

Portage de matériels pédagogiques sur la plateforme Learn-OCaml

Loïc Sylvestre

5 juillet 2019

Learn-OCaml

- ▷ Environnement pour la correction automatisée d'exercices de programmation en OCaml
 - ▶ logiciel libre (licence BSD)
 - ▶ projet de la **Fondation OCaml**
 - ▶ initialement conçu pour un MOOC

Introduction to Functional Programming in OCaml

- ▶ puis utilisé en séances de travaux pratiques à **PARIS 7**
 - « *Les équipes enseignantes ont pu traduire leurs énoncés de travaux dirigés dans le système sans difficultés* »

(Bozman, Canou, Di Cosmo, Couderc, Gesbert, Henry, Le Fessant, Mauny, Morel, Peyrot. **Learn-OCaml : un assistant à l'enseignement d'OCaml, JFLA 2019**)

Learn-OCaml

Sujet de mon stage de L3

- ▷ encadré par Emmanuel Chailloux

Point de départ : les supports de MPIL (3I008)

- ▷ séances de TD/TP + projets
- ▷ modèles fonctionnel, impératif et objet + interopérabilité
 - ▶ traits de programmation relativement avancés
 - ▶ motivent des expérimentations autour de Learn-OCaml
 - ▶ mettent à l'épreuve le système de correction automatisée

Plan

Partie I : Le fonctionnement de Learn-OCaml

- ▷ illustré par un exemple

Partie II : Les limites du système de correction

- ▷ illustrées par un contreexemple

Partie III : Des améliorations

- ▷ Faciliter l'écriture des exercices
- ▷ Introduire de nouvelles techniques de correction
- ▷ Un exemple détaillé

Partie I : Le fonctionnement de Learn-OCaml

Une boucle d'interaction nichée dans le navigateur

« L'évaluation de code OCaml repose sur le toplevel [...] pour compiler à la volée les phrases envoyées au toplevel vers JavaScript et les évaluer dynamiquement. »

(Canou, Bozman, Henry. Sous le capot du MOOC OCaml, JFLA 2016)

Des exercices, chaque exercice est décrit par 6 fichiers

- ▷ **descr.md** (l'énoncé)
- ▷ **prelude.ml**, **prepare.ml**, **template.ml**, **solution.ml** et **test.ml**

Déroulement d'un exercice

Lorsque l'étudiant commence un exercice

- ▷ **prelude.ml** et **prepare.ml** sont évalués dans le toplevel
- ▷ L'étudiant voit l'énoncé et le **prelude**
- ▷ Il rédige sa réponse à partir d'un **patron (template.ml)**

NB : **prepare.ml** est invisible dans l'application Learn-OCaml.

→ introduit des définitions sans en dévoiler l'implémentation.

Phase de correction

- ▷ à la demande de l'étudiant (à condition que son code compile)
- ▷ Nouveau toplevel dans lequel sont évalués successivement :
 - **prelude.ml** et **prepare.ml**
 - le code de l'étudiant encapsulé dans un module **Code**
 - **solution.ml** encapsulé dans un module **Solution**
 - des utilitaires
 - le fichier **test.ml**

La phase de correction

```
⟨prelude.ml⟩ ;;
```

```
⟨prepare.ml⟩ ;;
```

```
module Code = struct ⟨le code de l'étudiant⟩ end ;;
```

```
module Solution = struct ⟨solution.ml⟩ end ;;
```

```
module Introspection = ... ;;
```

```
let code_ast = ... ;;
```

```
let results = ref None ;;
```

```
let timeout = ... ;;
```

```
module Test_lib = ... ;;
```

```
module Report = ... ;;
```

```
⟨test.ml⟩ ;;
```

Les sources d'un exercice : *Peano*

▷ **descr.md**

```
# Peano
Définir la fonction
of_int : int → peano
```

▷ **prelude.ml**

```
type peano = Z | S of peano
```

▷ **prepare.ml**

```
exception Undefined
let ⊥ _ = raise Undefined
```

▷ **template.ml**

```
let of_int = ⊥
```

▷ **solution.ml**

```
let rec of_int = function
  | 0 → Z
  | n → S (of_int (n-1))
```

▷ **test.ml**

```
let report =
  Test_lib.ast_sanity_check
    code_ast
    make_report

Test_lib.set_result report
```

NB : **make_report** est une fonction spécifique à chaque exercice.

Son type est `unit → Report.t`

Construction du rapport de correction

```
let make_report () =
  let results =
    Test_lib.test_function_1_against_solution
      [%ty: int -> peano] "of_int"
      ~sampler:Test_lib.sample_int
      ~gen:5
      [0;3;7;10]
  in
  [ Report.section "Q1 : of_int" results ]

let report =
  Test_lib.ast_sanity_check
    code_ast
    make_report ;;

Test_lib.set_result report
```

Peano : l'énoncé et le prélude



The screenshot shows the Learn OCaml web interface. At the top, there is a navigation bar with the Learn OCaml logo, 'OCaml PRO' badge, and buttons for 'Exercices', 'Compiler', and 'Noter!'. Below this is a secondary navigation bar with buttons for 'Éditeur', 'Exercice', 'Toplevel', 'Rapport', and 'Détails'. The main content area is titled 'Exercice 1 : Les entiers de Peano'. The exercise title 'Peano' is displayed in a large font. Below the title, the instruction reads: 'Définir la fonction `of_int` de type `int -> peano`'. At the bottom, there is a section for the OCaml prelude, showing the code: `type peano = Z | S of peano`. A 'Cacher' button is visible on the right side of the prelude section.

