Bonjour

Méthodologie de Programmation avec Objective CAML

Cours 6

Vincent Poirriez

courriel :Vincent.Poirriez@univ-valenciennes.fr

Deux *vraies* applications

Nettoyer une arborescence de fichiers

Objectif supprimer tous les fichiers auxiliaires

Les types de fichiers à supprimer en argument

Les répertoires à supprimer en argument

Descente récursive : oui ou non ?

Le répertoire à nettoyer en argument

Demander confirmation avant de supprimer en laissant le temps de la réaction

Un soucis à garder en tête : si possible portable sur différents OS.

On regroupe dans un fichier `borepert.ml` les utilitaires sur les répertoires.

Listes des fichiers d'un répertoire

borepert.ml

```
7  let ls d =
8    let handle = Unix.opendir d in
9    let rec aux res =
10     try aux ((Unix.readdir handle)::res)
11     with End_of_file -> res
12   in
13   let r = aux [] in
14   Unix.closedir handle;
15   r
```

Distinguer les sortes de fichiers

Un fichier est normal si le champ `st_kind` est `S_REG`.

```
18 let fichiers_normaux tout =
19   let est_normal s = (Unix.stat s).Unix.st_kind = Unix.S_REG
20   in List.filter est_normal tout
```

C’est un répertoire si :

```
23 let répertoires tout =
24   let est_rép s = (Unix.stat s).Unix.st_kind = Unix.S_DIR
25   in  List.filter est_rép tout
```

Filtrer les noms à supprimer

Les critères sont : soit posséder un suffixe donné

soit terminer par un caractère donné,

soit commencer par un caractère . . .

Il faut donc sélectionner les noms vérifiant le *ou* d'une liste de propriétés :
Créons un fichier `logiclists.ml` pour regrouper les fonctions manipulant des listes de propriétés.

Logiclists

```
6  let rec ou_de lprop s = match lprop with
7  | [] -> false
8  | [p] -> p(s)
9  | p:: _ when p(s) -> true
10 | _:: suite -> ou_de suite s
```

La fonction de sélection

Sélectionner la sous liste des noms vérifiant au moins une des propriétés :

```
13  let au_moins_une propriétés éléments =
14    let à_sélectionner s = ou_de propriétés s in
15    List.filter à_sélectionner éléments
```

La fonction `List.filter` a pour type :

('a -> bool) -> 'a list -> 'a list

et `List.filter prop liste` renvoie la sous liste de `liste` des éléments vérifiant `prop`

Distinguer les vrais sous-répertoires

`ls` renvoie aussi le répertoire courant et le répertoire parent.

`r` est un vrai répertoire si ce n'est pas . et ce n'est pas pas ..

```
18 let rec et_de lprop s = match lprop with
19 | [] -> true
20 | [p] -> p(s)
21 | p:: _ when not (p(s)) -> false
22 | _:: suite -> et_de suite s
```

```
25 let toutes propriétés éléments =
26   List.filter (et_de propriétés) éléments
```

Borepert

```
28 let vrais_sous_rep sous_rep =
29   let n'est_pas_courant r = r <> Filename.current_dir_name
30   and n'est_pas_père r = r <> Filename.parent_dir_name  in
31   Logiclists.toutes [n'est_pas_courant; n'est_pas_père] sous_rep
```

Les outils de suppression

Il faut :

Écrire les propriétés qui permettront de décider de supprimer

Prévoir les informations à l'utilisateur

Prévoir une réponse avec un time-out

On regroupe tout cela dans un fichier `outilsbalai.ml`

Les messages

Regroupons les messages à l'utilisateur au début du fichier.

outilsbalai.ml

```
6  let rien_fait = "je␣n'ai␣rien␣fais"
7  let approbations = ['y';'Y';'o';'O']
8  let default_approb = "[O]/N␣"
9  let non_suppr = "certains␣fichiers␣non␣supprimés"
```

Intérêt : internationalisation plus facile.

Les propriétés pour être supprimer

Un fichier est supprimable si il(son nom) possède un suffixe de la liste des *suffixes*
ou si il commence par un caractère de la liste des *premiers* caractères
ou si il fini par un caractère de la liste des *derniers* caractères
ou si il fait parti de la liste `complets` des noms à supprimer.
Pour utiliser ce qui est dans le fichier `borepert.ml` :

```
12  let supprimable suffixes premierchar dernierchar complets =
13    [(fun s -> List.exists (Filename.check_suffix s) suffixes);
14     (fun s ->
15       List.exists (( = ) s.[String.length s -1]) dernierchar);
16     (fun s -> List.exists(( = ) s.[0]) premierchar);
17     (fun s -> List.mem s  complets);
18   ]
```

supprimable :

string list -> char list -> char list -> string list -> (string -> bool) list

Informer de la suppression prochaine

```
21  let informer_suppr fichiers delay =
22    Printf.printf "Dans␣le␣répertoire:␣%s\n" (Sys.getcwd());
23    Printf.printf
24      "Suppression␣dans␣%2.1f␣secondes␣des␣fichiers:\n" delay;
25    List.iter (Printf.printf "%s␣") fichiers;
26    print_endline default_approb;
27    flush stdout
```

