain briefly.

**Exercise 6 (Barbershop with priorities (12p.))** The barber shop introduces an "express" category of customers, who should have priority over regular customers. Implement a solution such that priority customers can bypass regular customers, using the same semaphores as in Exercise **4** above. Provide a solution to the Barbershop with priorities by completing the code below. Briefly explain how your solution gives priority to express customers.

```
int seatsInBarbershop = n;
int freeSeats = seatsInBarbershop;

process RegularCustomer {
  while (true) {


  }
}


process PriorityCustomer {
  while (true) {


  }
}


process Barber {
  while (true) {


  }
}
```

**Exercise 7 (Properties of barbershop with priorities (10p.))** Does your solution to Exercise **6** guarantee:

1. mutual exclusion? (2p.)
2. absence of deadlock? (2p.)
3. absence of unnecessary delay? (2p.)
4. fairness? If the solution is not fair, explain how you could make it fair. (4p.)

Explain each answer briefly.

# 3 Monitors

We consider monitors with the following operations:

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

**Exercise 8 (Barbershop monitor with priorities (12p.))** Use the monitor operations listed above to make a monitor solution to the priority customer barber shop of Exercise 6. Provide your solution by completing the code below and explain briefly how your solution gives priority to express customers.

```
monitor Barbershop {
  int seatsInBarbershop = n;
  int nr = 0;  // number of regular customers
  int np = 0;  // number of priority customers

  procedure barber(){
    while (true) {


    }
  }

  procedure regularCustomer() {
```

```
    while (true) {


    }
  }

  procedure priorityCustomer() {
    while (true) {


  }
}
```

**Exercise 9 (Monitor invariant (2p.))** What could be a monitor invariant for the Barbershop monitor? Explain briefly why the monitor invariant holds for your monitor solution.

# 4 Asynchronous message passing

We consider a producer consumer problem in which multiple producers send data to a consumer via a buffer channel. Let Producer1 send value 1 on the buffer channel, Producer2 send value 2 on the buffer, etc.

**Exercise 10 (Producer consumer (5p.))** Give a solution to the producer consumer problem outlined above for two producer processes and one consumer process, using asynchronous channels for synchronising the processes. The producers should be synchronised such that the consumer receives a 1 and a 2 every second time (e.g., 1 2 1 2 1 2 1 . . . ). Provide a solution by completing the code below and explain briefly why your solution solves this producer consumer problem.

```
chan buf(int i);

procedure Producer1 {
  while (true) {


  }
}

procedure Producer2 {
  while (true) {


  }
}

procedure Consumer {
  while (true) {


  }
}
```

**Exercise 11 (4p.)** Give a solution to the producer consumer problem outlined above using asynchronous channels for synchronising processes that scales to N producer processes. You may add additional channels and processes as necessary. Explain your solution.

# 5 Types

**Exercise 12 (Binary Session Types (8p.))** Consider the following Go-like code.

```
1 func main(b bool, val1 int, val2 int){
2   (c, c_dual) = make(chan T, chan T̄);
3   go f(c_dual);
4   if(b) {
```

```
5       (r, r_dual) = make(chan S, chan S̄);
6       c <- l1;
7       c <- r;
8       c <- val1;
9       r_dual <- val2;
10      if( <-r_dual ) println("success");
11      else           println("failure");
12    }
13    else c <- abort;
14  }
15
16  func f(c chan T̄){
17    switch <-c {
18      case l1:
19         ret = <- c;
20         param1 = <-c;
21         param2 = <- ret;
22         ret <- param1 + param2 >= 0;
23      case abort:
24      }
25  }
```

1. Give the session types for `T` and `S` so the program is well-typed.
2. Give the duals of `T` and `S`.
3. Give a subtype of `T` and subtype of $\overline{\texttt{T}}$.

**Exercise 13 (Linearity (12p.))** 1. Consider the following Go-like code. Does it type-check? If no, give the line of the statement where type-checking fails, the reason and the line where the misused channel is declared. If yes, give the type derivation tree given the rules below (you can skip the variable declaration and add them into the type environment directly).

```
1  func main(){
2    c = make(chan<!1,?1> int);
3    d = make(chan<!1,?1> int);
4    e = 0;
5    go func {
6      go func {
7        d <- 1; skip;
8      };
9      c <- (<-d); skip;
10   };
11   e = <-c; skip;
12 }
```