Learn OCaml **OCaml PRO** Exercices Compiler Noter! iswim

Éditeur Exercice Toplevel Rapport Détails

Exercice 1 : Les entiers de Peano

Peano

Définir la fonction `of_int` de type `int -> peano`

Prélude OCaml [↳ Cacher](#)

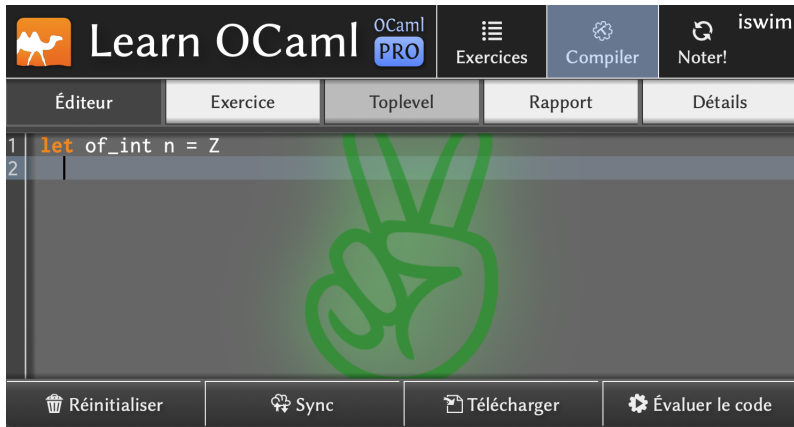
```
type peano = Z | S of peano
```

Peano : le template



The screenshot displays the Learn OCaml web interface. At the top, there is a navigation bar with the following elements from left to right: the Learn OCaml logo (a camel), the text "Learn OCaml", the "OCaml PRO" badge, a menu icon labeled "Exercices", a gear icon labeled "Compiler", and a refresh icon labeled "Noter!" with the username "iswim". Below this is a secondary navigation bar with tabs: "Éditeur" (selected), "Exercice", "Toplevel", "Rapport", and "Détails". The main area is a code editor showing a single line of OCaml code: `1 let of_int = __`. At the bottom, there is a footer bar with four buttons: "Réinitialiser" (trash icon), "Sync" (cloud icon), "Télécharger" (download icon), and "Évaluer le code" (gear icon).

Peano : la réponse de l'étudiant



The screenshot shows the Learn OCaml web interface. At the top, there is a navigation bar with the Learn OCaml logo (a camel), the text "Learn OCaml", and "OCaml PRO". To the right are buttons for "Exercices", "Compiler" (highlighted in blue), and "Noter!". Below this is a secondary navigation bar with tabs for "Éditeur", "Exercice", "Toplevel", "Rapport", and "Détails". The main area is a code editor with two lines of code: "1 let of_int n = Z" and "2". A large green peace sign watermark is overlaid on the editor. At the bottom, there is a footer bar with buttons for "Réinitialiser", "Sync", "Télécharger", and "Évaluer le code".

Learn OCaml OCaml PRO

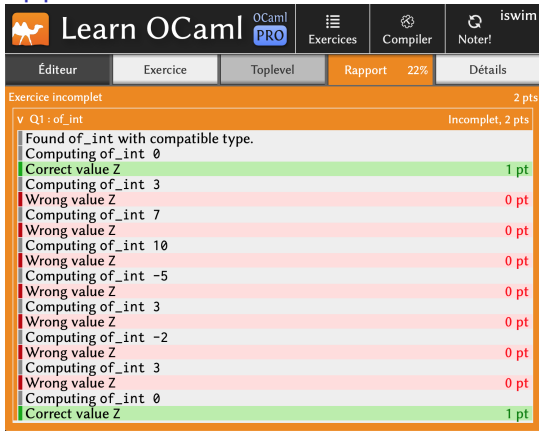
Exercices Compiler Noter! iswim

Éditeur Exercice Toplevel Rapport Détails

```
1 let of_int n = Z
2
```

Réinitialiser Sync Télécharger Évaluer le code

Peano : le rapport de correction



The screenshot shows the Learn OCaml interface. At the top, there are navigation buttons for 'Exercices', 'Compiler', and 'Noter!'. Below that, a progress bar shows 'Rapport 22%'. The main content area displays the results for an exercise named 'of_int'. The exercise is marked as 'Incomplet, 2 pts'. The results are as follows:

Test Case	Result	Points
Found of_int with compatible type.	Success	
Computing of_int 0	Success	
Correct value Z	Success	1 pt
Computing of_int 3	Success	
Wrong value Z	Failure	0 pt
Computing of_int 7	Success	
Wrong value Z	Failure	0 pt
Computing of_int 10	Success	
Wrong value Z	Failure	0 pt
Computing of_int -5	Success	
Wrong value Z	Failure	0 pt
Computing of_int 3	Success	
Wrong value Z	Failure	0 pt
Computing of_int -2	Success	
Wrong value Z	Failure	0 pt
Computing of_int 3	Success	
Wrong value Z	Failure	0 pt
Computing of_int 0	Success	
Correct value Z	Success	1 pt

```
let make_report () =  
  let results =  
    Test_lib.test_function_1_against_solution  
      [%ty: int -> peano] "of_int"  
      ~sampler:Test_lib.sample_int  
      ~gen:5  
      [0;3;7;10] in  
  [ Report.section "Q1 : of_int" results ]
```

Partie II : Les limites du système de correction

Comment traiter les fonctions polymorphes ?

