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

Çagdas Bozman[2], Benjamin Canou[1], Roberto Di Cosmo[3,5,6,7], Pierrick Couderc[2], Louis Gesbert[2], Grégoire Henry[1], Fabrice Le Fessant[2], Michel Mauny[3,4], Carine Morel[4,5], Loïc Peyrot[4,5], et Yann Régis-Gianas[3,4,5,6]

[1] Nomadic Development [2] OCamlPro [3] Inria [4] Fondation OCaml
[5] Université Paris Diderot [6] IRIF [7] Software Heritage [8] ENSTA

La plateforme Learn-OCaml est un assistant à l'enseignement du langage de programmation OCaml, développée dans le cadre du projet éponyme porté par la Fondation OCaml, qui vise à soutenir et développer l'usage d'OCaml dans l'enseignement. La plateforme permet d'écrire des exercices munis de correcteurs automatiques qui peuvent tester non seulement la correction fonctionnelle des programmes soumis par les étudiants mais aussi la façon dont ces programmes sont écrits ou calculent.

Dans cet article, nous présentons la plateforme dans son ensemble, le projet dans lequel s'inscrit son développement, et ses fonctionnalités principales, à travers notamment l'écriture d'un exercice et de son correcteur ainsi qu'un premier retour sur deux expériences d'usage menées à l'Université McGill et à l'Université Paris Diderot.

1 Introduction

La Fondation OCaml, créée en juin 2018, a pour mission de promouvoir l'usage d'OCaml, de soutenir son développement et celui de son écosystème. La Fondation prend la suite du Consortium Caml, créé à l'Inria en 2001, et qui a réuni depuis cette date une quinzaine d'utilisateurs industriels du langage. Abritée par la Fondation Partenariale Inria, la Fondation OCaml présente un certain nombre d'avantages par rapport au Consortium : elle dispose en effet des facilités de gestion fournies par sa fondation abritante – une organisation de droit privé – ainsi que la fiscalité avantageuse qui est offerte à ses mécènes. De plus, en tant que fondation, la Fondation OCaml a pu se doter de facultés «redistributives» qui lui permettent d'apporter son soutien à des actions conformes à ses missions qui peuvent être opérées par d'autres (organisations ou personnes physiques) que par la Fondation elle-même.

Le projet Learn-OCaml est l'une des premières actions de la Fondation OCaml : il vise à promouvoir l'usage d'OCaml dans l'enseignements et à faciliter aux étudiants, formateurs et ingénieurs, l'accès à des ressources pédagogiques de qualité et librement accessibles. Le développement de la plateforme logicielle Learn-OCaml est l'un des points importants de ce projet.

Cet article présente la plateforme logicielle Learn-OCaml développée par la fondation OCaml et OCamlPro. Elle s'appuie sur une suite logicielle développée par OCamlPro depuis plusieurs années et sur une plateforme d'exercices de programmation en ligne développée avec le soutien de l'IRILL et de SAPIENS, destinée à être au cœur du MOOC "Introduction to functional programming in OCaml". Ses spécificités ont déjà été présentées dans un article publié à ICFP'17[4]. Cet article reprend bien sûr une partie des éléments présentés à ICFP'17, mais illustre aussi certaines des fonctionnalités introduites par la nouvelle version de la plateforme, typiquement la possibilité de la déployer en dehors de l'infrastructure d'un MOOC et son adaptation au contexte d'un enseignement plus classique.

La publication de cet article aux JFLA 2019 est aussi un moyen d'inviter la communauté enseignante francophone à enrichir cette plateforme par l'ajout de nouveaux traits ou par la création de contenu pédagogique, et à l'utiliser dans des enseignement nouveaux ou existants.

2 Vue d'ensemble de la plateforme Learn-OCaml

Fonctionnalités principales Learn-OCaml assiste l'enseignement du langage de programmation OCaml en (i) donnant accès à des exercices de programmation corrigés automatiquement ; (ii) assurant le suivi de la progression des élèves ; (iii) permettant à l'enseignant de personnaliser le chemin d'apprentissage des élèves.

La plateforme Learn-OCaml suit une architecture client-serveur classique : son usage typique consiste donc à déployer une instance du serveur sur une machine accessible sur le réseau *via* le protocole HTTP/S et à interagir avec ce serveur à l'aide d'un client. Les clients actuels, au nombre de deux, consistent en une application web et une interface en ligne de commande. L'enseignant et l'élève utilisent le même client mais l'enseignant a accès à plus de fonctionnalités.

