Interopérabilité

LU3IN032 : Programmation comparée

Basile Pesin

7 février 2022

Outline

- Langages de haut et bas niveau
  - Machine virtuelle et Foreign Function Interface
  - Représentation des valeurs
  - Gestion de la mémoire
  - Exceptions
  - Callbacks
- Performances : Python vs Cython
- Langages à cible commune
  - Une plate-forme pour les gouverner tous : la JVM
  - Javascript : l'assembleur du Web

Motivation

- Gains de performances sur les sections critiques du code
- Accès système / matériel
- Réutiliser des bibliothèques écrites dans un autre langage
- Mélange de styles de programmation

Déclaration de fonctions externes en OCaml

```ocaml
type c_list

external empty_c_list : unit -> c_list = "empty_c_list"
external cons_c_list : int -> c_list -> c_list = "cons_c_list"
external print_c_list : c_list -> unit = "print_c_list"

let _ =
    let l = empty_c_list () in
    let l1 = cons_c_list 42 (cons_c_list 21 l) in
    print_c_list l;
    print_c_list l1
```

Listing 1 – Utilisation d'une liste chainée implémentée en C

Fonctions C

```c
#define CAML_NAME_SPACE
#include <caml/mlvalues.h>

typedef struct _List {
    int data;
    struct _List *next;
} List;

CAMLprim value empty_c_list(value u) { [...] }

CAMLprim value cons_c_list(value iv, value lv) { [...] }

CAMLprim value print_c_list(value lv) { [...] }
```

Listing 2 – Fonctions de gestion de liste chainée en C

Compilation

```
# Compile clist.c vers une bibliotheque clist.a
gcc -o clist.o clist.c -I~/.opam/default/lib/ocaml
ar rcs clist.a clist.o
# Compile clist.ml vers un executable
ocamlc -c -o clist.cmo clist.ml
ocamlc -o clist.byte -custom clist.cmo -cclib -L . -cclib -lclist
```

Représentation des valeurs : Types de bases

OCaml	C	Tag	value -> C, C -> value
int	int	NA	Int_val Val_int
bool	int	NA	Bool_val Val_bool
float	double	Double_tag	Double_val Store_double_val
string	char *	String_tag	String_val caml_alloc_initialized_string

n-uplets, enregistrements

```c
CAMLprim value update_record(value t) {
CAMLparam1(t);

int x = Val_int(Field(t, 1));
Store_field(t, 1, Int_val(x + 1));
CAMLreturn(Val_unit);
}
```

```ocaml
type rec2 = {
    a1 : string;
    a2 : int; (* Pas mutable ! *)
    a3 : bool;
}

external update_record : rec2 -> unit = "update_record"

let _ =
    let r1 = {a1 = "coucou";
              a2 = 41;
              a3 = false} in
update_record r1;
print_int r1.a2;
```

Types sommes, ADTs

```
type t =
| A              (* Val_int(0) *)
| B of string    (* bloc, tag 0, taille 1 *)
| C              (* Val_int(1) *)
| D of bool      (* bloc, tag 1, taille 1 *)
| E of t * t     (* bloc, tag 2, taille 2 *)
```

Listing 3 – Exemple d'encodage d'un type somme

Abstract et Custom types

```
#define List_val(v) (*(List **) Data_abstract_val(v))

CAMLprim value empty_c_list(value unit) {
    List *l = NULL;

    v = caml_alloc(sizeof(List *), Abstract_tag);
    List_val(v) = l;
    CAMLreturn(v);
}
```

Listing 4 – Construction d'une valeur de type abstrait

```
CAMLprim value print_c_list(value lv) {
    List *l = List_val(lv);

    while(l) {
        printf("%d ", l->data);
        l = l->next;
    }
    printf("\n");

    CAMLreturn(Val_unit);
}
```

Abstract et Custom types - custom_operations

```
struct custom_operations {
    char const *identifier;
    void (*finalize)(value v);
    int (*compare)(value v1, value v2);
    intnat (*hash)(value v);
    void (*serialize)(value v, uintnat * bsize_32,
                      uintnat * bsize_64);
    uintnat (*deserialize)(void * dst);
    int (*compare_ext)(value v1, value v2);
    const struct custom_fixed_length* fixed_length;
};
```