Supprimer effectivement si c’est demandé

```
30 let supprime_si_confirme fichiers message =
31   if message="" || (List.mem message.[0] approbations)
32   then (
33     try  List.iter Sys.remove fichiers with
34     | _ -> prerr_endline non_suppr)
35   else print_endline rien_fait
```

Il faut prévoir que l’on ne soit pas autoriser à supprimer. D’où le `try`

Listes des fichiers à supprimer, et des sous-répertoires

En utilisant ce qui est fait :

```
38  let à_supprimer rep supprimables =
39    Sys.chdir rep;
40    let tous = Borepert.ls Filename.current_dir_name in
41    let norm = Borepert.fichiers_normaux tous in
42    let sousrep = Borepert.répertoires tous in
43    ( Logiclists.au_moins_une supprimables norm ,
44      Borepert.vrais_sous_rep sousrep)
```

Temporiser une action

On veut laisser le temps à l'utilisateur de confirmer ou d'infirmer.

Sa réponse sera sur l'entrée standard.
Au bout d'un *certain* temps ou si on obtient une réponse, on fait l'action.

Utilisation de la fonction `Thread.wait_timed_read` pour attendre un caractère à lire sur le canal.

```
49 let temporise action defaut delay chin =
50   let entrée () =
51     let entrée_présente =
52       Thread.wait_timed_read (Unix.descr_of_in_channel chin) delay
53     in if entrée_présente then input_line(chin) else defaut
54   in
55   action (entrée ())
```

La fonction de suppression temporisée

```
62  let rec supprime d suffixes derniers premiers complets niveaux rep=
```

Les arguments : le délai d'attente, les suffixes, derniers et premiers caractères, les noms complets, les niveaux de descente récursives, le répertoire de départ.

```
63    let prop_suppr = supprimable suffixes premiers derniers complets i
64    let àsupr , ssrep = à_supprimer rep prop_suppr in
```

Les fichiers à supprimer et les vrais sous-répertoires.

```
65    (match àsupr with
66    | [] -> ()
67    | _ -> begin
68        informer_suppr àsupr d;
69        temporise (supprime_si_confirme àsupr) "oui" d stdin
70    end);
```

Ne rien faire si il n'y a rien à faire, supprimer après confirmation ou temps dépassé sinon. (suite après)

La descente récursive

Il reste à faire le parcours récursif si on n'a pas atteint le niveau 0 en faisant attention que l'on peut ne pas avoir de droits.

```
71    if niveaux <> 0 then
72      List.iter
73        (fun r -> try begin
74          supprime
75            d suffixes derniers premiers complets (niveaux-1) r;
76          Sys.chdir Filename.parent_dir_name
77        end
78        with Sys_error m ->  prerr_endline m)
79        ssrep
```

Ce qui est la fin de la fonction `supprime`

Analyser la ligne de commande

On prévoit une commande `à options` :

-r n n est le niveau de descente récursive

-d rep rep est le répertoire de départ

-sl d d est le temps d'attente de confirmation

-lc c c est un dernier caractère à prendre en compte

-pc c c est un premier caractère à prendre en compte

-s suf suf est un suffixe (sans .) à prendre en compte

Nous utilisons le module `Arg` qui est fait pour faciliter l'analyse d'une ligne de commande.

La fonction Arg.parse

```
val parse : (string * spec * string) list -> (string -> unit) -> string -> unit

Arg.parse speclist anonfun usage_msg parses the command line.
 speclist is a list of triples (key, spec, doc).
 key is the option keyword, it must start with a '-' character.
 spec gives the option type and the function to call when this option is found on the com
 doc is a one-line description of this option.
 anonfun is called on anonymous arguments.
The functions in spec and anonfun are called in the same order as their arguments appear
on the command line.

If an error occurs, Arg.parse exits the program, after printing an error message as follo
    * The reason for the error: unknown option, invalid or missing argument, etc.
    * usage_msg
    * The list of options, each followed by the corresponding doc string.

For the user to be able to specify anonymous arguments starting with a -,
 include for example ("-", String anonfun, doc) in speclist.

By default, parse recognizes two unit options, -help and --help,
which will display usage_msg and the list of options, and exit the program.
You can override this behaviour by specifying your own -help and --help options in specli
```

Les variables globales modifiables

Qui contiennent les valeurs par défaut.

balai.ml

```
6  let suffixes = ref ["cmo"; "cmi"; "cmx"; "o"; "log"; "aux"]
7  let derniers = ref ['~';'%']
8  let premiers = ref ['#';'%']
9  let complets = ref ["core"]
10 let niveaux = ref 0
11 let rep = ref (Sys.getcwd())
12 let delay = ref (1.5)
```

Un petit outil pour ajouter un élément à une référence sur une liste :

```
15 let ajoute_ref r s = r := (s :: !r)
```

La fonction d'analyse de la ligne

```
17 let parsecl () =
18  Arg.parse
19   [ ("-r", Arg.Int (fun n -> niveaux := n),
20      "Fait un nettoyage récursif sur n niveaux.
21 ␣␣␣␣␣␣␣n négatif pour une descente non bornée");
```

-r Est l'option

Arg.Int spécifie qu'il doit y avoir un entier après le `-r`

La fonction sous `Arg.Int` sera appliquée à cet entier.