Serveur Le serveur prend en charge le stockage d'un ensemble d'exercices munis de correcteurs automatiques. Il stocke aussi l'historique des réponses et des résultats des élèves. Enfin, il maintient un ensemble de méta-données sur les élèves (typiquement un ensemble d'étiquettes qui permettent de structurer une cohorte en groupes de travaux dirigés, de niveaux, ...) ainsi qu'un ensemble de méta-données sur les exercices (les compétences travaillées et requises, les exercices de remédiation, des statistiques de réussites, ...).

Une des méta-données essentielles du système est une liste d'exercices associée à chaque élève. Un nouvel exercice apparaît dans cette liste s'il est ouvert à l'élève. Cette ouverture peut être illimitée dans le temps ou prendre la forme d'un "devoir" caractérisé par une date d'ouverture et une date de rendu. À tout moment, l'enseignant peut mettre à jour cette liste pour un élève en particulier, pour un groupe d'élèves ou encore pour l'ensemble des élèves.

Comme dans le cas de l'environnement d'exercices déployé pour le MOOC OCaml[4], la correction automatique des réponses de l'élève est faite par le client [1], directement sur la machine de l'élève, qui transmet ensuite le rapport de correction au serveur pour stockage.

Cette architecture réduit considérablement la charge sur le serveur, qui ne doit gérer que l'envoi et la collecte des données. Elle raccourcit aussi le temps qui prend un cycle édition-correction, qui ne dépend que de l'algorithme de correction automatique et des ressources de la machine de l'étudiant ; le nombre d'étudiants travaillant simultanément sur la plateforme n'intervient pas. L'élève travaille ainsi plus efficacement, sans être tributaire d'une éventuelle surcharge du serveur ou de problèmes de connectivité.

Dans la plateforme Learn-OCaml, une fonctionnalité a été ajoutée pour permettre à l'enseignant de déclencher une correction des réponses des élèves côté serveur. Cela permet à l'enseignant de revalider les résultats des élèves sur le serveur et d'améliorer la fiabilité des évaluations, en rendant vaine toute tentative de modifier le client pour qu'il envoie des rapports de correction forgés pour associer la note maximale à des réponses invalides.

Pour finir, le serveur peut être déployé localement sur la machine de l'enseignant qui peut ainsi tester ses correcteurs sans avoir à les mettre en production. Ce déploiement s'appuie sur Docker[5] et peut donc s'effectuer sous GNU/Linux, MacOS X et MS/Windows. Sous les systèmes UNIX, pour déployer une instance locale de Learn-OCaml, il suffit ainsi d'exécuter :
`docker run --rm -v \`pwd\`:/repository:ro -p 80:8080 ocamlsf/learn-ocaml:dev`
depuis un répertoire contenant un dépôt d'exercices [2]. Un serveur est alors disponible à l'adresse `http://localhost:80`.

1. Historiquement, cette fonctionnalité est héritée du projet TryOCaml, développé par OCamlPro.
2. Nous allons voir comment créer un tel dépôt dans la section 3.

