Programmation Fonctionnelle – Exceptions

Adrien Durier

26 septembre 2023

Exceptions

Les exceptions

- Le mécanisme d'exception permet notamment de gérer le problème des fonctions partielles
- Si l'exception n'est pas rattrapée (voir ci-après), le programme se termine

Les exceptions : *rattrapage*

Une exception est rattrapée à l'aide de la construction :

`try` e_1 `with E ->` e_2

Les exceptions : *rattrapage*

Une exception est rattrapée à l'aide de la construction :

$$\texttt{try}\ e_1\ \texttt{with}\ \texttt{E}\ \texttt{->}\ e_2$$

```
# let test x y =
  try
    let q = x / y in Printf.printf "quotient = %d\n" q
  with
    Division_by_zero -> Printf.printf "error\n" ;;
- val test : int -> int -> unit = <fun>
```

Les exceptions : *rattrapage*

Une exception est rattrapée à l'aide de la construction :

$$\texttt{try}\ e_1\ \texttt{with}\ \texttt{E}\ \texttt{->}\ e_2$$

```
# let test x y =
  try
    let q = x / y in Printf.printf "quotient = %d\n" q
  with
    Division_by_zero -> Printf.printf "error\n" ;;
- val test : int -> int -> unit = <fun>
```

- si e_1 s'évalue normalement vers v, alors v est renvoyée
- si e_1 lève l'exception E, celle-ci est rattrapée et e_2 est évaluée
- si e_1 lève une autre exception que E, celle-ci est propagée

```
# test 4 2
- quotient = 2
# test 4 0
- error
```

Les exceptions : déclaration et levée

```
# exception Fin ;;
- exception Fin
# raise Fin; print_int 10 ;;
- Exception: Fin.
# exception E of int ;;
- exception E of int
# let f x = if x = 0 then raise (E(x)) else 10 / x ;;
- val f : int -> int = <fun>
# f 0 ;;
- Exception: E(0)
```

- Un type d'exception est défini à l'aide du mot-clé `exception`.
- Son nom commence toujours par une majuscule et elle peut attendre des arguments.
- On *lève* une exception à l'aide de la fonction raise.