Canou, Bozman, Henry. Sous le capot du MOOC OCaml, JFLA 2016

« *L'astuce est très simple. [...] Il suffit de demander deux instances de la fonction. Par exemple, si l'étudiant doit redéfinir*

`sort : α list \rightarrow α list`

on effectuera une moitié des cas de tests avec

`test_function_1 [%ty : int list \rightarrow int list] "sort"`

et une autre avec

`test_function_1 [%ty : char list \rightarrow char list] "sort".`

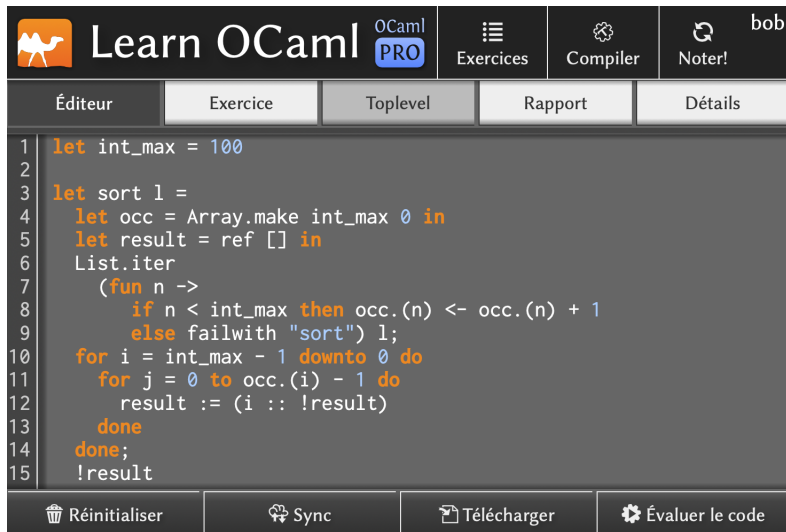
Ainsi on obtient des types monomorphes [...] tout en testant que la fonction est effectivement polymorphe. »

« écrire la fonction sort (5 points) »

▷ le code de test (**test.ml**)

```
1 let make_report () =
2   let report_int_list =
3     Test_lib.test_function_1_against_solution
4       [%ty: int list -> int list] "sort"
5       ~sampler:Test_lib.(sample_list ~sorted:false sample_int)
6       ~gen:3
7       []
8
9   and report_char_list =
10    Test_lib.test_function_1_against_solution
11      [%ty: char list -> char list] "sort"
12      ~sampler:Test_lib.(sample_list ~sorted:false sample_char)
13      ~gen:2
14      []
15  in
16  let report = List.append report_int_list
17    report_char_list in
18  [ (Report.section "Function sort" report) ] ;;
19
20 let report = ast_sanity_check code_ast make_report in
21 Test_lib.(set_result report)
```

Scénario 1 : un tri d'entiers en $O(n)$

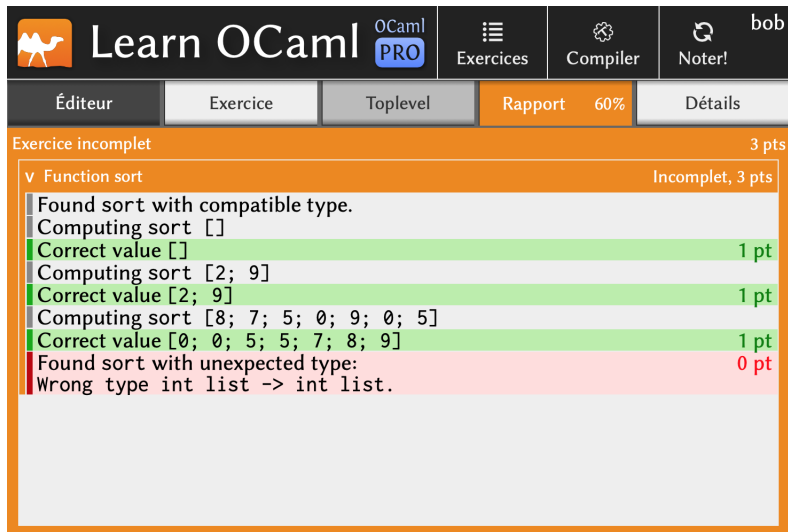


The screenshot shows the Learn OCaml web interface. At the top, there is a navigation bar with the Learn OCaml logo, the text "Learn OCaml", and a "PRO" badge. To the right are buttons for "Exercices", "Compiler", and "Noter!". The user's name "bob" is displayed in the top right corner. Below the navigation bar is a secondary menu with buttons for "Éditeur", "Exercice", "Toplevel", "Rapport", and "Détails". The main area contains a code editor with the following OCaml code:

```
1 let int_max = 100
2
3 let sort l =
4   let occ = Array.make int_max 0 in
5   let result = ref [] in
6   List.iter
7     (fun n ->
8       if n < int_max then occ.(n) <- occ.(n) + 1
9       else failwith "sort") l;
10  for i = int_max - 1 downto 0 do
11    for j = 0 to occ.(i) - 1 do
12      result := (i :: !result)
13    done
14  done;
15  !result
```

At the bottom of the interface, there is a footer bar with buttons for "Réinitialiser", "Sync", "Télécharger", and "Évaluer le code".