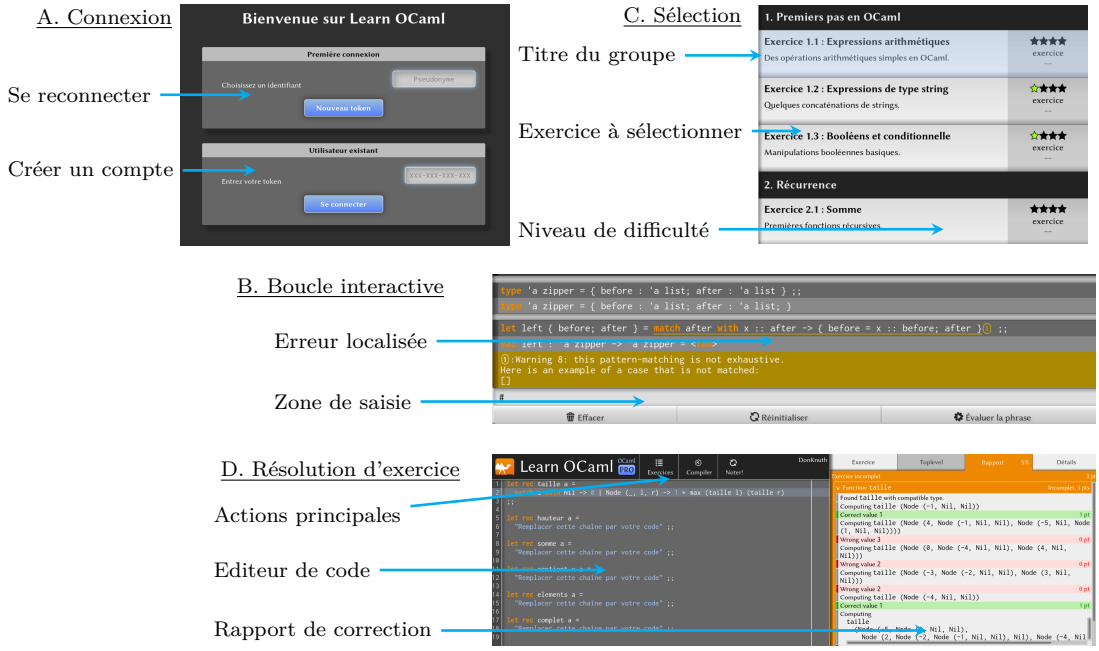

Figure 1 – Captures d'écran du client web de Learn-OCaml.

Client web Si on se connecte avec un navigateur web à l'URL racine d'un serveur Learn-OCaml, on accède au client web de la plateforme. Nous allons maintenant commenter rapidement les captures d'écran des figures 1 et 2. Le client permet ainsi de créer un compte sur la plateforme (écran A). À ce compte est associé un identifiant et un jeton unique. Ce jeton sert à s'authentifier lors des sessions futures. Une fois connecté à la plateforme, deux boutons intitulés "Exercice" et "Toplevel" permettent de choisir une activité [3]. L'écran B correspond à l'activité "Toplevel" : il s'agit d'un boucle interactive OCaml qui tourne dans le navigateur et permet à l'élève d'effectuer des tests. L'avantage de cette boucle interactive nichée dans le navigateur par rapport à la boucle d'interaction classique est qu'elle permet de mieux localiser et mettre en page les messages d'erreur : on profite en effet de l'affichage HTML pour structurer l'affichage et le rendre plus clair qu'en mode textuel. L'écran C correspond à l'activité "Exercice" mentionnée plus haut. Les exercices sont en général regroupés thématiquement et chaque exercice est décrit par un titre, une description courte et un niveau de difficulté quantifié par un nombre d'étoiles (plus un exercice a d'étoiles et plus il est jugé difficile). Enfin, l'écran D offert à l'élève correspond à l'activité de résolution d'exercice à proprement parler.

À gauche de l'écran, on distingue une zone d'édition de code source surmontée de plusieurs boutons. La zone d'édition est une version modifiée par OCamlPro de l'éditeur de texte ACE[1]. Elle prend en charge la coloration syntaxique mais aussi une forme d'indentation automatique (implémentée grâce à ocp-indent). Le bouton "Compiler" permet de s'assurer que le programme est bien formé. Le bouton "Noter" exécute la correction automatique. Cette dernière produit un rapport de correction (que l'on distingue à droite de l'écran). Chaque test rapporte un point s'il réussit. D'autres onglets sont accessibles à l'élève : l'onglet "Exercice" affiche

3. Une activité de consultation de tutoriels est également disponible mais nous ne la documentons pas ici.

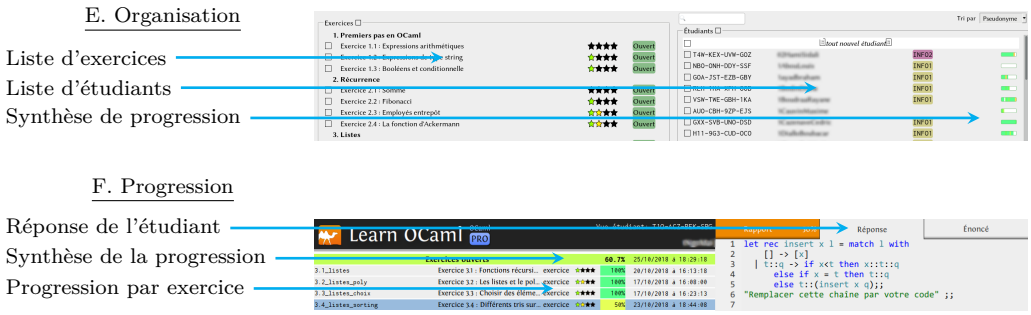

Figure 2 – Captures d'écran des activités de l'enseignant.

l'énoncé de l'exercice, l'onglet "Toplevel" permet d'exécuter une boucle interactive initialisée par le contexte de travail de l'exercice, l'onglet "Détails" permet de visualiser les méta-données de l'exercice en cours de résolution. Ces méta-données incluent par exemple les compétences travaillées ou requises par l'exercice, son identifiant ou encore le nom de ses auteurs.