Fait un ... est le commentaire qui documente l'option

Le premier argument de Arg.parse est une liste de triplets de cette forme.

suite de la liste d'options

```
22      ("-sl", Arg.Float (fun f -> delay := f),
23       "précise la durée de temporisation");
24      ("-d", Arg.String (fun s -> rep := s),
25       "Le répertoire à nettoyer");
26      ("-clder",Arg.Unit (fun () -> derniers := []),
27       "supprimer les derniers caractères par défauts. Défaut: ["^
28        (List.fold_right
29           (fun s v->(Char.escaped s) ^ ";"^v ) !derniers "")
30        ^"]") ;
31      ("-lc",Arg.String (fun s -> ajoute_ref derniers s.[0]),
32       "Rajoute un dernier caractère de suffixes ,
33 ......tous les fichiers dont le suffixe termine par ce caractère
34 ......seront supprimés");
```

Notez comment on obtient la liste des valeurs de derniers

La fin des options

```
35      ("-clprem",Arg.Unit (fun () -> premiers := []),
36       " supprimer les premiers caractères par défauts. Défaut: ["^
37       (List.fold_right
38          ( fun s v->( Char.escaped s) ^";" ^ v) !premiers "")
39       ^"]") ;
40      ("-pc",Arg.String (fun s -> ajoute_ref premiers s.[0]),
41       "Rajoute un premier caractère , tous les fichiers dont le nom
42 commence par ce caractère seront supprimés");
43      ("-clsuf",Arg.Unit (fun () -> suffixes := []),
44       " supprimer les suffixes par défauts. Défaut: ["^
45       (List.fold_right ( fun s v->s  ^";"^ v ) !suffixes "")
46       ^"]") ;
47      ("-s",Arg.String ( ajoute_ref  suffixes),
48       "Rajoute un suffixe , tous les fichiers dont le nom
49 termine par ce suffixe seront supprimés");
50      ]
```

Le deuxième argument de Arg.parse

Est la fonction qui sera appliquée à tous les arguments non précédés d'une option : les arguments anonymes.

```
51      (ajoute_ref complets)
```

Le troisième argument est la documentation en cas d'erreur d'utilisation :

```
52      (Sys.argv.(0)^
53       " [options] [f1] [f2]... \n  nettoyer tous les fichiers  fi ")
```

le point d'entrée

```
55 let point_d'entrée =
56   if !Sys.interactive then () else begin
57     parsecl ();
58     Unix.handle_unix_error
59       (Outilsbalai.supprime
60          !delay !suffixes !derniers !premiers !complets !niveaux)
61       !rep;
62     exit 0
63   end
```

```
handle_unix_error f x applies f to x and returns the result.
 If the exception Unix_error is raised, it prints a message describing the error
 and exits with code 2.
```

Le Makefile pour cette appli :

```
MODULES= logiclists. borepert. outilsbalai. balai.
SOURCES=$(MODULES:.=.ml)
BCOBJS=$(MODULES:.=.cmo)
OBJS=$(MODULES:.=.cmx)

EXEC=balai
#*********Compilateurs et options de compilation
OCAMLC=ocamlc
OCAMLOPT=ocamlopt
OPTFLAGS= -thread
BCFLAGS=-thread
BCLIBS= unix.cma threads.cma
OPTLIBS= unix.cmxa threads.cmxa
INCLUDES=

exec:$(BCOBJS)
        $(OCAMLC) $(BCFLAGS) -o $(EXEC) $(BCLIBS) $(BCOBJS)

opt:$(OBJS)
        $(OCAMLOPT) $(OPTFLAGS) -o $(EXEC).opt $(OPTLIBS) $(OBJS)
```

Le Makefile pour cette appli :(suite)

```
.SUFFIXES:
.SUFFIXES: .ml .mli .cmo .cmi .cmx .mll .mly

.ml.cmo:
	$(OCAMLC) $(BCFLAGS) -c $<

.mli.cmi:
	$(OCAMLC) -c $<

.ml.cmx:
	$(OCAMLOPT) -c $(OPTFLAGS) $<

depend: $(SOURCES)
	ocamldep *.mli *.ml > .depend

clean:
	rm -f *.cm[iox] *~ .*~  *.o *.aux *.log \#*\# ;\

allclean: clean
	rm -f $(EXEC) $(EXEC).opt

alldepend:: depend

include .depend
```

