IN5170 Oral Exam

Eduard Kamburjan   Einar Broch Johnsen

2021

Exercise 1

a) Explain the program

```
int x = 0;
bool flag = true;

co
<await x>2; > flag = false;
||
while (flag) <x=x+1;>
oc
```

b) Does the program terminate?

c) Will the program terminate under some fairness conditions?

d) What are safety and liveness properties?

Exercise 2a

a) Explain semaphores

b) Which semaphores would you use for Producer/Consumer, and why?

c) Is this an example of a split binary semaphore?

d) Does the solution scale to multiple producers or multiple consumers?

```
T buf[n];
int front := 0, rear := 0;
sem empty := n; XXX
sem full := 0;  XXX

process Producer {
  while (true) {
    P(empty); XXX
     buff[rear] := data;
     rear := (rear+1) % n;
    V(full);  XXX
  }
}

process Consumer {
  while (true) {
    P(full);  XXX
     result := buff[front];
     front := (front +1) % n;
    V(empty); XXX
  }
}
```

Figure 1: Remove lines with XXX on the whiteboard

Exercise 2b

How would you program Producer Consumer with monitors?

Exercise 3

Asynchronous Programming

- How do actors communicate?
- How does that map to OO languages like C# ?
- What are the problems with a paradigm that uses only async. message sending? (call-back hell)
- What are solutions? (futures, channels, cooperative scheduling)
- Compare futures and promises.
- Compare futures and linear channels for call-backs.

Exercise 4

Consider the following language

$$e ::= v \mid 1 \mid e ** e \mid v++$$

```
monitor Queue {
  T buf[n];
  int front := 0, rear := 0;
  int filled := 0;

  cond notempty; XXX
  cond notfull; XXX

  procedure Produce() {
    while (filled = n) { wait (notfull) };XXX
    filled := filled + 1;
    signal (notempty) ;XXX
  }

  procedure Consume() {
    while (filled = 0) { wait (notempty) };XXX
    filled := filled − 1;
    signal (notfull);XXX
  }
}
```

Figure 2: Remove on the whiteboard

Where $e**e$ executes its parameters in parallel and then multiplies then, and $v++$ is the usual incremental return with side-effect. Consider the types

$$T ::= \mathbf{int} \mid \mathbf{int}_1 \mid \mathbf{int}_0$$

Where the indexed types are linear. Consider the following rules.

$$\frac{\mathsf{un}(\Gamma)}{\Gamma \vdash 1 : \mathbf{int}}$$

$$\frac{\mathsf{un}(\Gamma[v \mapsto \mathbf{int}_0]) \qquad \Gamma(v) = \mathbf{int}_1}{\Gamma \vdash v : \mathbf{int}}$$

$$\frac{\mathsf{un}(\Gamma[v \mapsto \mathbf{int}_0]) \qquad \Gamma(v) = \mathbf{int}_1}{\Gamma \vdash v++ : \mathbf{int}}$$

$$\frac{\Gamma = \Gamma_1 + \Gamma_2 \qquad \Gamma_1 \vdash e_1 : \mathbf{int} \qquad \Gamma_2 \vdash e_2 : \mathbf{int}}{\Gamma \vdash e_1 ** e_2 : \mathbf{int}}$$

- What is the task of $\mathsf{un}(\Gamma)$?
- Define $\Gamma = \Gamma_1 + \Gamma_2$
- Explain how concurrency can introduce bugs (data race on $v**v++$)
- Explain how linearity solves this

- Explain how the type system enforces linearity
- How does linearity occur in Rust? How does it differ from the linearity in this system? How does it help with concurrency?
- What is the difference between a linear and a session type? What are the commonalities?