La figure 2 donne une idée des activités supplémentaires accessibles aux enseignants. L'écran E correspond à une activité de visualisation de l'ensemble des exercices et des élèves. On peut ainsi constater l'avancement de chaque élève ou d'un groupe d'élèves grâce à des barres de progression. C'est grâce à cette interface que l'enseignant modifie le statut d'un exercice : un exercice peut être fermé, ouvert à tous ou alors affecté à un groupe d'élèves (avec éventuellement des contraintes de temps pour sa réalisation). Cette interface permet aussi d'affecter des étiquettes aux élèves pour créer des groupes (de classes, de niveaux, *etc.*). L'écran F donne une vue plus précise de la progression d'un élève : ce tableau de bord affiche le score de l'étudiant pour chaque exercice et pour chaque compétence travaillée. Ces données sont téléchargeables au format CSV pour que l'enseignant puisse leur appliquer un traitement spécifique à ses besoins.

Interface en ligne de commande Le client web est utile pour des élèves qui débutent en programmation : il n'y a aucun coût d'installation et l'interface s'est révélée suffisamment intuitive d'utilisation pour que les élèves se mettent directement à travailler sur les exercices de programmation. Cependant, cette interface s'avère inadaptée pour des élèves ayant déjà une expérience significative en développement logiciel. En effet, dans ce cas, il est fort probable que l'élève ait déjà l'habitude d'un usage sophistiqué de son éditeur de texte favori et considère le passage à l'éditeur en ligne comme une régression [4].

Pour cette raison, la plateforme fournit aussi un outil en ligne de commande nommé `learn-ocaml-client` qui permet d'utiliser le sous-système de correction automatique depuis son environnement de développement favori. Il suffit ainsi d'éditer sa réponse dans un fichier source et d'invoquer `learn-ocaml-client` en lui passant l'identifiant de l'exercice et le chemin vers le fichier source utilisé pour que Learn-OCaml déclenche sa correction automatique (locale) et transmette la réponse et le rapport au serveur. Ce client s'appuie sur une API HTTP qui est exposée par le serveur. L'existence de cette API rend envisageable le développement d'autres clients dans le futur.

4. Typiquement, le mécanisme d'indentation automatique est utile aux élèves ne sachant pas mettre en forme leur code source... mais agace ceux qui possèdent déjà cette compétence.

descr.md

```
# Arbres binaires de recherche
Soient `thing` un type et `compare : thing -> thing -> int` l'implémentation d'un préordre total
sur les valeurs de type `thing`. Deux valeurs `a` et `b` de type `thing` sont dites équivalentes
ssi `compare a b = 0`, `a` est plus petit que `b` ssi `compare a b < 0`, `a` est plus grand que `b`
ssi `compare a b > 0`.

Un *arbre binaire de recherche* est un arbre binaire étiqueté par des valeurs de type `thing` tel que:
- `Leaf x` est un arbre binaire de recherche;
- `Node (l, x, r)` est un arbre binaire de recherche ssi
  - `l` et `r` sont des arbres binaires de recherche et
  - pour toute étiquette `y` de `l`, `y` est plus petit que `x` et
  - pour toute étiquette `y` de `r`, `y` est plus grand que `x`.

**Question** Écrire une fonction récursive `mem : thing -> thing bst -> bool` telle que `mem x t`
renvoie `true` ssi une valeur de type `thing` équivalente à `x` apparaît dans `t`.
Essayez de minimiser le nombre de comparaisons utilisées par `mem`.
```

meta.json

```
{ "learnocaml_version" : "2", "kind" : "exercise", "stars" : 2,
   "title" : "Binary Search Trees", "short_description"  : "How to search in a binary search tree.",
   "authors" : [ [ "Don Knuth", "<don.knuth@does-not-read-emails>" ] ],
   "focus":[ "pattern-matching", "complexity" ],
   "requirements" : [ "recursion", "parametric type", "algebraic datatype" ] }
```