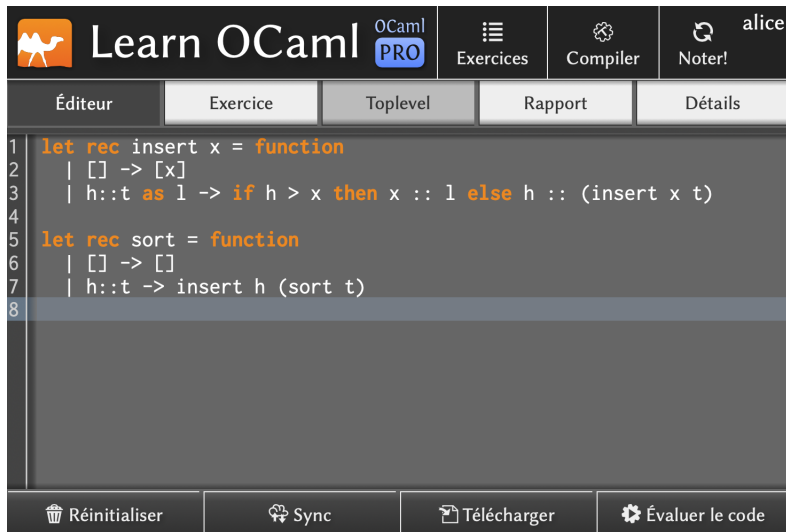
Scénario 1 : un tri d'entiers en $O(n)$



The screenshot shows the Learn OCaml interface. At the top, there is a navigation bar with the OCaml logo, the text "Learn OCaml", and a "PRO" badge. To the right are buttons for "Exercices", "Compiler", and "Noter!". Below this is a secondary navigation bar with buttons for "Éditeur", "Exercice", "Toplevel", "Rapport 60%", and "Détails". The main content area is titled "Exercice incomplet" and shows a test report for a function named "sort". The report lists several test cases with their results and scores.

Test Case	Result	Score
Found sort with compatible type.	Success	
Computing sort []	Success	
Correct value []	Success	1 pt
Computing sort [2; 9]	Success	
Correct value [2; 9]	Success	1 pt
Computing sort [8; 7; 5; 0; 9; 0; 5]	Success	
Correct value [0; 0; 5; 5; 7; 8; 9]	Success	1 pt
Found sort with unexpected type: Wrong type int list -> int list.	Failure	0 pt

Scénario 2 : un tri par insertion




The screenshot shows the Learn OCaml web interface. At the top, there is a navigation bar with the Learn OCaml logo, the text "Learn OCaml", and a "PRO" badge. To the right are buttons for "Exercices", "Compiler", and "Noter!". The user's name "alice" is displayed in the top right corner. Below the navigation bar is a secondary menu with buttons for "Éditeur", "Exercice", "Toplevel", "Rapport", and "Détails". The main area is a code editor with the following OCaml code:

```
1 let rec insert x = function
2   | [] -> [x]
3   | h::t as l -> if h > x then x :: l else h :: (insert x t)
4
5 let rec sort = function
6   | [] -> []
7   | h::t -> insert h (sort t)
8
```

At the bottom of the interface, there is a footer bar with four buttons: "Réinitialiser", "Sync", "Télécharger", and "Évaluer le code".

Scénario 2 : un tri par insertion

 **Learn OCaml** OCaml PRO Exercices Compiler alice Noter!


Éditeur Exercice Toplevel Rapport Détails

Exercice terminé 5 pts

v Function sort Terminé, 5 pts





Found sort with compatible type.	
Computing sort [0; 1; 8; 2; 8; 0]	
Correct value [0; 0; 1; 2; 8; 8]	1 pt
Computing sort [9; 9; 2; 4; 9; 5]	
Correct value [2; 4; 5; 9; 9; 9]	1 pt
Computing sort [4; 8; 5; 9; 8; 6]	
Correct value [4; 5; 6; 8; 8; 9]	1 pt
Found sort with compatible type.	
Computing sort ['a'; 'i'; 'x'; 'v'; 'u'; 'y']	
Correct value ['a'; 'i'; 'u'; 'v'; 'x'; 'y']	1 pt
Computing sort ['t'; 'd']	
Correct value ['d'; 't']	1 pt

Scénario 3 : un tri fusion

 **Learn OCaml** OCaml PRO Exercices Compiler lucky luke Noter!

Éditeur	Exercice	Toplevel	Rapport	Détails
---------	----------	----------	---------	---------

```
1 let split l =
2   let rec aux a1 a2 = function
3     | [] -> (List.rev a1,List.rev a2)
4     | h :: t -> aux a2 (h::a1) t in
5   aux [] [] l
6
7 let rec merge l1 l2 = match (l1,l2) with
8   | [],_ -> l2
9   | _,[] -> l1
10  | t1::q1,t2::q2 -> if t1 > t2 then t1::(merge q1 l2)
11                    else t2::(merge l1 q2);;
12
13 let rec sort = function
14   [] -> []
15   | [_] as s -> s
16   | l -> let l1,l2 = split l in
17         merge (sort l1) (sort l2)
```

 Réinitialiser	 Sync	 Télécharger	 Évaluer le code
---	--	---	---

Scénario 3 : un tri fusion

Learn OCaml OCaml PRO Exercices Compiler lucky luke Noter!