2. Consider the following code. Does it type-check? If no, give the line of the statement where type-checking fails, the reason and the line where the misused channel is declared. If yes, give the type derivation tree given the rules below (you can skip the variable declaration and add them into the type environment directly).

```
1  func main(){
2    c = make(chan<!1,?1> int)
3    d = make(chan<!1,?1> chan<!1,?1> int)
4    e = make(chan<!1,?1> int);
5    f = 0;
6    res = 0;
7    ret = 0;
8    go func {
9      go func {
10       d <- e; e <- 1; skip;
11     }
12     res = <-d;
13     ret = 0;
14     if((<-res) < 0) {
15        ret = -1;
16     } else {
17        ret = 0;
18     }
19     c <- ret*(<-e); skip;
20   };
21   f = <-c; skip;
22 }
```

$$\frac{\Gamma_1 \vdash \mathtt{s}_1 : \mathtt{Unit} \qquad \Gamma_2 \vdash \mathtt{s}_2 : \mathtt{Unit}}{\Gamma_1 + \Gamma_2 \vdash \mathtt{go\ func}\ \{\mathtt{s}_1\};\ \mathtt{s}_2 : \mathtt{Unit}}\ \text{L-PARALLEL}$$

$$\frac{\Gamma_1 \vdash \mathtt{e} : T \qquad \Gamma(\mathtt{v}) = T \qquad \Gamma_2 \vdash \mathtt{skip} : \mathtt{Unit}}{\Gamma_1 + \Gamma_2 \vdash \mathtt{v} = \mathtt{e}; \mathtt{skip} : \mathtt{Unit}}\ \text{L-ASSIGN-NO-SUB}$$

$$\frac{T' \preceq T}{\Gamma + \{c : \mathtt{chan}_{?1,!0} T'\} \vdash {<}{-}\mathtt{c} : T}\ \text{L-READ} \qquad \frac{\Gamma \vdash v : T' \qquad T' \preceq T}{\Gamma + \{c : \mathtt{chan}_{?0,!1} T\} \vdash \mathtt{c}\ {<}{-}\ \mathtt{v} : \mathtt{Unit}}\ \text{L-WRITE}$$

$$\frac{\Gamma_1 \vdash \mathtt{e}_1 : \mathtt{int} \qquad \Gamma_2 \vdash \mathtt{e}_2 : \mathtt{int}}{\Gamma_1 + \Gamma_2 \vdash \mathtt{e}_1 < \mathtt{e}_2 : \mathtt{Bool}}\ \text{L-ADD}$$

$$\frac{\mathtt{un}(\Gamma) \qquad n \in \mathbb{Z}}{\Gamma \vdash n : \mathtt{int}}\ \text{L-LITERAL} \qquad \frac{\mathtt{un}(\Gamma)}{\Gamma \vdash \mathbf{skip} : \mathtt{Unit}}\ \text{L-SKIP}$$

$$\frac{\Gamma_1 \vdash \mathtt{e} : \mathtt{Bool} \qquad \Gamma_2 \vdash \mathtt{s}_1;\ \mathtt{s}_3 : \mathtt{Unit} \qquad \Gamma_2 \vdash \mathtt{s}_2; \mathtt{s}_3 : \mathtt{Unit}}{\Gamma_1 + \Gamma_2 \vdash \mathtt{if\ e\ then}\ \{\mathtt{s}_1\}\ \mathtt{else}\ \{\mathtt{s}_2\}\ \mathtt{s}_3 : \mathtt{Unit}}\ \text{L-IF}$$

**Exercise 14 (Rust (10p.))** Consider the following two methods.

```
fn f(write_ref : &mut Vec<i32>){
  write_ref[0] = 0;
}
fn g(read_ref : &Vec<i32>){
  println!("{}", read_ref[0]);
}
```

1. Does type checking succeed? Annotate for each line of code below the current owner of the created vector.

```
fn main() {
  let mut vec = vec![1,2,3];
  f(&mut vec);
  thread::spawn(move || g(&vec));
}
```

2. For which of the below versions of `main` ([1], [2], [3]) does type checking fail? For each version explain why it fails (if it does) or why the restrictions on ownership and references are satisfied (if is succeeds)

```
fn main() { //[1]
  let mut vec = vec![1,2,3];
  f(&mut vec);
  thread::spawn(move || g(&vec));
  g(&vec);
}
fn main() { //[2]
  let mut vec = vec![1,2,3];
  f(&mut vec);
  g(&vec);
  g(&vec);
}
fn main() { //[3]
  let mut vec = vec![1,2,3];
  f(&mut vec);
  f(&mut vec);
  g(&vec);
}
```