Programmer en OCaml pour et sur les calculatrices Numworks

Laura Ly
Basile Pesin
Lilian Besson
Emmanuel Chailloux

28 janvier 2026

Plan

1. Pourquoi OCaml sur une calculatrice : matériel, contraintes et motivations
2. OMicroB et son portage
3. Implémentation d'un top-level
4. Limitations et enjeux pédagogiques

Pourquoi OCaml sur une calculatrice : matériel, contraintes et motivations

- Programmation de plus en plus présente dans les programmes scolaires
  - **Lycée** : SNT, NSI, Mathématiques…
  - **Classes préparatoires :** filière MP2I, cours d'informatique…
  - **Universités :** aussi

- Fourniture scolaire omniprésente en filière scientifique : **la calculatrice "lycée"**
  - Mode examen
  - Calculs numériques avancés, calculs matriciels, résolutions d'équations…
  - Mini-Python : écriture, modification et exécution de programmes, saisie clavier et affichage des résultats

Pourquoi OCaml sur une calculatrice : Matériel, contraintes et motivations

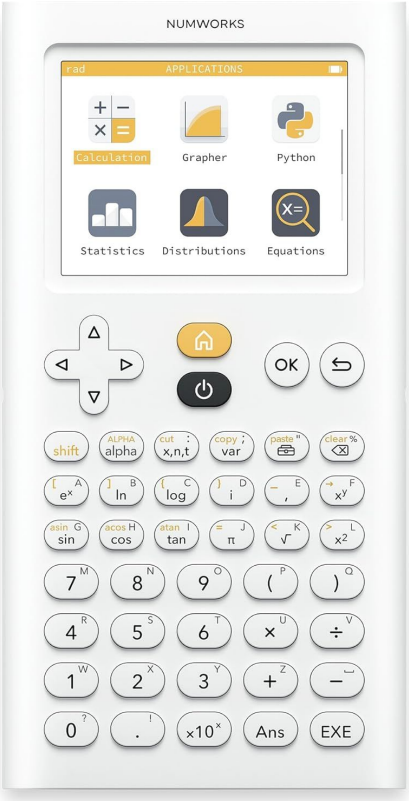


Figure 1 – Une calculatrice NumWorks

- Écran IPS (LCD) de 320 × 240 pixels
- Clavier alphanumérique

- Processeur ARMv7 à 216 MHz
- 256 Kio de RAM
- 8 Mio de Flash

- Système d'exploitation *open source* : Epsilon
- Système de gestion de fichiers (lié à l'application préinstallée Python)
- Installation d'applications personnalisées (`.nwa`)

OMicroB et son portage

- **OMicroB** : une machine virtuelle (VM)
  - implémentée en C → portable sur microcontrôleurs (AVR, PIC32 par exemple)
  - peut exécuter du *bytecode* produit par le compilateur OCaml

