FPV Final Week *

Zixuan Fan

Jul 2022

1 Introduction

This slides covers the final part of the lecture Functional Programming and Verification, i.e. parallel programming. 2 Modules Thread and Event are being elaborated and discussed. Furthermore, common structural of parallel programming in FP are also provided. The tutorial also saves time for QA sessions.

2 Thread

Different from imperative programming. We don't declare and initialize a new thread. Rather create a thread directly and have it as the returned value of the creation function.

```
val create: ('a -> 'b) -> 'b -> t
let _ = create id 1;;
```

*All contents are based on the Artemis exercises and lecture slides of Prof. Seidl. No content is guaranteed to be totally correct.

```
3
4  (*in imperative programming*)
5  initialize(thread, func, args)
6
7  val self: unit -> t;;
8  val id: t -> int;;
9
10 val exit: unit -> unit;;
11 val join: t -> unit;;
12 val yield: unit -> unit;;
```

To access the current thread and its id, the corresponding functions are generated. Similar to most parallel programming features, **exit, kill, join, yield** are all available, whereas some of them are not used as a rule of thumb. Additionally, a waiting feature is implemented but is nevertheless not discussed in both lecture and tutorial.

3 Event

The event module enables the communication between threads and parallel controlling. Still, most features of parallel programming are preserved, but implemented in a functional way.

The module allows for communication using the type channel. A channel of a certain type **'a** only sends and receives messages of this type. Attempts of communication using other types will not compile. Intuitively, the **send** and **receive** functions are implemented.

To avoid problems regrading to the synchronisation, the function **sync** is used, its locks the current event. More functions to specify the parallel controlling are also available.

```
type 'a channel;;

val new_channel: unit -> 'a channel;;

val send: 'a channel ->'a -> unit event;;
val receive: 'a channel -> 'a event;;
```

4 Parallel Controlling Styles

Both of the following design style can be seen in the first tutorial exercise. The second exercise, blogging system is based on the first style and is most likely to be part of the exam, if parallel programming is included.

4.1 Main Thread Orchestration

As told from the name, this structure is centralized. The control is based on the communication between main thread and sub-threads. It is easy to implement but not quite efficient, sometimes even worse than the single thread implementation.

4.2 Self Organisation

On the contrary, this style is somewhat self-organising and hence somewhat decentralized. It is although efficient when dealing with considerable amounts of thread, the implementation is hard and annoying. Not likely to exist in the exam.

Wish you all good scores in the exam and a nice vocation!