Compilons puis utilisons

```
$ touch .depend
$ make depend
ocamldep *.mli *.ml > .depend
$ make exec
ocamlc -thread -c logiclists.ml
ocamlc -thread -c borepert.ml
ocamlc -thread -c outilsbalai.ml
ocamlc -thread -c balai.ml
ocamlc -thread -o balai unix.cma threads.cma logiclists.cmo borepert.cmo outilsbalai.cmo
$ make opt
ocamlopt -c -thread  borepert.ml
ocamlopt -c -thread  logiclists.ml
ocamlopt -c -thread  outilsbalai.ml
ocamlopt -c -thread  balai.ml
ocamlopt -thread -o balai.opt
  unix.cmxa threads.cmxa logiclists.cmx borepert.cmx outilsbalai.cmx balai.cmx
$ ./balai -sl 3 -r 2 -d ../.. balai.opt
Dans le répertoire: /win3/VincentUniv/Enseignements/MPLic/COURS0203/Cours06
Suppression dans 3.0 secondes des fichiers:
#balai.tex# balai.log balai.tex~ crmp06.aux crmp06.log crmp06.tex~ Makefile~ [O]/N
Dans le répertoire: /win3/VincentUniv/Enseignements/MPLic/COURS0203/Cours06/mlstuff/balai
Suppression dans 3.0 secondes des fichiers:
logiclists.cmi logiclists.cmo Makefile~ logiclists.ml~ balai.ml~ outilsbalai.ml~ borepert.ml~ balai.cmi
balai.cmo outilsbalai.cmi outilsbalai.cmo borepert.cmi borepert.cmo [O]/N
n
je n'ai rien fais
```

Bilan provisoire

Utilisation prorammation fonctionnelle et impérative $\Rightarrow$ Code lisible et court. Pas de duplication.

Utilisation des bibliothèques existantes $\Rightarrow$ Arg pour une documentation systématique de la commande.$\Rightarrow$ List pour les fonctions usuelles sur les listes.

Regroupement en fichiers par affinité $\Rightarrow$ Programmation modulaire$\Rightarrow$ Réutilisabilité, travail d'équipe,...$\Rightarrow$ En Ocaml, notion de *Module*, une valeur ou un type titi définie dans `toto.ml` a pour nom complet : Toto.titi$\Rightarrow$ Il reste à plus documenter les modules :$\Rightarrow$ Interfaces

Interfaces de module

Dans une interface : $\Rightarrow$ recopie des types$\Rightarrow$ documente les valeurs exportées

Documentation en deux parties :

Le profile des valeurs $\Rightarrow$ type

Un commentaire sur ce que fait la valeur :

Pas le comment mais le quoi.

Écriture d'interface

Automatisable : `ocamlc -c -i`

```
$ ocamlc -c -i logiclists.ml
val ou_de : ('a -> bool) list -> 'a -> bool
val au_moins_une : ('a -> bool) list -> 'a list -> 'a list
val et_de : ('a -> bool) list -> 'a -> bool
val toutes : ('a -> bool) list -> 'a list -> 'a list
```

Mettre dans un fichier de nom adéquat .mli :

```
$ ocamlc -c -i logiclists.ml > tmp.mli ; mv tmp.mli logiclis
```

Édition et écriture des commentaires

logiclists.mli

```
(*logiclists.mli*)

(** Disjonction et conjonction de propriétés
 @author Vincent Poirriez
 @since 10 octobre 2002
*)

(** Disjonction de propriétés.
    {!Logiclists.ou_de}[lprop x]
    @return la disjonction des propriétés de [lprop] appliquées à [x
*)
val ou_de : ('a -> bool) list -> 'a -> bool
```

Ne pas exporter certaines valeurs

Il suffit de ne pas les mentionner dans l'interface :

outilsbalai.mli

```
(*outilsbalai.mli*)

(** Pour supprimer des fichiers d'une arborescence *)

(** [temporise action defaut delay ch] @return le résultat de
[action valeur] où [valeur] est lue sur le canal [ch] avant le temps
[delay] secondes. Si aucune valeur n'est lisible sur [ch] avant [del
alors [defaut] est utilisé pour [valeur].
*)
val temporise : (string -> 'a) -> string -> float -> in_channel -> '
```

Interface et travail d'équipe

Interface=contrat :Se mettre d'accord sur l'interface Si A dépend de B, se mettre d'accord sur b.mli et implanter en parallèle a.ml et b.ml