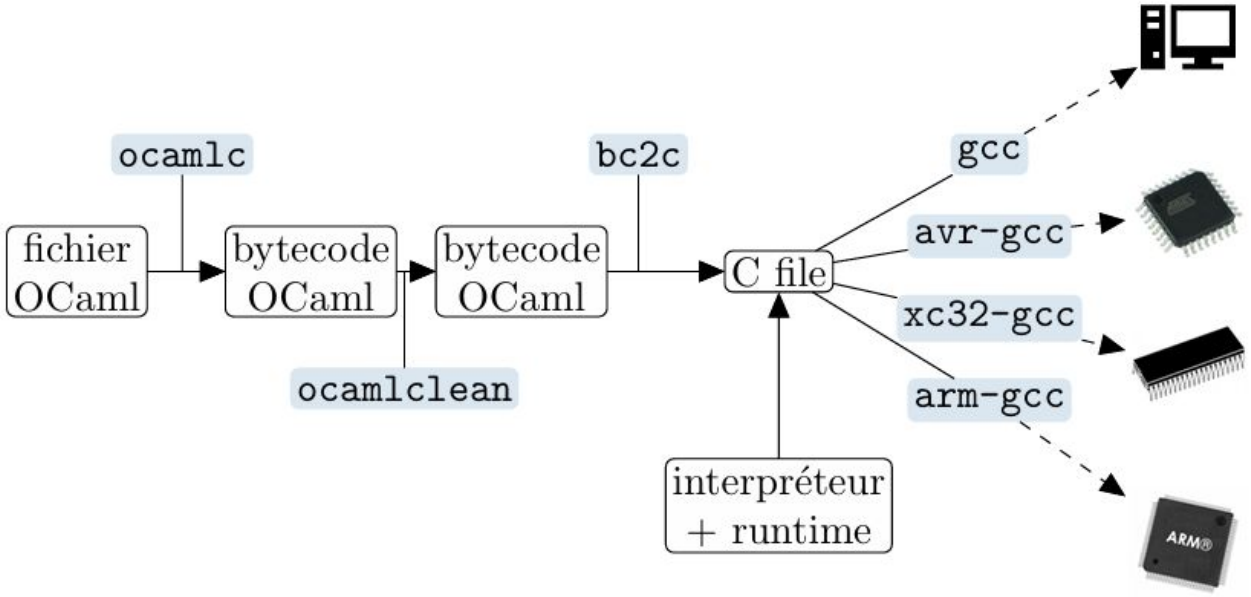


Figure 2 – Chaîne de compilation d'OMicroB

OMicroB et son portage

- **Ajout de NumWorks comme nouvelle cible d'OMicroB**
  - Adaptation du *Makefile* et du *configure*
  - Configuration matérielle (`device_config`)

- **Bibliothèque standard**
  - Accès écran, clavier, couleurs
  - Accès au stockage fichiers
  - Gestion entrées-sorties

↪ **Résultats**
  - Compilation de OCaml en une application `.nwa` native
  - Exécution directe sur la calculatrice

Démonstration du jeu de la vie sur NumWorks

```
class world width height =
  object(self)
    val mutable tcell = Array.make_matrix width height false
    val mutable gen = 0
    method draw() =
      Screen.clear ();
      for i = 0 to (width-1) do
        for j = 0 to (height-1) do
          draw_cell i j tcell.(i).(j)
        done
      done
    method getCell(i,j) = tcell.(i).(j)
    method setCell(i,j,c) = tcell.(i).(j) <- c
    method getCells = tcell
```

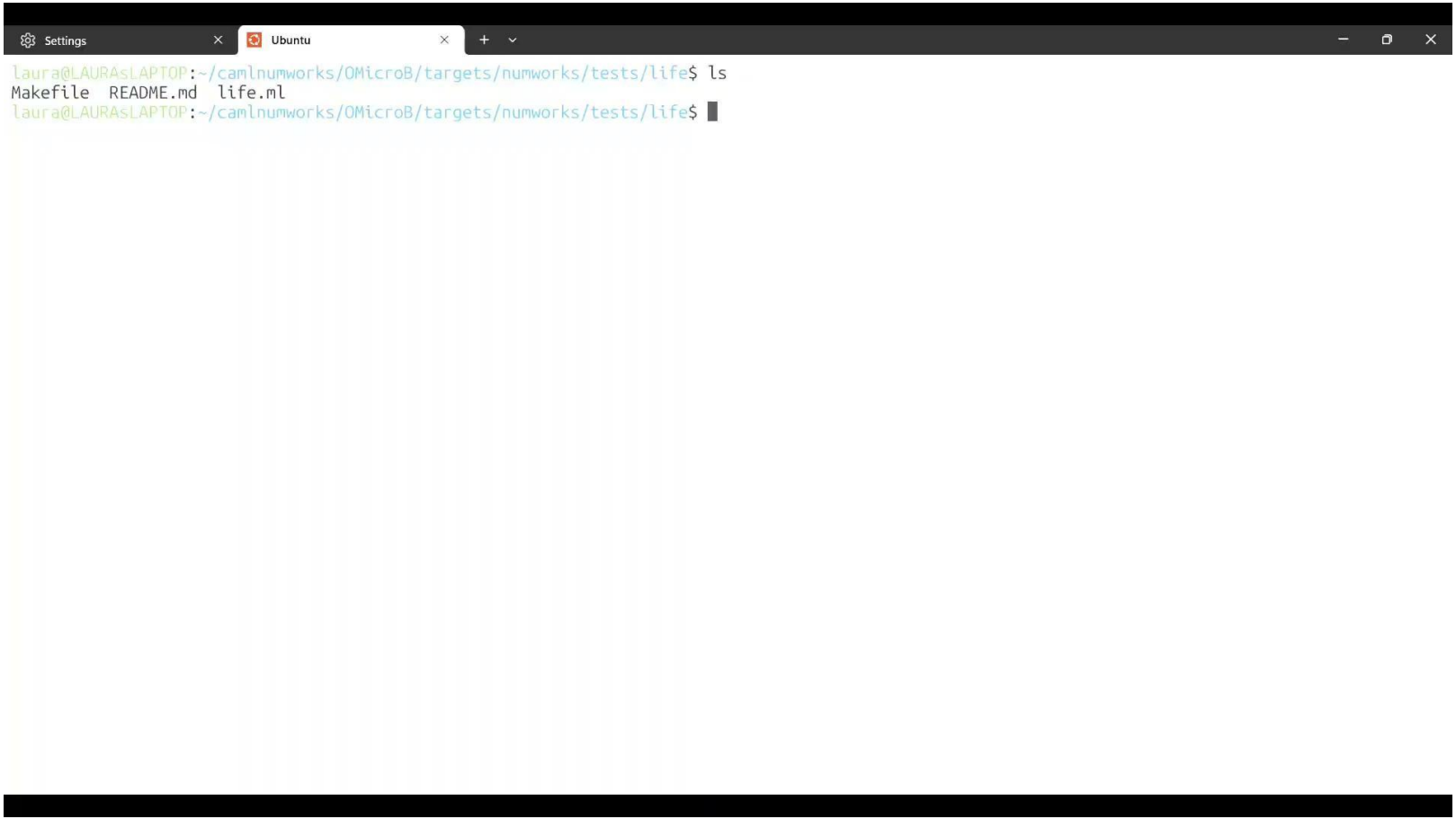
Démonstration du jeu de la vie sur NumWorks

```
let draw_cursor cx cy =
  Screen.fill_rect Color.red (cx * 10) (cy * 10) 4 2;
  Screen.fill_rect Color.red (cx * 10) (cy * 10) 2 4;
  Screen.fill_rect Color.red (cx * 10 + 6) (cy * 10) 4 2;
  Screen.fill_rect Color.red (cx * 10 + 8) (cy * 10) 2 4;
  Screen.fill_rect Color.red (cx * 10) (cy * 10 + 8) 4 2;
  Screen.fill_rect Color.red (cx * 10) (cy * 10 + 6) 2 4;
  Screen.fill_rect Color.red (cx * 10 + 6) (cy * 10 + 8) 4 2;
  Screen.fill_rect Color.red (cx * 10 + 8) (cy * 10 + 6) 2 4;;
```