- `finalize` pour la libération de la mémoire
- `compare` et `compare_ext` pour comparaison deux valeurs
- `hash` permet de définir le hashing d'une valeur, comme son nom l'indique.
- `serialize` et `deserialize` pour la sérialisation

custom_operations pour c_list

```
#define List_val(v) (*(List **) Data_custom_val(v))

void finalize_c_list(value v) {
    List *l = List_val(v);
    while(l) {
        List *tmp = l;
        l = l->next;
        free(tmp);
    }
}

struct custom_operations clist_ops = {
    "c_list", finalize_c_list,
    custom_compare_default, custom_hash_default,
    custom_serialize_default, custom_deserialize_default,
    custom_compare_ext_default, custom_fixed_length_default
    };

    CAMLprim value empty_c_list(value unit) {
    List *l = NULL;
    v = caml_alloc_custom(&clist_ops, sizeof(List *), 0, 1);
    List_val(v) = l;
    CAMLreturn(v);
}
```

Gestion de la mémoire - règles

On va parfois allouer dans le tas, donc le GC peut se déclencher

- Les paramètres de la fonction doivent être marqués par `CAMLparam0()`, `CAMLparam1(x)`, `CAMLparam2(x, y)`, `...`
  Alloue un cadre et y place les paramètres
- On utilise `CAMLreturn(v)` plutôt que `return` (et il faut toujours l'utiliser).
  Supprime le cadre
- Pour les variables locale de type `value`, on utilise `CAMLlocal1(x)`, `CAMLlocal2(x, y)`, `...`
  Place les variables dans le cadre

Gestion de la mémoire - exemple

```c
CAMLprim value cons_c_list(value iv, value lv) {
    CAMLparam2(iv, lv);
    CAMLlocal1(nv);

    // Get the C values back
    List *l = List_val(lv);
    int i = Int_val(iv);

    List *nl = malloc(sizeof(List));
    nl->data = i;
    nl->next = l;

    // Return the new list, packed as a caml value
    nv = caml_alloc(1, Abstract_tag);
    List_val(nv) = nl;
    CAMLreturn(nv);
}
```

Listing 6 – Gestion de la mémoire dans la fonction `cons_c_list`

Comment lever des exceptions ?

Deux exceptions « prédéfinies »

- `Failure : string -> exn` peut être levée via `void caml_failwith(const char *msg)`
- `Invalid_argument : string -> exn` peut être levée via `void caml_invalid_argument(const char *msg)`

Exception utilisateur

```
exception Error of string
let _ = Callback.register_exception "my_error" (Error "dummy")
```

- `caml_raise_constant(e)` pour une exception sans paramètre
- `caml_raise_with_arg(e, v)` pour une exception avec un paramètre
- `caml_raise_with_args(e, n, v)` pour une exception à `n` arguments (`v` est un tableau de `value` de taille `n`)
- `caml_raise_with_string(e, msg)` pour une exception avec comme seul paramètre une chaîne de caractères.

```
void raise_error(char * msg) {
    caml_raise_with_string(*caml_named_value("my_error"), msg);
}
```

Callbacks

```
CAMLprim value apply(value closure, value arg) {
    CAMLparam2(closure, arg);

    CAMLreturn(caml_callback(closure, arg));
}
```

```
external apply : ('a -> 'b) -> 'a -> 'b = "apply"

let _ =
print_int (apply (fun x -> x * 2) 21)
```

Itération de Newton, Python

```
def sq1(x):
    return x**2 - 2

def sq1der(x):
    return 2*x

def newton1(f, der, init):
    epsilon = 0.001
    x = init

    while True:
        x1 = x - f(x)/der(x)

        if abs(x - x1) < epsilon:
            return x1
        x = x1
```

Itération de Newton, Cython

```
cdef float sq2(float x):
    return x**2 - 2

cdef float sq2der(float x):
    return 2*x

ctypedef float (*ffunc)(float)

cdef newton2(ffunc f, ffunc der, float init):
    cdef float epsilon, x, x1
    epsilon = 0.001
    x = init

    while True:
        x1 = x - f(x)/der(x)

        if abs(x - x1) < epsilon:
            return x1
        x = x1
```

Synthèse des résultats

Programme	Interpreté	Compilé non typé	Compilé typé	Compilé, typé `boundscheck(False)`
Newton	1.13s	0.69s	0.04s	0.04s
InsertSort	4.07s	1.98s	0.07s	0.05s
Erathostene	1.73s	1.23s	0.85s	0.82s
MatMult	6.52s	2.44s	0.11s	0.05s

La JVM, un succès industriel

Langages visant la JVM

Conçus pour la JVM :

- Clojure
- Kotlin
- Scala
- Groovy

Compilateurs pour des langages existants (ou dialectes) :

- Jython pour Python
- OCaml-Java pour OCaml
- Bigloo pour Scheme

Interactions Clojure-Java

Il est très facile de manipuler des instances de classes Java : un nouvel objet peut être créé grâce au mot-clé `new`.

