Foundations of Computer Science

Exceptions and error handling

Dr. Robert Harle & Dr. Jeremy Yallop

2020–2021

During a computation, what if **something goes wrong**?

```
3 / 0    (* division by zero *)
```

```
hd []    (* pattern matching falure *)
```

Exception handling allows us to **recover** from these.

Raising an exception **abandons** the current expression

```
raise Failure
```

Handling the exception attempts an **alternative**

```
try f () with Failure -> g ()
```

```
In[1]: exception Failure
Out[1]: exception Failure
In[2]: exception NoChange of int
Out[2]: exception NoChange of int
In[3]: raise Failure
  Out: Exception: Failure.
```

Each exception declaration introduces a distinct type of exception that can be handled separately.

Exceptions are like enumerations and can have data attached to them.

```
In[1]: exception Failure
Out[1]: exception Failure
In[2]: exception NoChange of int
Out[2]: exception NoChange of int
In[3]: raise Failure
  Out: Exception: Failure.
```

Each exception declaration introduces a distinct type of exception that can be handled separately.

Exceptions are like enumerations and can have data attached to them.

```
In[1]: exception Failure
Out[1]: exception Failure
In[2]: exception NoChange of int
Out[2]: exception NoChange of int
In[3]: raise Failure
  Out: Exception: Failure.
```

Each exception declaration introduces a distinct type of exception that can be handled separately.

Exceptions are like enumerations and can have data attached to them.

```
In[1]: exception Failure
Out[1]: exception Failure
In[2]: exception NoChange of int
Out[2]: exception NoChange of int
In[3]: raise Failure
  Out: Exception: Failure.
```

Each exception declaration introduces a distinct type of exception that can be handled separately.

Exceptions are like enumerations and can have data attached to them.

```
 In[1]: exception Failure
Out[1]: exception Failure
 In[2]: exception NoChange of int
Out[2]: exception NoChange of int
 In[3]: raise Failure
   Out: Exception: Failure.
```

Each exception declaration introduces a distinct type of exception that can be handled separately.

Exceptions are like enumerations and can have data attached to them.

```
 In[1]: exception Failure
Out[1]: exception Failure
 In[2]: exception NoChange of int
Out[2]: exception NoChange of int
 In[3]: raise Failure
   Out: Exception: Failure.
```

Each exception declaration introduces a distinct type of exception that can be handled separately.

Exceptions are like enumerations and can have data attached to them.

```
 In[1]: exception Failure
Out[1]: exception Failure
 In[2]: exception NoChange of int
Out[2]: exception NoChange of int
 In[3]: raise Failure
   Out: Exception: Failure.
```

Each exception declaration introduces a distinct type of exception that can be handled separately.

Exceptions are like enumerations and can have data attached to them.

Install exception handler for enclosing block:

```
In[4]: try
         print_endline "pre exception";
         raise (NoChange 1);
         print_endline "post exception";
       with NoChange _ ->
         print_endline "handled a NoChange exception"
       Line 3, characters 5-23:
       Warning 21: this statement never returns
                   (or has an unsound type.)

Out[4]: pre exception
        handled a NoChange exception
        - : unit = ()
```

`raise` dynamically jumps to the nearest `try/with` handler that matches that exception

OCaml does not mark functions to indicate that exceptions might be raised.

Install exception handler for enclosing block:

```
In[4]: try
         print_endline "pre exception";
         raise (NoChange 1);
         print_endline "post exception";
       with NoChange _ ->
         print_endline "handled a NoChange exception"
       Line 3, characters 6-23:
       Warning 21: this statement never returns
                   (or has an unsound type.)

Out[4]: pre exception
        handled a NoChange exception
        - : unit = ()
```

`raise` dynamically jumps to the nearest `try/with` handler that matches that exception

OCaml does not mark functions to indicate that exceptions might be raised.

Install exception handler for enclosing block:

```
In[4]: try
         print_endline "pre exception";
         raise (NoChange 1);
         print_endline "post exception";
       with NoChange _ ->
         print_endline "handled a NoChange exception"
       Line 3, characters 5-23:
       Warning 21: this statement never returns
                   (or has an unsound type.)

Out[4]: pre exception
        handled a NoChange exception
        - : unit = ()
```

`raise` dynamically jumps to the nearest `try/with` handler that matches that exception

OCaml does not mark functions to indicate that exceptions might be raised.