Démonstration du jeu de la vie sur NumWorks

```
let edit w =
  let rec loop cx cy =
    w#draw();
    draw_cursor cx cy;
    Screen.print "Edition" 0 0;
    match Keyboard.wait_key_press () with
    | Key_left -> loop ((cx + width - 1) mod width) cy
    | Key_up -> loop cx ((cy + height - 1) mod height)
    | Key_right -> loop ((cx + 1) mod width) cy
    | Key_down -> loop cx ((cy + 1) mod height)
    | Key_ok -> w#setCell(cx,cy,(not w#getCells.(cx).(cy))); loop cx cy
    | Key_exe -> w
    | _ -> loop cx cy
  in loop (width/2) (height/2);;
```

Démonstration du jeu de la vie sur NumWorks



Implémentation d'un top-level

- **Problème principal :** embarquer un *lexer*, un *parser*, un *typer*, et un compilateur *bytecode*

  ↪ **Très lourd !**

- **Possibilité :** embarquer directement ocamlc/ocamlrun sur la calculatrice via OMicroB

- **Solution envisagée (et utilisée) :** portage de l'interprète 4.13 de Camlboot via OMicroB
  - interprète d'AST
  - écrit en OCaml (dont Menhir)

Implémentation d'un top-level

- **Isolement des modules utiles :** `Parsetree`, `Lexer`, `Parser`...
  - Réécritures de modules comme `Lazy` car incompatibles avec OMicroB
- **Imports de modules spécifiques à CamlBoot :** `Eval` et toutes ses dépendances
  - Suppression des modules incompatibles avec OMicroB, comme `Format`
- **Implémentation de la boucle REPL**
  - Utilisation de `Parsetree`, `Lexer`, `Parser` et `Eval`
  - Environnement persistant
  - Possibilité de charger des fichiers
  - Gestion et affichage d'erreurs