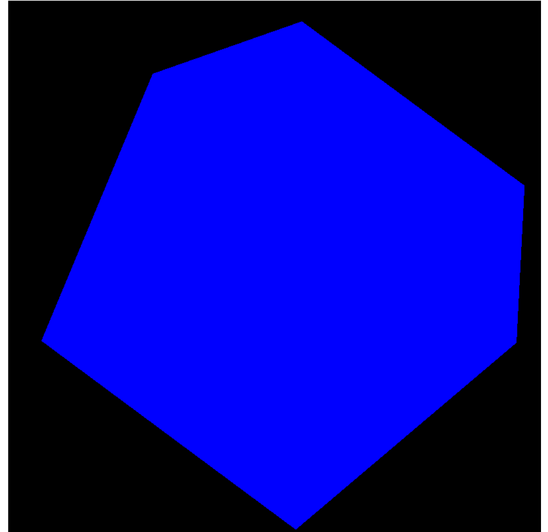
```
(def p (new java.awt.Point 1 2))
(.translate p 2 3)
(.getX p) ;; 3.0
(.-x p)   ;; 3
```

Listing 7 – Instantiation et manipulation d'un objet

```
(def list (new java.util.ArrayList))
(doto list (.add 41) (.add 42))
```

Listing 8 – Macro `doto`

Démo : le projet Yaw



Javascript : l'assembleur du Web

Conçus pour être compilé / transpilé vers JavaScript :

- TypeScript
- CofeeScript
- ClojureScript
- Reason

Compilateurs vers JS pour des langages existants :

- Js_of_ocaml
- asm.js (assembleur, rendu « obsolète » par WebAssembly)
- pleins d'autres !

Js_of_ocaml - Utilisation de base

```
let _ = print_endline "Hello␣World"
```

Listing 9 – hello.ml

```
ocamlc -o hello.byte hello.ml
js_of_ocaml hello.byte
node hello.js # affiche "Hello World"
```

Listing 10 – Compilation avec Js_of_ocaml

```
<html>
    <head>
        <title>Hello World</title>
        <meta charset="UTF-8"/>
    </head>
    <body>
        <script src="hello.js"></script>
    </body>
</html>
```

Listing 11 – Page hello.html chargeant hello.js

Js_of_ocaml - Création d'objets

```
let albert = object%js
    val surname = Js.string "Albert"
    val name = Js.string "Einstein"
    val mutable age = 42
end

albert : < age : int Js.prop;
           name : Js.js_string Js.t Js.readonly_prop;
           surname : Js.js_string Js.t Js.readonly_prop > Js.t
```

Js_of_ocaml - Type et manipulation d'objets

```
class type person = object
    method surname : js.js_string js.t js.readonly_prop
    method name : js.js_string js.t js.readonly_prop
    method age : int js.prop
    method inc_age : unit -> unit js.meth
end

let albert : person js.t = object%js(self)
    val surname = js.string "albert"
    val name = js.string "einstein"
    val mutable age = 42
    method inc_age () =
        self##.age := self##.age + 1
end

let _ =
    print_int albert##.age; (* 42 *)
    albert##inc_age ();
    print_int albert##.age; (* 43 *)
```

Js_of_ocaml - Interaction avec le DOM - HTML

```
<html>
    <head>
        <title>Compteur</title>
        <meta charset="UTF-8"/>
    </head>
    <body>
        <script src="script.js"></script>
        <form action="">
            <input id="decr" type="button" value="decrementer" />
            <input id="incr" type="button" value="incrementer" />
        </form>
        <p>Compteur : <b id="counter">0</b></p>
    </body>
</html>
```

Js_of_ocaml - Interaction avec le DOM - Fonctions de base

```
open Js_of_ocaml
open Js_of_ocaml_tyxml
module T = Tyxml_js.Html5

let by_id s = Dom_html.getElementById s

let of_node = Tyxml_js.To_dom.of_node

let replace_child parent child =
let oldchild = Js.Opt.get (parent##.childNodes##item 0)
    (fun _ -> invalid_arg "replace_child") in
ignore (parent##replaceChild child oldchild)
```

Js_of_ocaml - Interaction avec le DOM - Script

```
14 let counter = ref 0
15
16 let update_counter diff (_ : #Dom_html.event Js.t) =
17     counter := !counter + diff;
18     replace_child (by_id "counter") (of_node T.(txt (string_of_int !counter)));
19     Js._true
20
21 let init (_ : #Dom_html.event Js.t) =
22     (by_id "incr")##.onclick := Dom_html.handler (update_counter 1);
23     (by_id "decr")##.onclick := Dom_html.handler (update_counter (-1));
24     Js._true
25
26 let _ =
27     Dom_html.window##.onload := Dom_html.handler init
```