Install exception handler for enclosing block:

```
In[4]: try
         print_endline "pre exception";
         raise (NoChange 1);
         print_endline "post exception";
       with NoChange _ ->
         print_endline "handled a NoChange exception"
       Line 3, characters 5-23:
       Warning 21: this statement never returns
                   (or has an unsound type.)

Out[4]: pre exception
        handled a NoChange exception
        - : unit = ()
```

`raise` dynamically jumps to the nearest `try/with` handler that matches that exception

OCaml does not mark functions to indicate that exceptions might be raised.

```
In[5]: exception Change

Out[5]: exception Change

In[6]: let rec change till amt =
         match till, amt with
         | _, 0 -> []
         | [], _ -> raise Change (* Backtrack *)
         | c::till, amt ->
             if amt < 0 then raise Change (* Backtrack *)
             else
               try (* Attempt the solution *)
                   c :: change (c::till) (amt - c)
               with Change ->
                   (* Remove some change and retry if stuck *)
                   change till amt

Out[6]: val change : int list -> int -> int list = <fun>
```

```
In[5]: exception Change

Out[5]: exception Change

In[6]: let rec change till amt =
         match till, amt with
         | _, 0 -> []
         | [], _ -> raise Change (* Backtrack *)
         | c::till, amt ->
             if amt < 0 then raise Change (* Backtrack *)
             else
               try (* Attempt the solution *)
                   c :: change (c::till) (amt - c)
               with Change ->
                   (* Remove some change and retry if stuck *)
                   change till amt

Out[6]: val change : int list -> int -> int list = <fun>
```

```
In[5]:  exception Change

Out[5]: exception Change

In[6]:  let rec change till amt =
          match till, amt with
          | _, 0 -> []
          | [], _ -> raise Change (* Backtrack *)
          | c::till, amt ->
              if amt < 0 then raise Change (* Backtrack *)
              else
                try (* Attempt the solution *)
                  c :: change (c::till) (amt - c)
                with Change ->
                    (* Remove some change and retry if stuck *)
                    change till amt

Out[6]: val change : int list -> int -> int list = <fun>
```

```
In[5]: exception Change

Out[5]: exception Change

In[6]: let rec change till amt =
         match till, amt with
         | _, 0 -> []
         | [], _ -> raise Change (* Backtrack *)
         | c::till, amt ->
                 if amt < 0 then raise Change (* Backtrack *)
                 else
                   try (* Attempt the solution *)
                       c :: change (c::till) (amt - c)
                   with Change ->
                       (* Remove some change and retry if stuck *)
                       change till amt

Out[6]: val change : int list -> int -> int list = <fun>
```

```
In[5]: exception Change

Out[5]: exception Change

In[6]: let rec change till amt =
         match till, amt with
         | _, 0 -> []
         | [], _ -> raise Change (* Backtrack *)
         | c::till, amt ->
                 if amt < 0 then raise Change (* Backtrack *)
                 else
                   try (* Attempt the solution *)
                       c :: change (c::till) (amt - c)
                   with Change ->
                       (* Remove some change and retry if stuck *)
                       change till amt

Out[6]: val change : int list -> int -> int list = <fun>
```

Change with backtracking

change [5; 2] 6

Change with backtracking

change [5; 2] 6 ⇒ `try 5::change [5; 2] 1 with Change -> change [2] 6`

Change with backtracking

change [5; 2] 6 ⇒ `try 5::change [5; 2] 1 with Change -> change [2] 6`

⇒ `try 5::(try 5::change [5; 2] (-4) with Change -> change [2] 1)`

`with Change -> change [2] 6`

Change with backtracking

change [5; 2] 6 ⇒ **try** 5::change [5; 2] 1 **with** Change -> change [2] 6

⇒ **try** 5::(**try** 5::change [5; 2] (-4) **with** Change -> change [2] 1)
with Change -> change [2] 6

⇒ **try** 5::(change [2] 1) **with** Change -> change [2] 6

Change with backtracking

change [5; 2] 6 ⇒ **try** 5::change [5; 2] 1 **with** Change -> change [2] 6

⇒ **try** 5::(**try** 5::change [5; 2] (-4) **with** Change -> change [2] 1)
with Change -> change [2] 6

⇒ **try** 5::(change [2] 1) **with** Change -> change [2] 6

⇒ **try** 5::(**try** 2::change [2] (-1) **with** Change -> change [] 1)
with Change -> change [2] 6

Change with backtracking

```
change [5; 2] 6 ⇒ try 5::change [5; 2] 1 with Change -> change [2] 6
                ⇒ try 5::(try 5::change [5; 2] (-4) with Change -> change [2] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [2] 1) with Change -> change [2] 6
                ⇒ try 5::(try 2::change [2] (-1) with Change -> change [] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [] 1) with Change -> change [2] 6
```

Change with backtracking

```
change [5; 2] 6 ⇒ try 5::change [5; 2] 1 with Change -> change [2] 6
                ⇒ try 5::(try 5::change [5; 2] (-4) with Change -> change [2] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [2] 1) with Change -> change [2] 6
                ⇒ try 5::(try 2::change [2] (-1) with Change -> change [] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [] 1) with Change -> change [2] 6
                ⇒ change [2] 6
```

