FPV Week 9 *

Zixuan Fan

JUL 2022

1 Introduction

In this week, we are dealing with the module features of Ocaml In comparison, it is a feature similar to classes in object-oriented programming. This sheet also includes some error handling features that are not well discussed in the previous week.

2 Module

A module is comparable to a class in object-oriented programming, but they are not the same concept. It should be perceived as a set of functions and values where everything is integrated and easily callable. No actually object can be created, because there is no constructor.

2.1 Syntax to make a module

```
module MyModule = struct
   type a = int
   ...
   let add a b = a + b
   ...
```

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

```
end
```

Its type is shown by

```
module MyModule :
sig
   type a = int
   ...
   val add : a -> a -> a
   ...
end
```

2.2 Syntax of Signature

Signatures can be used as a template or an abstract type to simplify the implementation of similar functions.

```
module type MySig = sig
type t
...
val to_string : t -> string
...
end
```

To use a signature we have to explicitly show that our implementation use this signature.

```
module MyImpl:MySig with type t = int = sturct
type t = int
...
val to_string = string_of_int
...
end
```

To use a module we have to open it, as we usually open the List module. We can also use the keyword include to inherit the previous module or signature.

```
open List

module MySig2 = sig
  include Mysig
```

```
  type t2
  ...
  val to_int : t -> int
  ...
end
```

2.3 Functor

If you have learned about inheritence of **c++**, you should be familiar with this feature. A functor is a signature that accepts another module for its implementation.

```
module type MyFunctor(MyModule : MySig) with t = Mysig.t = sig
 type t = MySig.t
 ...
end
```

Some more features of functor can be seen in the tutorial exercise.

3 Exceptions

As you may know, throwing an exception is not functional and is regarded as bad behaviour in functional programming, hence it was not covered that much in the lecture. So a brief and optional part is covered here for a revision and better understanding. A list of pre-defined exceptions can be found here in the Ocaml Library. The try-with grammar is actually an implementation of pattern matching, so it is some functional thing out of non-functional behaviour.

```
  let res =
    try let n = f 0 with Division_by_Zero -> Printf("Division by
    zero\n")
        Failure -> Prinf("Unknown Failure")
  in n

```

4 Contest

As you may have noticed, the homework of this week also covers a contest, whose deadline is out of this reason delayed for 1 week. The chair encourages participation and prepared some prizes for the top 3. As a hint, check min-max algorithm and alpha-beta algorithm out. They should be helpful for the race.