template.ml

```
let mem x t = "À compléter"
```

solution.ml

```
let rec mem x = function
  | Leaf y ->
     compare x y = 0
  | Node (l, y, r) ->
     let cmp = compare y x in
     if cmp < 0 then mem x l
     else if cmp > 0 then mem x r
     else true
```

prelude.ml

```
type 'a bst =
  | Leaf of 'a
  | Node of 'a bst * 'a * 'a bst
```

prepare.ml

```
let comparison_counter = ref 0
let reset_comparison_counter () = comparison_counter := 0
let consumed_comparisons () = !comparison_counter
module Thing : sig
  type t
  val compare : t -> t -> int
  val sample : unit -> t
  val shake : t -> t
end = struct
  type t = Thing of int * int
  let sample () = Thing (Random.bits (), Random.bits ())
  let compare (Thing (x, _)) (Thing (y, _)) =
    incr comparison_counter;
    compare x y
  let shake (Thing (x, _)) = Thing (x, Random.bits ())
end
type thing = Thing.t
let sample_thing = Thing.sample
```

Figure 3 – Exercice : La recherche d'un élément dans un arbre binaire de recherche.

3 Les fonctionnalités de Learn-OCaml par l'exemple

Pour donner une idée des fonctionnalités de correction automatique de Learn-OCaml, nous allons commenter un exemple d'exercice autocorrigé. Les fichiers nécessaires à la définition de cet exercice se trouvent dans les figures 3 et 4. On commente ces fichiers tour à tour.

descr.md L'énoncé de l'exercice est spécifié par le fichier `descr.md`. Ce fichier au format Markdown peut contenir du texte mis en page, des formules mathématiques écrites en LaTeX ou encore des images. Learn-OCaml accepte aussi des énoncés écrits au format HTML. Notons qu'il suffit d'ajouter une extension `.fr`, `.en`, etc, à la fin du nom du fichier pour préciser sa langue. Plusieurs traductions d'un même énoncé peuvent ainsi coexister.

meta.json Les métadonnées d'un exercice sont fournies sous la forme d'un fichier JSON nommé `meta.json`.

prelude.ml Dans l'exercice qui nous intéresse ici, on demande à l'élève d'implémenter une fonction de recherche dans un arbre binaire de recherche. Pour écrire cette fonction, l'étudiant

test.ml

```
open Test_lib
open Report

let failure msg = [Message ([ Text msg ], Failure)]
let success msg = [Message ([ Text msg ], Success 1)]

let rec grade () =
  let expected_type = [%ty : thing -> thing bst -> bool ] in (
  check_no_while @@ fun () ->
  let before_user _ _ = reset_comparison_counter ()
  and sampler () =
    let t = sample_bst () in
    if Random.int 2 = 0 then (Thing.shake (pick t), t) else (sample_thing (), t)
  and after _ tree _ _ =
    if consumed_comparisons () > height tree then
      failure "Le nombre de comparaisons doit être inférieur à la hauteur de l'arbre!"
    else
      success "Bravo"
  and manual_tests =
    let x = sample_thing () in [ (x, Leaf x); (sample_thing (), Leaf x) ]
  in
  test_function_2_against_solution ~gen:10 ~before_user ~sampler ~after expected_type "mem" manual_tests
) |> set_result
and sample_bst () =
  if Random.int 2 = 0 then Leaf (sample_thing ()) else Node (sample_bst (), sample_thing (), sample_bst ())
and pick = function
  | Leaf x -> x
  | Node (l, y, r) -> match Random.int 3 with | 0 -> y | 1 -> pick l | 2 -> pick r | _ -> assert false
and height = function Leaf _ -> 1 | Node (l, x, r) -> 1 + max (height l) (height r)
and check_no_while cb =
  find_binding code_ast "mem" @@ fun expr ->
  let flag = ref false in
  let on_expression = Parsetree.(function { pexp_desc = Pexp_while _ } -> flag := true; [] | _ -> []) in
  ast_check_expr ~on_expression expr |> ignore;
  if !flag then failure "Utilisez la récursion! Pas la boucle while" else cb ()

let () = grade ()
```

Figure 4 – Exercice : La recherche d'un élément dans un arbre binaire de recherche (suite).

doit s'appuyer sur la définition du type `bst` donnée dans le fichier `prelude.ml`. Le contenu de ce fichier fait partie de l'énoncé de l'exercice et est évalué juste avant la réponse de l'étudiant.

