TP de programmation n° 3

Julien Reichert

Mardi 22 octobre 2013

Ce TP sera exceptionnel, à n'en point douter.

1 Les exceptions

Si vous n'êtes pas un programmeur parfait, vous avez sans doute fait face au cours de vos tests à des exceptions.

Qu'est-ce qu'une exception ?

```
0/0;;
Exception: Division_by_zero.
```

Ceci est une exception. OCaml ne sait pas quoi faire quand on lui demande d'exécuter l'instruction ci-dessus [1].

```
0. + 0;;
Error: This expression has type float
but an expression was expected of type int
```

Ceci n'est pas une exception. C'est une erreur. Et en plus, cela signifie que vous n'avez pas été attentifs au premier TP, donc je suis triste.

```
let tab_reponse = Array.make 42 0 in tab_reponse.(42);;
Exception: Invalid_argument "index out of bounds".

List.hd [];;
Exception: Failure "hd".
```

1. seul Chuck Norris sait quoi faire

Ce sont deux exceptions. Il s'agit d'un argument invalide et d'une erreur précisée.

Les exceptions peuvent être vues comme un type somme enrichi. L'un des éléments de ce type est donc Division_by_zero, d'autres sont Invalid_argument of string et Failure of string, on peut en créer manuellement par

`Exception Perdu` [2]

Les exceptions sont déclenchées automatiquement le cas échéant ou en le demandant expressément :

```
raise Perdu;;
raise (Failure "I lost the game!";;
failwith "He perdido!";;
raise (Invalid_argument "Ich habe verloren!");;
invalid_arg "Ho perduto!";;
```

Remarquez que les lignes 2 et 3 ainsi que 4 et 5 sont équivalentes.

Les exceptions que vous définissez ont le type de votre choix, comme dans le cas des types somme.

```
Exception Reponse of int;;
raise (Reponse 42);;
```

2 Rattraper une exception

Bon, assez de bêtises, c'est l'heure de réparer tout ça.

Pour rattraper une exception, on utilise la structure `try ...with` qui crée un contexte où toutes les exceptions seront gérées a posteriori, en redéclenchant potentiellement des exceptions non rattrapées.

```
let recherche_paresseuse pred tab =
try
for i = 0 to Array.length tab - 1 do
if pred tab.(i) then failwith "Trouve" done; false
with Failure "Trouve" -> true;;
```

Dans cet exemple, dès que `pred` est satisfait, l'erreur précisée `"Trouve"` est déclenchée, ce qui fait sortir du corps de la fonction. On rattrape l'exception en retournant `true`, et si elle n'est pas déclenchée le résultat est `false`. Si une

2. J'ai perdu !

autre exception est déclenchée, comme le filtrage ne la rattrape pas, elle est en quelque sorte transmise.

Une autre version de la même fonction, pour varier la syntaxe :

```
let recherche_paresseuse pred tab =
try
for i = 0 to Array.length tab do
if pred tab.(i) then failwith "Trouve" done
with Failure "Trouve" -> print_string "true"
| Invalid_argument _ -> print_string "false";;
```

Attention, il faut que le résultat à l'intérieur du `try` et le résultat après le `with` soient de même type, d'où le recours à une fonction d'écriture ici. Quand une exception se déclenche, il n'y a toutefois pas de conflit de type, voilà pourquoi OCaml ne fait pas d'avertissement précisant que le filtrage n'est pas exhaustif.

Une dernière remarque : sans rattrapage, une exception se propage naturellement jusqu'au toplevel, en interrompant au passage toutes les sous-procédures imbriquées.

3 La pile d'appels

Afin de connaître le comportement détaillé d'une fonction, on accède à sa pile d'appels en demandant à la tracer.

```
let rec fact = function
|0 -> 1
|n -> n * fact (n-1);;

#trace fact;; (* fact is now traced. *)

fact 3;;
fact <-- 3
fact <-- 2
fact <-- 1
fact <-- 0
fact --> 1
fact --> 1
fact --> 2
fact --> 6

#untrace fact;; (* fact is no longer traced. *)
```

4 Quelques autres exceptions de base

Voici d'autres exceptions classiques : `Not_found`, qui est par exemple déclenchée par `List.find`, `Exit`, `Break` et `Abort` pour « sortir plus ou moins brutalement d'une boucle », `Sys_error of string` qui concerne le système d'exploitation, `Match_failure of string * int * int` qui signale un échec du filtrage à une position précisée dans un fichier et `End_of_file`.

5 Exercices

Exercice 1 : Écrire une fonction `try_finalize` spécifiée de la façon suivante : `try_finalize f x g y` applique la fonction `f` à la valeur `x`.
- si une exception est levée, alors on évalue `g y` et on relève ensuite l'exception,
- si aucune exception n'est levée, on évalue aussi `g y` et on retourne la valeur retournée par `f x`.
- si une exception est levée par `g y`, cette exception est levée par la fonction try_finalize.

Source : http ://form-ocaml.forge.ocamlcore.org/html/formation_ocaml.html#htoc28

Exercice 2 : Utiliser des exceptions dans diverses fonctions de base qui ont déjà été rencontrées dans les TP précédents.