Implémentation d'un top-level

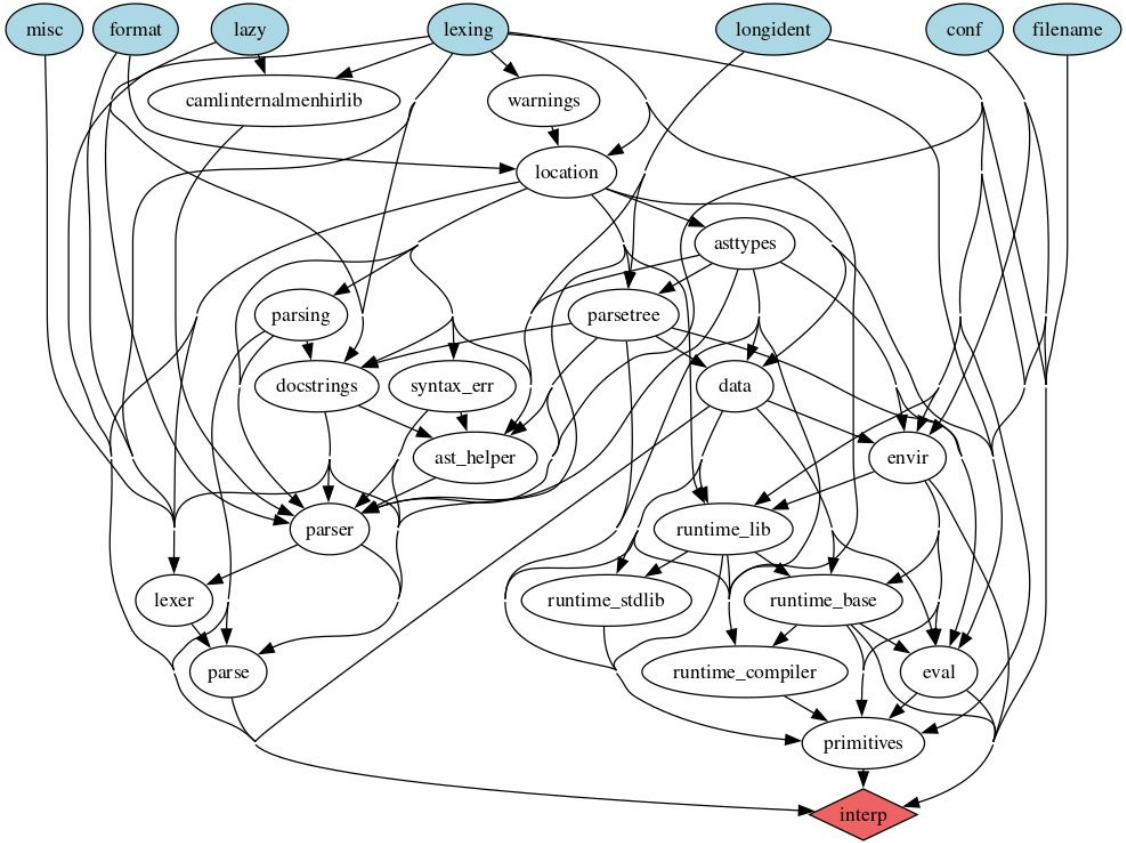


Figure 3 – Schéma des dépendances de l'interpréteur de Camlboot

Démonstration du top-level

```
module Keyboard = struct
  external wait_key_press : unit -> int = "numworks_keyboard_wait_key_press"
  external scan : unit -> unit = "numworks_keyboard_scan"
  external key_down : int -> bool = "numworks_keyboard_key_down"
end

module Color = struct
  type t = int

  let make r g b =
    (r lsl 11) + ((2 * g) lsl 5) + b

  let black : int = make 0 0 0
  let red : int = make 31 0 0
end

module Screen = struct
  external clear : unit -> unit = "numworks_screen_clear"
  external print : string -> int -> int -> unit = "numworks_screen_print"
  external fill_rect : Color.t -> int -> int -> int -> int -> unit = "numworks_screen_fill_rect"
end
```