template.ml Il est souvent utile de proposer une base de départ à l'étudiant pour qu'il se concentre sur les aspects importants de l'exercice ou pour qu'il construise sa réponse en suivant une architecture préétablie. Ce code source à compléter est spécifié par le contenu du fichier `template.ml`.

solution.ml Pour faciliter la correction et aussi pour s'assurer de la faisabilité d'un exercice, Learn-OCaml exige que l'enseignant fournisse une solution à l'exercice dans le fichier `solution.ml`. Cette solution est systématiquement corrigée avant le déploiement pour s'assurer de sa validité.

prepare.ml Pour s'assurer que l'étudiant n'utilise que les propriétés du préordre total (i.e. la fonction `compare` et pas la fonction d'égalité générique d'OCaml), il faut se doter d'une définition rusée du type `thing` : le type `thing` doit être abstrait pour que `compare` soit l'unique opération licite sur ses habitants et il faut aussi s'assurer que les opérations de comparaison génériques d'OCaml ne soient pas compatibles avec la relation d'équivalence induite par le préordre. Ces propriétés sont garanties par l'implémentation du module `Thing` définie dans le fichier `prepare.ml`. En effet, d'une part, la signature de ce module cache la définition concrète

du type `thing`. D'autre part, les valeurs de type `thing` sont des paires d'entiers dont seule la première composante est observée par le préordre et dont la seconde composante ne sert qu'à dissocier la fonction d'équivalence de l'égalité générique d'OCaml.

Pour s'assurer que l'algorithme utilisé par l'étudiant a la complexité attendue, il faut pouvoir compter les opérations de comparaison qu'il consomme. C'est le rôle des incréments de la variable globale `comparison_counter` introduite par `prepare.ml`.

Cette machinerie nécessaire à la définition du type `thing` et à l'instrumentation de la fonction de comparaison ne doit pas être montrée à l'élève car elle est relativement complexe et surtout totalement décorrélée de l'exercice de programmation dont il est question ici. Pour cette raison, le contenu du fichier `prepare.ml` est évalué avant la réponse de l'élève, et après `prelude.ml` mais contrairement à ce dernier, il n'est pas divulgué à l'élève.

test.ml Pour comprendre le rôle de la fonction `sample` définie dans `prepare.ml`, il faut maintenant se pencher sur le fichier `test.ml` qui contient le programme de correction automatique à proprement parler. Concentrons-nous donc sur la figure 4.

Dans Learn-OCaml, la correction d'un exercice se résume à la production d'un rapport de correction. En deux mots, un rapport est un document structuré qui contient des annotations d'évaluation sous la forme de points de succès (`Success of int`) ou d'échec (`Failure`). Les constructeurs de données du type `report` sont mis à disposition par le module `Report`. Les lignes 4 et 5 les utilisent pour introduire deux opérateurs `failure` et `success` qui servent à informer l'élève de la nature de son erreur éventuelle ou bien à le gratifier d'un point en cas de réussite.

Vient ensuite la fonction `grade ()` et ses fonctions auxiliaires. En première approximation, cette fonction peut se lire comme suit :

```
1  let grade () = ...
2    check_no_while (fun () -> ( ...
3    test_function_2_against_solution ~gen:10 ~sampler ~before_user ~after expected_type "mem" manual_tests
4  ) |> set_result
```

On distingue de cette façon trois parties importantes : (i) l'appel à la fonction `check_no_while` sert à vérifier que l'étudiant n'utilise pas de boucle `while` dans sa solution ; (ii) l'appel à la fonction `test_function_2_against_solution` sert à comparer la réponse de l'étudiant à la solution de référence de l'enseignant ; (iii) l'appel à la fonction `set_result` sert à transmettre le rapport à l'élève.

La première fonction `check_no_while` est implémentée par observation de l'arbre de syntaxe abstraite de la fonction `mem` de l'élève. La fonction `ast_check_expr` du module `Test_lib` réalise en effet un parcours générique de l'arbre de syntaxe que l'on peut personnaliser en lui passant une fonction `~on_expression`. Dans notre cas d'étude, on lève un drapeau lorsque la construction `while` est utilisée dans la définition de `mem`.