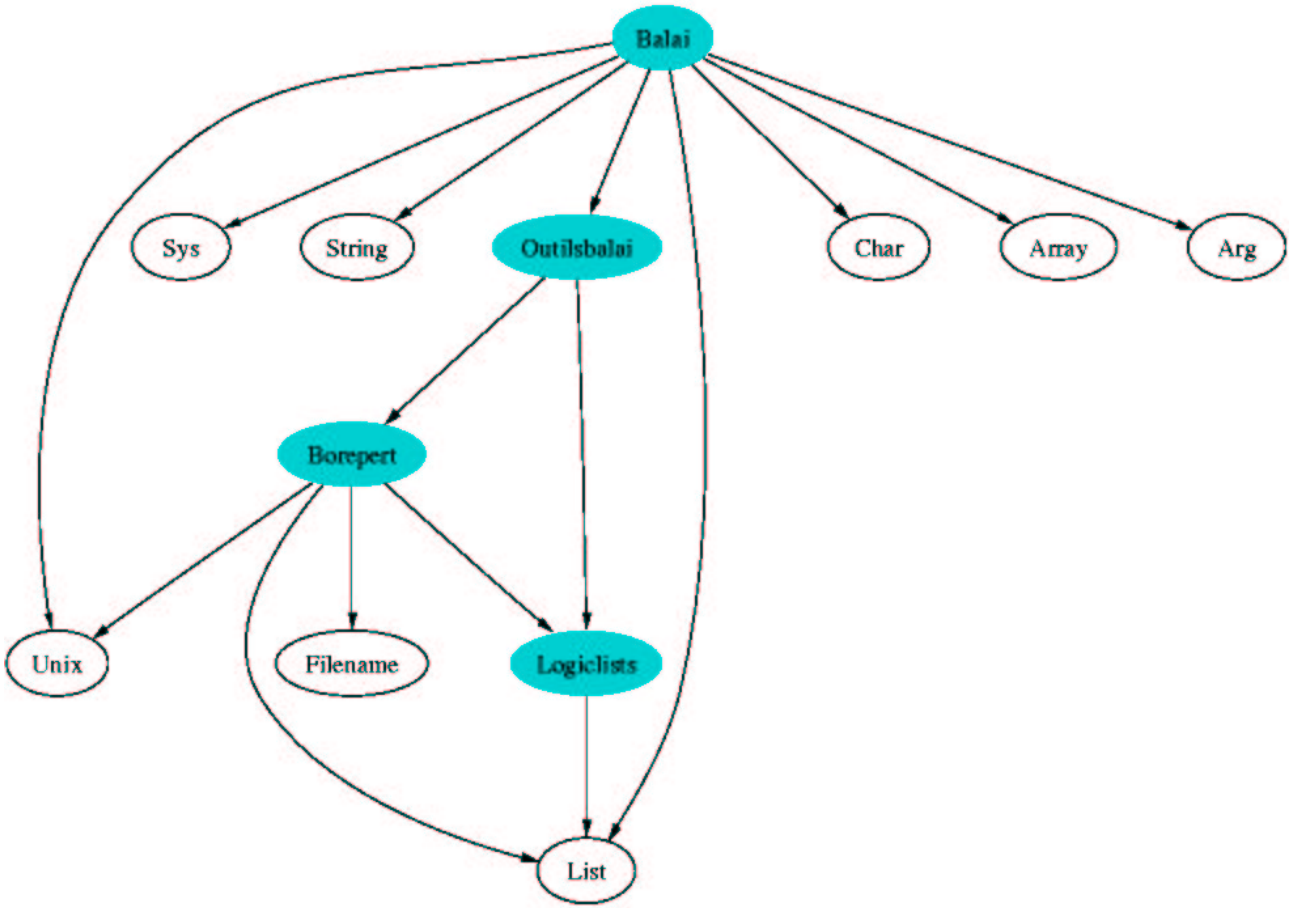
Dépendre de l'interface, pas de l'implantation :Si l'implantation est modifiée mais respecte l'interface $\Rightarrow$

pas besoin de changer a.ml

Le graphe de dépendance de l'application

Obtenu par la ligne de commande

```
$ ocamldoc  -dot -dot-include-all *.ml *.mli
$ dot -Tps ocamldoc.out -o balaigr.ps
```



Un tri externe

Le problème du tri externe

Les données à trier sont :

– présentes dans des fichiers : il peut y avoir plusieurs fichiers à *fusionner*

– trop nombreuses pour toutes résider en mémoire

– il faut obtenir un fichier contenant toutes les données triées.

Le problème est :

– Indépendant des valeurs à trier

– Indépendant de l’ordre à utiliser

Les idées de l'algorithme

- Créer des fichiers intermédiaires, triés, ces fichiers s'appellent des monotonies.
- Garantir qu'à tout moment, le volume de données en mémoire est inférieur à la limite fixée `nbmax`
- maintenir en mémoire deux listes de valeurs lues mais pas encore écrites :
  - une liste des valeurs qui sont plus petites que la dernière écrite,
    ces valeurs seront écrites dans la monotonie en cours d'écriture
  - une liste des valeurs plus grandes que la dernière écrite,
    ces valeurs seront écrites dans la prochaine monotonie

  Ces deux listes sont maintenues triées.
- Le noyau : lire une valeur, la ranger dans la bonne liste, écrire la plus grande valeur de la liste des valeurs à écrire si celle ci n'est pas vide, sinon :fermer la monotonie, en ouvrir une nouvelle, et continuer.
- Quand toutes les valeurs sont lues et écrites dans les monotonies, il reste à fusionner les monotonies.

Écrire et lire les valeurs dans les fichiers

– Nécessité de cohérence
– Dépend des valeurs traitées

L'interface à respecter :

io.mli

```
(** le type des données lues et écrites *)
type t

(** la fonction de lecture *)
val lire : in_channel -> t

(** la fonction d'écriture *)
val écrire : out_channel -> t -> unit

(** la fonction de comparaison *)
val compare : t -> t -> int
```

Modélisons l'état courant de l'algorithme

Utilisons un type enregistrement paramètré par le type des valeurs :

cremonot.ml

```
9  type 'a état =
10     { monotonies_créées : string list ; (**les fichiers déjà créés*)
11       source : in_channel; (** le fichier source actuel*)
12       dest : out_channel ; (**la monotonie en cours d'écriture*)
13       dernier_rangé : 'a ; (** la dernière valeur écrite dans [dest]
14       àranger : 'a list ; (**Les valeurs à ranger, liste triée*)
15       marqués : 'a list ; (** Les valeurs plus "grandes" que
16                              la dernière écrite dans [dest].
17                              C'est aussi une liste triée*)
18     }
```

Un invariant d'état est : *la taille de àranger@marqués est constante*. Sauf dans le cas où il y a moins d'éléments à trier que d'éléments possibles dans àranger et marqués. Dans ce cas, on fait un tri interne et pas externe.