Éditeur Exercice Toplevel **Rapport** Détails

Exercice échoué 0 pt





▼ Function sort Échoué

Found sort with compatible type.	
Computing sort [1; 6; 1; 7; 8]	
Wrong value [8; 7; 6; 1; 1]	0 pt
Computing sort [7; 5; 6; 7; 6; 5; 6]	
Wrong value [7; 7; 6; 6; 6; 5; 5]	0 pt
Computing sort [3; 7; 2; 9; 7; 8; 7; 7]	
Wrong value [9; 8; 7; 7; 7; 7; 3; 2]	0 pt
Found sort with compatible type.	
Computing sort ['s'; 'c'; 's'; 'd'; 'e'; 'b'; 'b'; 'd']	
Wrong value ['s'; 's'; 'e'; 'd'; 'd'; 'c'; 'b'; 'b']	0 pt
Computing sort ['v'; 'g'; 'n'; 't'; 'n'; 'p'; 'w'; 'o'; 'w']	
Wrong value ['w'; 'w'; 'v'; 't'; 'p'; 'o'; 'n'; 'n'; 'g']	0 pt

► Question : Où est l'erreur de Lucky Luke ?

Scénario 3 : un tri fusion

► Lucky Luke relance la correction :

 **Learn OCaml** OCaml PRO  Exercices  Compiler  lucky luke
Noter!

Éditeur	Exercice	Toplevel	Rapport 60%	Détails
---------	----------	----------	-------------	---------

Exercice incomplet 3 pts

v Function sort Incomplet, 3 pts

```
Found sort with compatible type.  
Computing sort []  
Correct value [] 1 pt  
Computing sort [2]  
Correct value [2] 1 pt  
Computing sort [9; 8; 0; 5; 0; 7; 0; 7]  
Wrong value [9; 8; 7; 7; 5; 0; 0] 0 pt  
Found sort with compatible type.  
Computing sort []  
Correct value [] 1 pt  
Computing sort ['l'; 'h'; 'v'; 'g'; 'x'; 'e'; 'p']  
Wrong value ['x'; 'v'; 'p'; 'l'; 'h'; 'g'; 'e'] 0 pt
```

Partie III : Des améliorations

- ▶ séparation et réutilisabilité du code de test
- ▶ gestion du polymorphisme
- ▶ mise en place d'un nouveau schémas de correction

Séparation du code de test : *Peano revisité*

▷ **descr.md**

```
# Peano
Définir la fonction
of_int : int → peano
```

▷ **prelude.ml**

```
type peano = Z | S of peano
```

▷ **prepare.ml**

```
exception Undefined
let ⊥ _ = raise Undefined
```

▷ **template.ml**

```
let of_int = ⊥
```

▷ **solution.ml**

```
let rec of_int = function
  | 0 → Z
  | n → S (of_int (n-1))
```

▷ **test.ml**

```
let make_report () =
  Make_report.test_arity1
  "of_int"
  [%ty : int -> peano]

Set_result.set make_report
```


Séparation du code de test : *Peano revisité*

▷ **descr.md**

```
# Peano
Définir la fonction
of_int : int → peano
```

▷ **prelude.ml**

```
type peano = Z | S of peano
```

▷ **prepare.ml**

```
exception Undefined
let ⊥ _ = raise Undefined
```

▷ **template.ml**

```
let of_int = ⊥
```

▷ **solution.ml**

```
let rec of_int = function
  | 0 → Z
  | n → S (of_int (n-1))
```

▷ **depend.txt** [optionnel]

```
../../../../libs/set_result.mli,
../../../../libs/set_result.ml,
../../../../libs/make_report.ml
```

▷ **test.ml**

```
let make_report () =
  Make_report.test_arity1
  "of_int"
  [%ty : int -> peano]

Set_result.set make_report
```

Séparation du code de test

Effet sur l'environnement de correction

```
<prelude.ml> ;;
<prepare.ml> ;;

module Code = struct <le code de l'étudiant> end ;;

module Solution = struct <solution.ml> end ;;

module Introspection = ... ;;
let code_ast = ... ;;

let results = ref None ;;
let timeout = ... ;;

module Test_lib = ... ;;
module Report = ... ;;

module Set_result : sig <set_result.mli> end = struct <set_result.ml> end
module Make_report = struct <make_report.ml> end ;;
(* Make_report peut dépendre de Set_result *)

<test.ml> ;;
```

Fonctions polymorphes : sur la piste des types abstraits

- ▷ fichiers séparés `set_results.ml`
et `variables.ml` réutilisables

```
1 let set make_report =  
2   let report =  
3     Test_lib.ast_sanity_check  
4       code_ast  
5       make_report in  
6   Test_lib.set_result report
```

```
1 module type PARAM = sig  
2   type t  
3   val sample : unit → t  
4 end  
5  
6 module Int = struct  
7   type t = int  
8   let sample = Random.bits  
9 end
```

- ▷ le code de test (`test.ml`)

```
1 module A = (Variables.Int : Variables.PARAM)  
2  
3 let make_report () =  
4   [ (Report.section "Function sort"  
5     Test_lib.test_function_1_against_solution  
6       [%ty: A.t list -> A.t list] "sort"  
7       ~sampler:(Test_lib.sample_list ~sorted:false A.sample)  
8       ~gen:5  
9       [] ) ]  
10  
11 let () = Set_result.set make_report
```

J'ouvre une parenthèse : l'introspection en Learn-OCaml

L'accès aux valeurs

```
module I = Introspection
```

```
match I.get_value "Code.v" [%ty :  $\tau$ ] with
```

```
| Absent →  $e_1$ 
```

```
| Incompatible message →  $e_2$ 
```

```
| Present v →  $e_3$ 
```

Remarque : τ ne peut pas contenir de variable de types.