Les deux dernières fonctions sont elles aussi offertes par le module `Test_lib`. Il reste à expliquer la signification des arguments de la fonction `test_function_2_against_solution`. Avant toute évaluation, Learn-OCaml commence par vérifier que la fonction de l'élève a bien le type attendu spécifié par l'argument `expected_type`. Si ce n'est pas le cas, un message d'erreur de type est transmis à l'étudiant dans le rapport et aucun point ne lui est affecté. L'argument `gen` précise le nombre d'entrées utilisées pour comparer la réponse de l'élève à la solution de l'enseignant.

Ces entrées sont obtenues par le biais de deux sources distinctes. Tout d'abord, la fonction consomme tous les couples fournis manuellement par l'enseignant dans l'argument `manual_tests`. Si nécessaire, elle complète ensuite ces tests pour atteindre le nombre `gen` en utilisant le `~sampler`. Il s'agit d'une fonction sans argument qui produit un couple à chaque

fois qu'on l'appelle. Dans notre exemple, la fonction `sampler` est implémentée en générant un arbre aléatoire `t` puis en décidant aléatoirement de chercher un élément existant dans l'arbre (à équivalence près, grâce à l'appel de `Thing.shake`) ou bien de chercher dans cet arbre un élément de type `thing` tiré aléatoirement.

Enfin, on peut enregistrer des actions à accomplir avant et après chaque test unitaire. Dans notre exemple, l'argument `before_user` remet à zéro le compteur de comparaisons. Quant à l'argument `after`, il sert à apporter des compléments au rapport après l'évaluation de la fonction de l'élève et de l'enseignant. On peut donc observer la valeur renvoyée par chacune d'elles mais aussi comparer les sorties standard et d'erreurs. On en profite ici pour vérifier que le nombre de comparaisons consommées par la fonction de l'élève est raisonnable.

4 Conclusion

Premiers retours d'expérience La plateforme Learn-OCaml est déjà utilisée de façon expérimentale par Brigitte Pientka à l'Université McGill au Canada et par Michele Pagani à l'Université Paris Diderot. Comme ces expériences sont en cours, il est difficile de faire des retours d'expérience significatifs. Cependant, nous pouvons déjà rapporter que (i) les équipes enseignantes ont pu traduire leurs énoncés de travaux dirigés dans le système sans difficultés particulières, la documentation de Learn-OCaml semblant prendre en charge la plupart des cas d'usage ; (ii) durant la période de préparation des cours (juillet et août 2018) et le début du semestre de cours (septembre et octobre 2018), 59 rapports de bug ont été soumis et parmi eux, seulement 2 étaient critiques ; (iii) les étudiants réalisent plus vite les feuilles d'exercices quand ils utilisent la plateforme que lors d'une séance de travaux pratiques classiques ; (iv) les étudiants utilisent la plateforme en dehors des séances de travaux pratiques.

Pour la suite Non seulement le code source de Learn-OCaml est disponible sous une licence libre[2] mais plusieurs autres projets pédagogiques collaboratifs vont maintenant s'appuyer sur cette première pierre.

Le premier d'entre eux est la création d'un grand corpus d'exercices corrigés automatiquement, sous licence libre et accessible ainsi à tous les enseignants. On espère ainsi fédérer les efforts de la communauté enseignante dans l'objectif de créer du matériel pédagogique de qualité, réutilisable en cours.

Pour les enseignants d'informatique qui ne connaîtraient pas les arcanes d'UNIX, le projet Learn-OCaml va mettre en place un serveur d'instances qui permettra à tous de déployer la plateforme pour sa classe en quelques clics. Dans le même esprit, une extension de l'application web pour y inclure un éditeur intégré d'exercices est en cours de développement à l'Université de Toulouse III par Erik Martin Dorel et ses étudiants. Enfin, un outil nommé `autogen`[3] s'appuyant sur des extensions PPX permet de simplifier la création d'exercices en produisant les cinq fichiers décrits dans la section 3 à partir d'un unique fichier source.

Références

[1] ACE, The High Performance Editor for the Web. https://ace.c9.io/.

[2] Learn-OCaml. https://github.com/ocaml-sf/learn-ocaml.

[3] Learn-OCaml Autogen. https://github.com/ocaml-sf/learn-ocaml-autogen.

[4] Benjamin Canou, Roberto Di Cosmo, and Grégoire Henry. Scaling Up Functional Programming Education : Under the Hood of the OCaml MOOC. *PACML*, 1(ICFP), August 2017.

[5] Dirk Merkel. Docker : Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.*, 2014(239), March 2014.