L’action de base de l’algorithme

```
25  let ( < ) a b = (Io.compare a b) < 0 (*Pour faciliter l'écriture*)

28  let lit_et_range état =
29    let suivante = Io.lire état.source in (* lire la valeur*)
30    let nouvel_état = match état.àranger with (*mettre à sa place*)
31    | max::suite ->
32        if suivante < état.dernier_rangé then
33          {état with
34           dernier_rangé = max;
35           àranger = suite;
36           marqués = insert Io.compare suivante état.marqués}
37        else if suivante < max then {état with dernier_rangé = suivant
38        else { état with
39               dernier_rangé = max;
40               àranger = insert Io.compare suivante suite}
41    | [] -> assert(false)
```

L’action de base de l’algorithme(fin)

```
42    in
43    Io.écrire état.dest nouvel_état.dernier_rangé; (*écrire*)
44    nouvel_état                          (*renvoyer le nouvel état *)
```

On lit avant d’écrire pour préserver l’invariant d’état car si le fichier en lecture est vidé l’exception End_of_file est déclenchée avant d’écrire.

Lire toutes les valeurs d'un fichier

La fonction prend un argument un générateur de nom pour les monotonies.

```
55  let rec les_monotonies_d'un_fichier gen_nom état= try
56    let nouv_état = match état.àranger with
57    |[] -> (* tous les éléments ont été lus après un élément
58              "plus grand" qu'eux, ils sont donc dans [état.marqués].
59              Il faut continuer avec une nouvelle monotonie.*)
60        begin
61          close_out état.dest;
62          let nouv = gen_nom () in
63          {état with
64           monotonies_créées = nouv::état.monotonies_créées;
65           dest = open_out_bin nouv ;
66           àranger = état.marqués;
67           marqués = []}
68        end
```

Lire toutes les valeurs d'un fichier(fin)

```
69    | _ -> lit_et_range état
70    in
71    les_monotonies_d'un_fichier gen_nom nouv_état
72  with End_of_file -> begin close_in état.source; état end
```

état vérifie l'invariant

gen_nom est une fonction générateur de noms tous distincts

les_monotonies_d'un_fichier gen_nom état lit toutes les valeurs présentes sur le canal état.source et les écrits dans des monotonies.

renvoie un nouvel état nétat tel que nétat.source est fermé.

```
78  let ouvre_et_vide_le_fichier gen_nom f état =
79    let nouvétat = {état with source = open_in_bin f} in
80    les_monotonies_d'un_fichier gen_nom nouvétat
```

Parcourir une liste de fichiers

Quand un état vérifiant l'invariant a été construit :

```
86 let parcours gen_nom fichiers état =
87   List.fold_right (ouvre_et_vide_le_fichier gen_nom) fichiers état
```

Construire un état vérifiant l'invariant (lire les premiers), il faut :

nb_maxi le nombre d'éléments à lire, dépend de la taille de la mémoire.

lus maintenir la liste des éléments déjà lus

src un canal ouvert en lecture dans lequel sera lu l'élément suivant

fichiers une liste de fichiers que l'on peut ouvrir si il n'y a plus assez d'éléments dans [src]

Lire les premiers

```
101 let rec lire_les_premiers nb_maxi lus src fichiers = try
102   match (nb_maxi, Io.lire src) with
103   | (1, suivant) -> (0, suivant :: lus, src, fichiers)
104   | (_, suivant) ->
105       lire_les_premiers (nb_maxi - 1) (suivant :: lus) src fichiers
106 with End_of_file ->
107   (*Dans le cas où il y a moins de valeurs dans [src] que [nb_maxi]
108   match fichiers with
109   | [] -> (nb_maxi, lus, src, [])
110   | f::suite -> begin
111       close_in src;
112       lire_les_premiers nb_maxi lus (open_in_bin f) suite
113   end
```

Construire le premier état

Une exception pour un cas exceptionnel :

```
115 exception Pas_de_valeur_à_trier
```

```
124 let premier_état nb_max gen_nom fichiers =
125   let (nb_lus, lus, ouvert, frestants) =
126     match fichiers with
127     | f :: suite ->
128         lire_les_premiers nb_max [] (open_in_bin f) suite
129     | [] -> invalid_arg "Triext.premier_état: pas de fichier à ouvri
130   in
131   let monot1 = gen_nom () in
132   match List.sort Io.compare lus with
```