J'ouvre une parenthèse : l'introspection en Learn-OCaml

L'accès aux modules

- ▷ par une contrainte de signature
- ▷ fournit un module empaqueté dans une valeur

```
match I.get_value "Code.M" [%ty : (module S)] with  
| Absent → e1  
| Incompatible message → e2  
| Present m → let module M = (val m : S) in e3
```

Remarque : S peut contenir des variables de types.

Je ferme la parenthèse.

La bonne manière de traiter les fonctions polymorphes

- ▷ définir une signature **S = sig val f : τ end**
- ▷ accéder au module **Code** (le code de l'étudiant)
 - ▶ par la contrainte de signature **S**
 - ▶ fournit un module empaqueté dans une valeur
- ▷ désempacter la valeur pour obtenir un module **Code_safe**
- ▷ accéder à la valeur f avec la notation qualifiée **Code_safe.f**
- ▷ tester une instance de **Code_safe.f**

La bonne manière de traiter les fonctions polymorphes

- ▷ Ne pas récupérer la valeur `sort` qui nous intéresse, mais le module `Code` qui l'englobe!

```
1 module type S = sig  
2   val sort :  $\alpha$  list  $\rightarrow$   $\alpha$  list  
3 end  
4  
5 match I.get_value "Code" [%ty : (module S)] with  
6 | Absent  $\rightarrow$  e1  
7 | Incompatible message  $\rightarrow$  e2  
8 | Present code  $\rightarrow$  let module Code = (val code : S) in  
9   e3
```

Guider l'étudiant vers la solution

d'après François Pottier

```
test_arity1 "f" [%ty:  $\tau$ ] Solution.f [x1; x2; ...; xn]
```

- ▷ chercher une variable `Code.f` liée à une valeur compatible avec τ
- ▷ vérifier pour chaque sample x_i de $[x_1; x_2; \dots; x_n]$:

```
(let vi = Code.f xi  
  and wi = Solution.f xi in equal vi wi)
```

- ▷ trouver un contreexemple (x_i, v_i, w_i) et produire le rapport :

```
Found f with compatible type.  
f is incorrect.  
The following expression:  
f xi  
produces the following result:  
vi  
This is invalid. Producing the following result is valid:  
wi
```


Je propose un mécanisme de correction plus général

- ▷ tester des expressions fonctionnelles :

```
test_arity1  ~test:    (fun x → e1)
              ~expect: (fun x → e2)
              ~samples: [x1; x2; ...; xn]
```

- ▷ vérifier pour chaque sample x_i :

```
(let vi = test xi
 and wi = expect xi in equal vi wi)
```

- ▷ trouver un contreexemple (x_i, v_i, w_i) et produire le rapport :

The following expression:

```
(fun x → e1) xi
```

produces the following result:

v_i

This is invalid. Producing the following result is valid:

w_i

Tester des expressions vis à vis d'autres expressions :

- ▷ permet d'exprimer des propriétés que le code doit vérifier.
 - ▶ montre clairement à l'étudiant ce qu'il doit corriger, par exemple

```
(fun n → odd (pred n)) ~ even
```

- ▷ permet de corriger des fonctions manipulant des types abstraits

```
module Code = sig
  type  $\alpha$  t
  val map : ( $\alpha \rightarrow \beta$ ) →  $\alpha$  t →  $\beta$  t
end
```

- ▶ `Solution.map` est incompatible avec `Code.map` !
- ▶ Mais nul besoin de `Solution.map` :

```
(fun  $l \rightarrow$  map identity  $l$ ) ~ (fun  $l \rightarrow l$ )
(fun  $l \rightarrow$  map  $f^{-1}$  (map  $f$   $l$ )) ~ (fun  $l \rightarrow l$ )
```

Préconditions/postconditions

- ▷ Idée : abstraire l'application

```
test_arity1  ~test:    (fun x → e1)
              ~expect: (fun x → e2)
              ~apply:  (fun f x → f x)
              ~samples: [x1; x2; ...; xn]
```

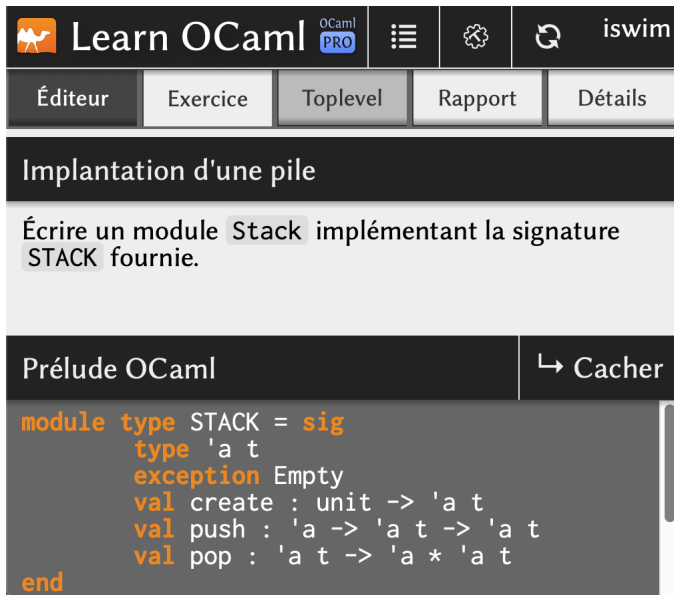
calcule pour chaque x_i :




```
(let vi = apply test xi
 and wi = expect xi in equal vi wi)
```