Change with backtracking

```
change [5; 2] 6 ⇒ try 5::change [5; 2] 1 with Change -> change [2] 6
                ⇒ try 5::(try 5::change [5; 2] (-4) with Change -> change [2] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [2] 1) with Change -> change [2] 6
                ⇒ try 5::(try 2::change [2] (-1) with Change -> change [] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [] 1) with Change -> change [2] 6
                ⇒ change [2] 6
                ⇒ try 2::change [2] 4 with Change -> change [] 6
```

Change with backtracking

```
change [5; 2] 6 ⇒ try 5::change [5; 2] 1 with Change -> change [2] 6
                ⇒ try 5::(try 5::change [5; 2] (-4) with Change -> change [2] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [2] 1) with Change -> change [2] 6
                ⇒ try 5::(try 2::change [2] (-1) with Change -> change [] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [] 1) with Change -> change [2] 6
                ⇒ change [2] 6
                ⇒ try 2::change [2] 4 with Change -> change [] 6
                ⇒ try 2::(try 2::change [2] 2 with Change -> change [] 4)
                  with Change -> change [] 6
```

Change with backtracking

```
change [5; 2] 6 ⇒ try 5::change [5; 2] 1 with Change -> change [2] 6
                ⇒ try 5::(try 5::change [5; 2] (-4) with Change -> change [2] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [2] 1) with Change -> change [2] 6
                ⇒ try 5::(try 2::change [2] (-1) with Change -> change [] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [] 1) with Change -> change [2] 6
                ⇒ change [2] 6
                ⇒ try 2::change [2] 4 with Change -> change [] 6
                ⇒ try 2::(try 2::change [2] 2 with Change -> change [] 4)
                  with Change -> change [] 6
                ⇒ try 2::(try 2::(try 2::change [2] 0 with Change -> change [] 2)
                           with Change -> change [] 4)
                  with Change -> change [] 6
```

Change with backtracking

```
change [5; 2] 6 ⇒ try 5::change [5; 2] 1 with Change -> change [2] 6
                ⇒ try 5::(try 5::change [5; 2] (-4) with Change -> change [2] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [2] 1) with Change -> change [2] 6
                ⇒ try 5::(try 2::change [2] (-1) with Change -> change [] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [] 1) with Change -> change [2] 6
                ⇒ change [2] 6
                ⇒ try 2::change [2] 4 with Change -> change [] 6
                ⇒ try 2::(try 2::change [2] 2 with Change -> change [] 4)
                  with Change -> change [] 6
                ⇒ try 2::(try 2::(try 2::change [2] 0 with Change -> change [] 2)
                           with Change -> change [] 4)
                  with Change -> change [] 6
                ⇒ try 2::(try 2::[2] with Change -> change [] 4) with Change -> change [] 6
```

Change with backtracking

```
change [5; 2] 6 ⇒ try 5::change [5; 2] 1 with Change -> change [2] 6
                ⇒ try 5::(try 5::change [5; 2] (-4) with Change -> change [2] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [2] 1) with Change -> change [2] 6
                ⇒ try 5::(try 2::change [2] (-1) with Change -> change [] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [] 1) with Change -> change [2] 6
                ⇒ change [2] 6
                ⇒ try 2::change [2] 4 with Change -> change [] 6
                ⇒ try 2::(try 2::change [2] 2 with Change -> change [] 4)
                  with Change -> change [] 6
                ⇒ try 2::(try 2::(try 2::change [2] 0 with Change -> change [] 2)
                           with Change -> change [] 4)
                  with Change -> change [] 6
                ⇒ try 2::(try 2::[2] with Change -> change [] 4) with Change -> change [] 6
                ⇒ try 2::[2; 2] with Change -> change [] 6
```

Change with backtracking

```
change [5; 2] 6 ⇒ try 5::change [5; 2] 1 with Change -> change [2] 6
                ⇒ try 5::(try 5::change [5; 2] (-4) with Change -> change [2] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [2] 1) with Change -> change [2] 6
                ⇒ try 5::(try 2::change [2] (-1) with Change -> change [] 1)
                  with Change -> change [2] 6
                ⇒ try 5::(change [] 1) with Change -> change [2] 6
                ⇒ change [2] 6
                ⇒ try 2::change [2] 4 with Change -> change [] 6
                ⇒ try 2::(try 2::change [2] 2 with Change -> change [] 4)
                  with Change -> change [] 6
                ⇒ try 2::(try 2::(try 2::change [2] 0 with Change -> change [] 2)
                           with Change -> change [] 4)
                  with Change -> change [] 6
                ⇒ try 2::(try 2::[2] with Change -> change [] 4) with Change -> change [] 6
                ⇒ try 2::[2; 2] with Change -> change [] 6
                ⇒ [2; 2; 2]
```