Construire le premier état(fin)

```
133   | max :: suite -> begin
134       let ch_dest = open_out_bin monot1 in
135       Io.écrire ch_dest max;
136       ({ monotonies_créées = [monot1];
137          source = ouvert;
138          dest = ch_dest;
139          dernier_rangé = max;
140          àranger = suite;
141          marqués = []}, frestants)
142   end
143   | _ -> raise Pas_de_valeur_à_trier
```

Vider les derniers éléments

Quand tous les éléments ont été lus dans les fichiers, il reste à vider ceux qui restent dans les listes état.marqués et état.àranger. On renvoie la liste des monotonies créées.

```
149 let vider_les_derniers gen_nom état =
150   List.iter (Io.écrire état.dest) état.àranger;
151   close_out état.dest;
152   if état.marqués = [] then état.monotonies_créées
153   else begin
154     let nouv = gen_nom () in
155     let dest = open_out_bin nouv in
156     List.iter (Io.écrire dest) état.marqués;
157     close_out dest;
158     nouv::état.monotonies_créées
159   end
```

Créer les monotonies à partir des fichiers

En entrée : le répertoire où seront écrites les monotonies ;
le nom (sans extension) des monotonies ;
le nombre maximal d'éléments en mémoire ;
et les fichiers source.

```
169 let créer_les_monotonies destination nom nb_max fichiers =
170   let gen_nom = Botriext.gensym (destination^nom) in
171   let état, frestants = premier_état nb_max gen_nom fichiers in
172   let dernierétat = parcours gen_nom frestants état in
173   vider_les_derniers gen_nom dernierétat
```

Suppose qu'il y a une fonction gensym dans le module Botriext.

un générateur de symboles

Dans le fichier `botriext.ml`

```
let gensym base =
  let cpt = ref (-1) in
  fun () -> begin
```

cpt est une variable rémanente.

```
    incr cpt;
    let ext =
      if !cpt >= 100 then string_of_int !cpt else
      if !cpt >= 10 then "0"^(string_of_int !cpt)
      else "00"^(string_of_int !cpt)
    in base^"."^ext
  end
```

Fusionner les monotonies

C’est un problème en soit

Des fichiers contenant des données triées doivent être fusionnés dans un seul fichier trié.

Récupérer la première valeur d’un fichier et un canal ouvert en lecture sur le fichier :

Tramonot

```
16 let première_valeur_et_descripteur f =
17   let ch = open_in_bin f in
18   let v = Io.lire ch in
19   (v,ch)
```

Le faire pour tous les fichiers et récupérer la liste :

```
27 let valeurs_et_descripteurs monotonies =
28   List.map première_valeur_et_descripteur monotonies
```

La fusion

La liste des couples (premier,descripteur) est triée par ordre de valeur.

```
33 let compare_couples (v1,c1) (v2,c2) = Io.compare v1 v2
```

```
43 let rec fusion dest val_et_src = match val_et_src with
44 | [] -> close_out dest  (* tout à été fusionner *)
45 | (min,monot)::suite -> begin
46     Io.écrire dest min;
47     try
48       let v = Io.lire monot in
49       fusion dest (insert compare_couples (v,monot) suite)
50     with End_of_file ->
51       begin
52         close_in monot; fusion dest suite
53       end
54 end
```

La fonction à exporter

Il faut trier la liste des couples (premier,descripteur), ouvrir en écriture un canal sur `destination`, appeler `fusion`.

```
62 let fusionner destination monotonies =
63  fusion
64     (open_out destination)
65     (List.sort compare_couples (valeurs_et_descripteurs monotonies))
```

La fonction de tri externe

Les arguments sont : le nombre maximum d'éléments en mémoire, le nom des fichiers sources, le nom du fichier destination, le nom du répertoire où écrire les monotonies, le nom de base pour les monotonies.

Triext

```
12 let tri_externe nb_max sources destination rep_monot base_monot=
13   if sources = [] then [] else begin
14     let monotonies =
15       créer_les_monotonies rep_monot base_monot nb_max sources
16     in
17     fusionner destination monotonies;
18     monotonies
19   end
```

Les valeurs par défauts

```
23 let efface_monotonies = ref true
24 let dest = ref "sortedval.res"
25 let sources = ref ([] : string list)
26 let base = ref "monotonie"
27 let rep = ref "/tmp/"
28 let nb_maxi = ref 10
```

Analyse de la ligne de commande

```
31 let parsecl () =
32   Arg.parse
33     [("-msiz", Arg.Int(fun i -> nb_maxi := i),
34       "le nombre maximal de valeurs en mémoire. default:["^
35       (string_of_int !nb_maxi) ^"]");
36      ("-basem", Arg.String(fun s -> base := s),
37       "la base utilisée pour les fichiers de monotonies. default["
38       ^ !base ^ "]");
39      ("-tmp", Arg.String(fun s -> rep := s),
40       "le répertoire où seront écrites les monotonies, default = ["
41       ^ !rep ^ "]");
42      ("-res", Arg.String(fun s -> dest := s),
43       "le nom du fichier résultat, default = ["^ !dest ^ "]");
44      ("-keepm", Arg.Clear efface_monotonies,
45       "Pour garder les monotonies")]
```

Analyse de la ligne de commande(fin), point d'entrée

```
46      (fun s -> sources := s :: !sources)
47      ("Usage: "^Sys.argv.(0)^" [options] liste des fichiers sources")
```