- ▷ Un exemple : les descripteurs de fichiers

```
let apply_ic f ic =
  let p = pos_in ic in
  let r = f ic in
  if pos_in ic <> p then (seek_in ic p ; e)
  else report r
```

Exemple synthétique



Learn OCaml OCaml PRO    iswim

Éditeur Exercice Toplevel Rapport Détails

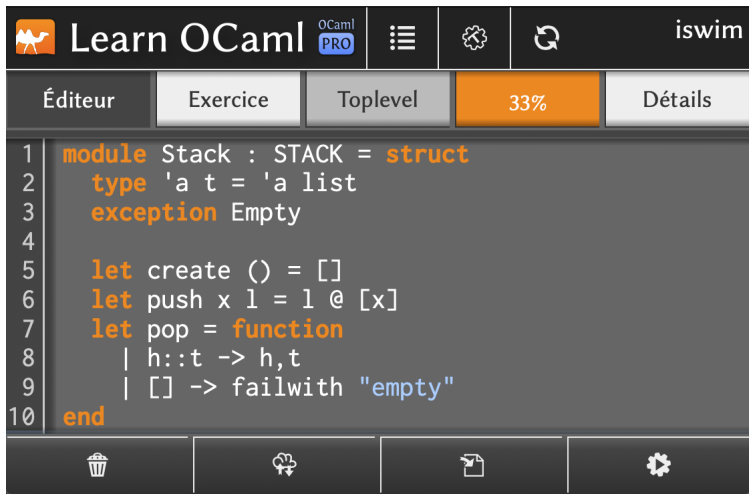
Implantation d'une pile

Écrire un module `Stack` implémentant la signature `STACK` fournie.

Prélude OCaml ↳ Cacher

```
module type STACK = sig
  type 'a t
  exception Empty
  val create : unit -> 'a t
  val push : 'a -> 'a t -> 'a t
  val pop : 'a t -> 'a * 'a t
end
```

Une copie d'étudiant



The screenshot shows the Learn OCaml IDE interface. The top bar includes the Learn OCaml logo, the OCaml PRO version indicator, a menu icon, a settings gear, a refresh icon, and the user name 'iswim'. Below the top bar is a navigation bar with tabs for 'Éditeur', 'Exercice', 'Toplevel', '33%', and 'Détails'. The main area is a code editor displaying OCaml code for a stack implementation. The code is as follows:

```
1 module Stack : STACK = struct
2   type 'a t = 'a list
3   exception Empty
4
5   let create () = []
6   let push x l = l @ [x]
7   let pop = function
8     | h::t -> h,t
9     | [] -> failwith "empty"
10 end
```

The bottom bar contains icons for a trash can, a refresh icon, a document icon, and a settings gear.

Le rapport de correction (0)

The screenshot shows the 'Learn OCaml' interface. At the top, there is a navigation bar with the OCaml logo, 'Learn OCaml PRO', a menu icon, a settings gear, a refresh icon, and the username 'iswim'. Below this is a secondary bar with tabs for 'Éditeur', 'Exercice', 'Toplevel', '33%', and 'Détails'. The main content area has an orange header that says 'Exercice incomplet' on the left and '1 pts' on the right. Below this is a red-bordered box containing the exercise details. The box has a red header with 'v test 0' on the left and 'Échoué' on the right. The main text inside the box reads: 'The following expression: 0 pt' followed by a code block:

```
(fun () ->
  Stack.(
    let stk = create () in
    pop stk
  )) ()
```

 Below the code, it says: '... raises the following exception: (Failure empty)'. The final line states: 'This is incorrect. Raising the following exception is correct: Code.Stack.Empty'.

Learn OCaml OCaml PRO [Menu] [Settings] [Refresh] iswim

Éditeur Exercice Toplevel 33% Détails

Exercice incomplet 1 pts

v test 0 **Échoué**

The following expression: **0 pt**

```
(fun () ->
  Stack.(
    let stk = create () in
    pop stk
  )) ()
```

... raises the following exception:
(Failure empty)

This is incorrect. Raising the following exception is correct:
Code.Stack.Empty

Le rapport de correction (1)

The screenshot shows the 'Learn OCaml' web application interface. At the top, there is a navigation bar with the OCaml logo, the text 'Learn OCaml', and 'OCaml PRO'. To the right are icons for a menu, settings, and refresh, along with the username 'iswim'. Below this is a secondary navigation bar with buttons for 'Éditeur', 'Exercice', 'Toplevel', '33%', and 'Détails'. The main content area has an orange header bar that says 'Exercice incomplet' on the left and '1 pts' on the right. Below this is a green box containing the test result. The box is titled 'v test 1' on the left and 'Terminé, 1 pts' on the right. The text inside the box reads: 'The following expression: 1 pt' followed by a code block:

```
(fun x ->
  Stack.(
    let stk = create () in
    pop (push x stk)
  ))
```

 and ends with 'seems correct.'