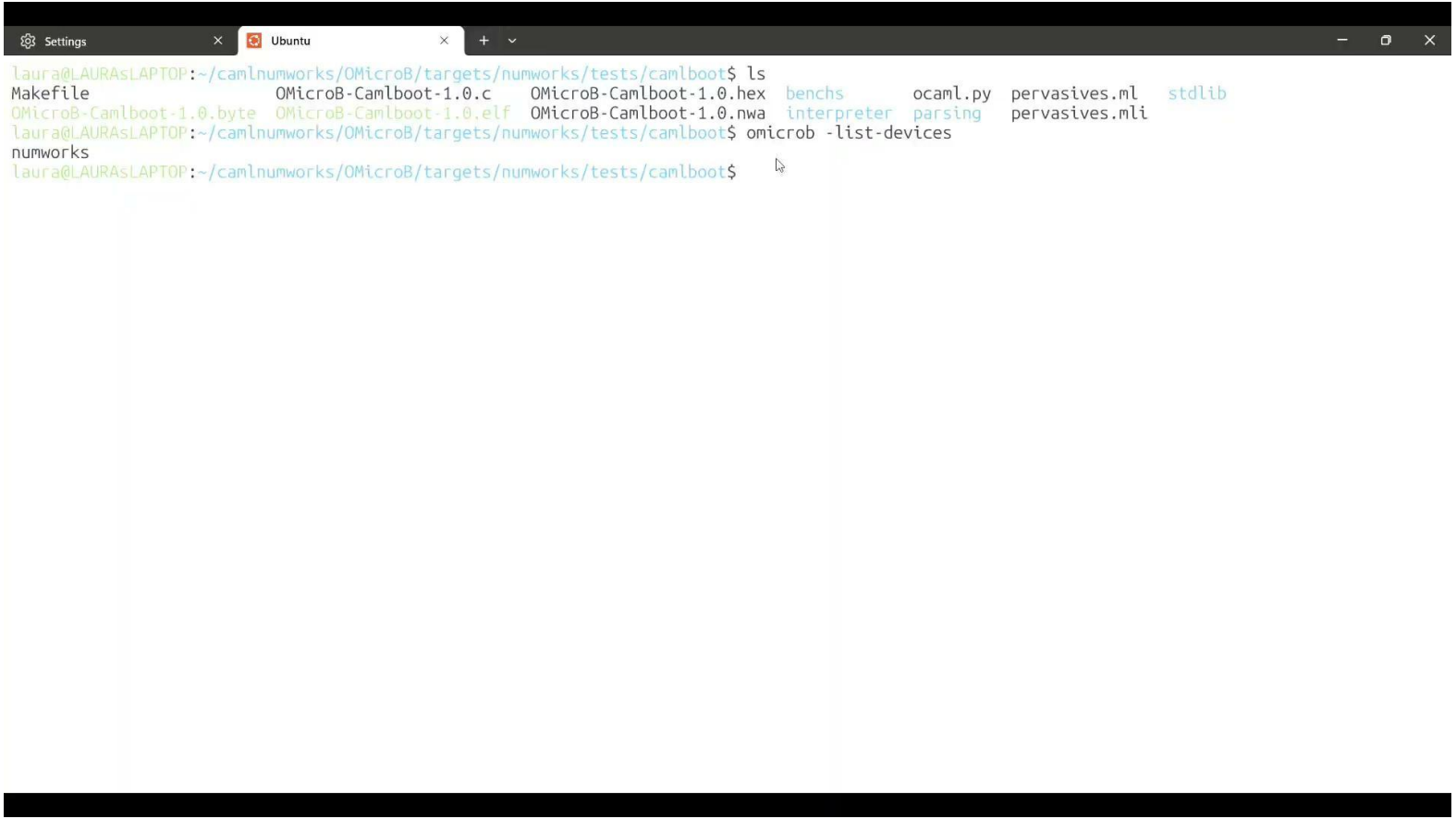
Démonstration du top-level

```
let fill_rect c x y w h = Screen.fill_rect c x (y + 10) w h

let draw_cell x y alive =
  if alive then fill_rect Color.black (x*10) (y*10) 10 10

type world = {
    width: int;
    height: int;
    tcell: bool Array.t Array.t;
  }

let mk_world width height = {
    width = width;
    height = height;
    tcell = Array.make_matrix width height false;
  }
```

Démonstration du top-level

```
let edit w =
  try
    let x = ref (width/2) and y = ref (height/2) in
    while true do
      draw w;
      draw_cursor !x !y;
      Screen.print "Edition\n" 0 0;
      match Keyboard.wait_key_press () with
      | 0 (* left *) -> x := (!x + width - 1) mod width
      | 1 (* up *) -> y := (!y + height - 1) mod height
      | 2 (* down *) -> y := (!y + 1) mod height
      | 3 (* right *) -> x := (!x + 1) mod width
      | 4 (* ok *) -> set_cell w !x !y (not (get_cell w !x !y))
      | 45 (* exe *) -> raise Fin
      | _ -> ()
    done
  with Fin -> ();;
```

Démonstration du top-level



Limitations et enjeux pédagogiques

- Typage dynamique pour le top-level
- Lent
- Contrainte mémoire
  - On pourrait potentiellement stocker dans la flash ce qui est “fixe” comme l’AST de la librairie standard

- Enjeux pédagogiques :
  - Expérimentation autonome et immédiate, encourager l’exploration du langage
  - Introduction à la compilation