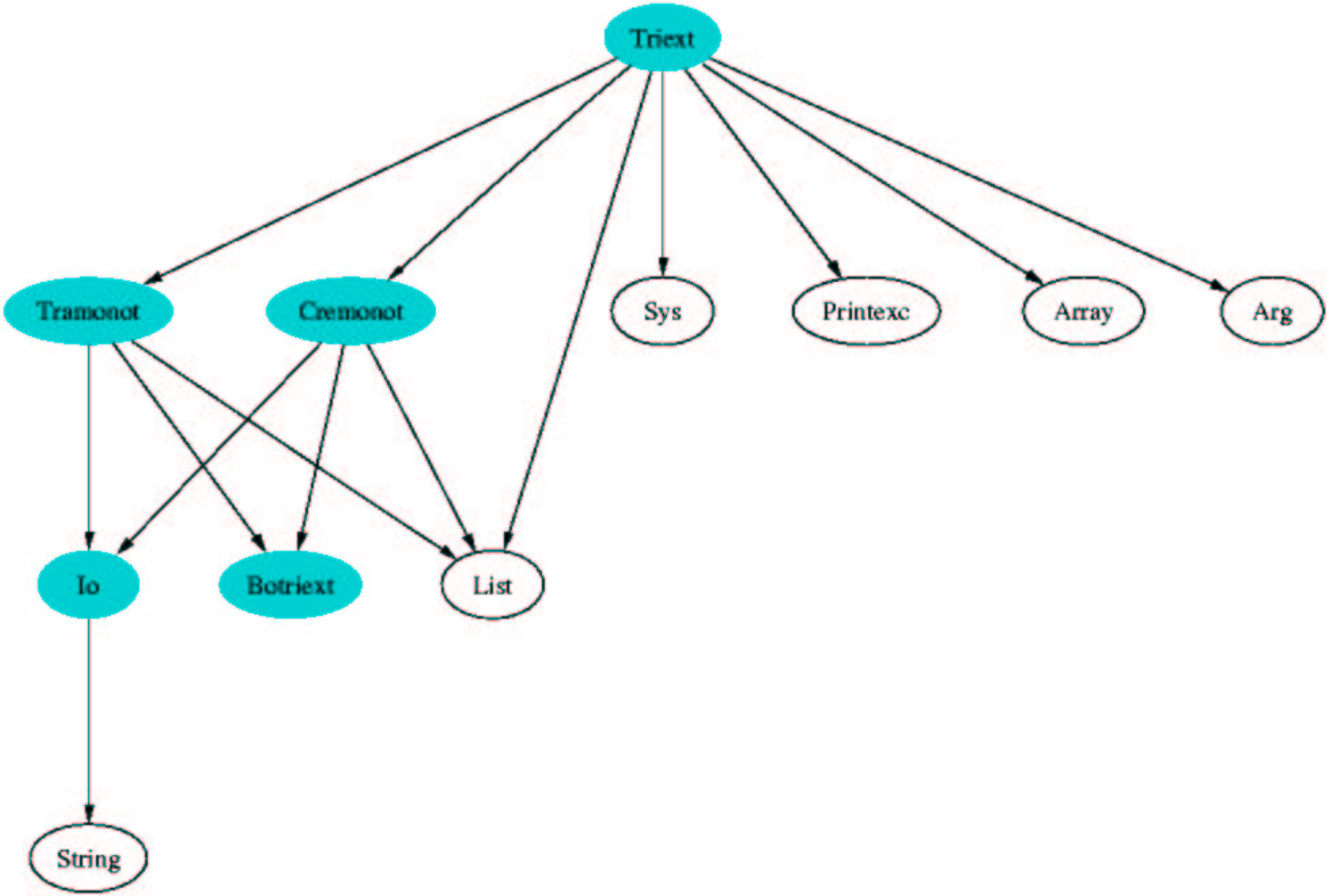
```
50  let main () =
51    if !Sys.interactive  then () else begin
52      parsecl();
53          print_endline "tri en cours ... ";
54      let monotonies = tri_externe !nb_maxi !sources !dest !rep !base
55      if !efface_monotonies then
56        List.iter Sys.remove monotonies;
57      print_endline ("tri effectué, résultat dans : " ^ !dest);
58      exit 0
59    end
```

```
61  let point_d'entrée = Printexc.catch main ()
```

Graphe de dépendance de l'application

Obtenu par : `$ocamldoc -dot *.ml -dot-include-all`

```
$ dot -Tps ocamldoc.out -o triextgr.ps
```



Makefile qui va bien

Le début, pour le reste pas de changement.

```
MODULES= botriext. io. cremonot. tramonot. triext.
SOURCES=$(MODULES:.=.ml)
BCOBJS=$(MODULES:.=.cmo)
OBJS=$(MODULES:.=.cmx)

EXEC=triext
#*********Compilateurs et options de compilation
OCAMLC=ocamlc
OCAMLOPT=ocamlopt
OPTFLAGS=
BCFLAGS=
BCLIBS=
OPTLIBS=
INCLUDES=
```