Le rapport de correction (2)

The screenshot shows the 'Learn OCaml' interface. At the top, there is a navigation bar with the OCaml logo, the text 'Learn OCaml', a 'PRO' badge, a menu icon, a settings gear, a refresh icon, and the username 'iswim'. Below this is a secondary bar with tabs for 'Éditeur', 'Exercice', 'Toplevel', '33%', and 'Détails'. The main content area is titled 'Exercice incomplet' and shows '1 pts' earned. A red header bar indicates 'v test 2' and 'Échoué'. The main text area contains the following information:

The following expression: 0 pt

```
(fun x1 x2 ->  
  Stack.(  
    let stk = create () in  
    pop (push x2 (push x1 stk))  
  )) 176 57
```

... produces the following value:
(176, <abstr>)

This is incorrect. Producing the following value is correct:
(57, <abstr>)

Le programme de test ... d'abord, des dépendances !

```
../../../../libs/printer.mli,  
../../../../libs/sampler.mli,  
../../../../libs/check_fun/protect.mli,  
../../../../libs/check_fun/fail_report.mli,  
../../../../libs/check_fun/lookup_value.mli,  
../../../../libs/check_fun/check_fun.mli,  
../../../../libs/check_fun/set_result.mli,  
  
../../../../libs/printer.ml,  
../../../../libs/sampler.ml,  
../../../../libs/check_fun/protect.ml,  
../../../../libs/check_fun/fail_report.ml,  
../../../../libs/check_fun/lookup_value.ml,  
../../../../libs/check_fun/check_fun.ml,  
../../../../libs/check_fun/set_result.ml
```

- ▷ **L'algorithmique sous-jacente est structurée en modules séparés réutilisables**

Le programme de test (*test.ml*)

```
1 let test_stack () =
2   let stk = Lookup_value.get "Code.Stack" [%ty: (module STACK)]
3   in
4   let module Stack = (val stk : STACK) in
5
6   let test0 () =
7     Check_fun.arity1
8     ~source: "(fun () ->
9               \ Stack.(
10                \   let stk = create () in \n\
11                \   pop stk                 \n\
12                \ ))"
13     ~test: (fun () → Stack.(pop (create ())))
14     ~expect: (fun () → raise Stack.Empty)
15     ~samples: [();()]
16     ~view: Printer.(unit, (pair int abstract))
17   () in
```

- ▷ la chaîne aux lignes 7-11 n'est pas interprétée : c'est un simple affichage.
 - ▶ L'affichage des valeurs fonctionnelles est une question à développer.

Le programme de test (*test.ml*)

```
17 let test1 () =
18   Check_fun.arity1
19     ~source: "(fun x ->                               \n\  
20               \   Stack.(                               \n\  
21                 \   let stk = create () in \n\  
22                 \   pop (push x stk)         \n\  
23                 \   ))"  
24   ~test:   (fun x → Stack.(pop (push x (create ())))))  
25   ~expect: (fun x → x,Stack.create ())  
26   ~seed:   Sampler.int  
27   ~view:   Printer.(int,abstract)  
28   ()
```

Le programme de test (*test.ml*)

```
29 let test2 () =
30   Check_fun.arity2
31     ~source: "(fun x1 x2 ->
32               \      Stack.(
33                 \      let stk = create () in
34                 \      pop (push x2 (push x1 stk))
35                 \    )"
36     ~test: (fun x1 x2 → let stk = create () in
37              Stack.(pop (push x2 (push x1 stk))))
38     ~expect: (fun x1 x2 → (x2, Stack.(push x1 (create ())))))
39     ~seed: Sampler.(int, int)
40     ~view: Printer.(int, int, pair int abstract)
41   ()
42
43 let make_report () =
44   [test0; test1; test2]
45
46 let () =
47   Set_result.set make_report
```

Conclusion

Learn-OCaml

- ▷ Facile à utiliser, modulo des améliorations

Améliorations proposées

- ▷ séparation du code de test, pour faciliter le travail de l'enseignant.
- ▷ nouvelles techniques de correction, avec un double objectif :
 - ▶ apporter à l'étudiant une correction interactive et de qualité ;
 - ▶ permettre à l'enseignant de porter des contenus pédagogiques avancés en OCaml.
→ support pour la correction de programmes modulaires avec types abstraits (projets de L3 / M1)

Quelques pistes

- ▷ support pour l'interopérabilité avec **JavaScript**
- ▷ écriture de tests par l'étudiant (cf. **Mr Scheme** et **Mr Python**)



Benjamin Canou, Roberto Di Cosmo, Grégoire Henry.

Sous le capot du MOOC OCaml.

Journées Francophones des Langages Applicatifs (JFLA 2016), Jan 2016, Saint-Malo, France. hal-01333599



Benjamin Canou, Grégoire Henry, Çağdas Bozman, Fabrice Le Fessant.

Learn OCaml, An Online Learning Center for OCaml.

OCaml Users and Developers Workshop 2016, Sep 2016, Nara, Japan. hal-01352015



Benjamin Canou, Çağdas Bozman, Grégoire Henry.

Scaling up functional programming education : under the hood of the OCaml MOOC.

Proc. ACM Program. Lang. 1, ICFP, Article 4 (September 2017)



Bozman, Canou, Di Cosmo, Couderc, Gesbert, Henry, Le Fessant, Mauny, Morel, Peyrot, Régis-Gianas.

Learn-OCaml : un assistant à l'enseignement d'OCaml

Journées Francophones des Langages Applicatifs (JFLA), Jan 2019, Les Rousses, France. hal-01962838