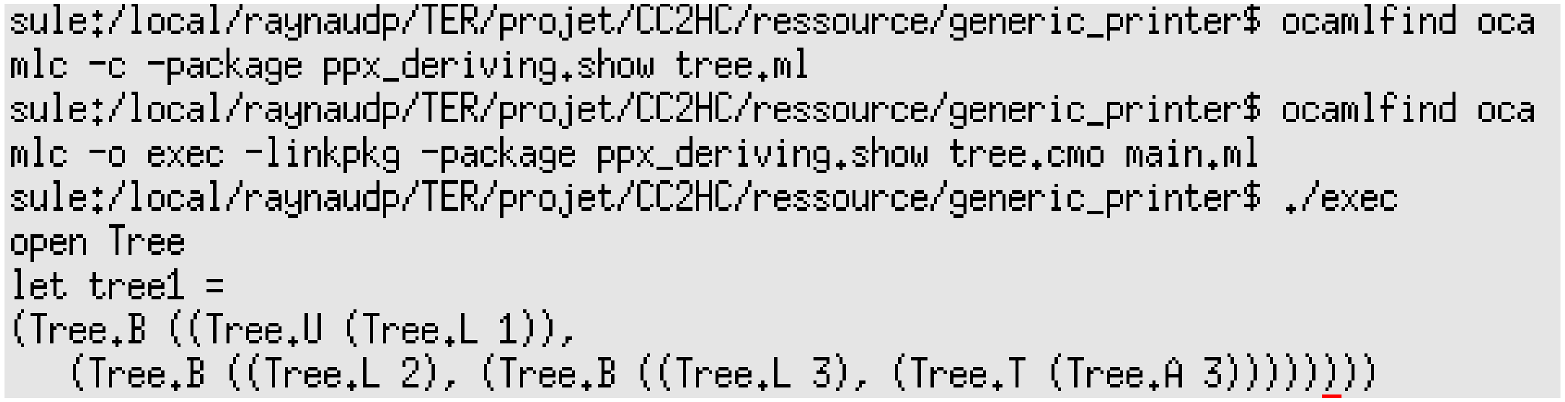
```
type test =
        | A of int
  [@@deriving show]
;;

type tree =
  | U of tree
  | B of tree * tree
  | L of int
        | T of test
  [@@deriving show]
;;
```

```
open Tree

let output_ocaml_decl_of_tree: string -> tree -> unit
=
  fun name tree ->
    (String.concat "\n" [ "open Tree" ; "let " ^ name
^ " =" ; show_tree tree ; ";;"])
    |> print_string
;;

let t = B(U(L 1), B (L 2, B( (L 3), (T(A 3)) ) )  ) ;;
let _ =
      output_ocaml_decl_of_tree "tree1" t ;;
```



```
sule:/local/raynaudp/TER/projet/CC2HC/ressource/generic_printer$ ocamlfind oca
mlc -c -package ppx_deriving.show tree.ml
sule:/local/raynaudp/TER/projet/CC2HC/ressource/generic_printer$ ocamlfind oca
mlc -o exec -linkpkg -package ppx_deriving.show tree.cmo main.ml
sule:/local/raynaudp/TER/projet/CC2HC/ressource/generic_printer$ ./exec
open Tree
let tree1 =
(Tree.B ((Tree.U (Tree.L 1)),
   (Tree.B ((Tree.L 2), (Tree.B ((Tree.L 3), (Tree.T (Tree.A 3))))))))
```