Proof assistants

TD 4- Monads and Modules

1 Monades

1.1 Exceptions

Consider the type of binary trees with leaves labelled by natural numbers:

```
Inductive tree : Type :=
    Leaf : nat -> tree | Node : tree -> tree -> tree.
```

1- Write a function expecting a tree as argument and returning either `Some n` where n is the product of all leaves if its greater than 0, or `None` if one of the leaves is null.
2- Define the exception monad. Beside the usual monad operations, it should have 2 operations: `raise` to raise an exception, and `try` to catch it. Hint: use the `option` type.
3- Rewrite the fonction computing the product of leaves, this time using the exception mechanism to return 0 as soon as a leaf is null.

1.2 Non determinism

Write the non-determinism monad, that lets you execute non deterministically a task among a finite number of possible tasks. Beside the usual monad operations, it should have one operation `par` of type $M(A) \to M(A) \to M(A)$ such that `par e1 e2` executes non deterministically either `e1` or `e2`.

2 Modules

1- Write a module signature representing a carrier type and a preorder on that type.
2- Write a module signature representing a functor expecting as argument a carrier type and a boolean-valued order, and producing a finite set structure implementing the following operations: membership, empty set, adding an element, removing an element, together with the basic properties of these operations.
3- Implement this functor signature using lists to represent finite sets.