Documentation for `ocamaweb.ml`

extracted by `mldoc`

November 27, 2002

Author : *Charles-Albert Lehalle*
charles.lehalle@miriadtech.com
Affiliation : MIRIAD Technologies,
8 av Hoche, 75008 PARIS - FRANCE.

1 OCAMAWEB

parsing functions for converting a MATLAB code into a LaTeX one (literate programming style in MATLAB).

pour générer ce document :

```
  set PATH=%PATH%;F:/Programs/OCaml/lib
  ../../caldoc -o ocamaweb.tex -key "<" -main -standalone -notitle ocamaweb.ml
  caldoc  is a modified version of  mldoc.
```

```
@author  charles-albert lehalle
@date    feb 2002
@last    may 2002
```

```
let version = "5.2";;

(*> *)
```

```
@histo
@version 5.2 - mots clef matlab dans ocamaweb.xml
@version 5.1 - la macro definition est remplacée par ocamawebdefinition (pour JL)
@version 5.0 - recours à ocamaweb.xml
@version 4.43 - correction d'un bug de typo (merci JB)
@version 4.42 - autorisation des apostrophes dans les champs spéciaux (title, etc)
@version 4.41 - correction d'un bug issu de 4.4 : ordre de traitement des fichiers
(merci JL)
```

```
@version 4.4 - suppression du chemin du fichier traité (pour JFL)
@version 4.3 - correction du bug de starrification du premier niveau de titre
@version 4.1-2 - corrections TeX pour Jérôme L.
@version 4.0 - correction de divers "bugs" (défault de special keys)
@version 3.9 - correction de l'ordre des "latexifications" et "pdfications"
             - gestion plus robuste des variables d'environnement
@version 3.8 - améliorations de mise en page (vers ocamaweb.sty et directement dans
le code)
@version 3.7 - inclusion du chemin jusqu'à ocamaweb.sty dans le fichier LaTeX
@version 3.6 - version avec un seul argument en ligne de commande :
               envoi des sorties dans le répertoire OCAMAWEB_DEST
@version 3.5 - version avec compilation LaTeX incluse
@version 3.4 - version compilée
@version 3.3 - édition d'une ligne "utilise" / "est utilisé par" en bas de chaque
section
@version 3.2 - mise à jour de ocamaweb.sty
             - disponibilité de l'"étoilage" des sections
@version 3.1 - système de récupération de "clefs globales" (section 3.2)
@version 3.0 - externalisation de la feuille de style LaTeX
             - numérotation des blocs
@version 2.2 - mise au format de mldoc recompilé par mes soins (option key ajoutée)
             - problème: je ne récupère pas les % des commentaires de la section de
code
             - attention: je dois gérer mes propres backslash en LaTeX
@version 2.1 - utilisation d'une variable d'environnement
             - "problème notable" de 1.0 contourné : les fils de 1er niveau sont
inversés
@version 2.0 - résolution d'un problème de parsing de la première ligne
@version 1.0 - problème notable : enpilage en sens "inverse" des sections de haut
niveau
             - attention à l'évaluation récursive des types
             - choix définitif de la structure d'arbre
```

Macros LaTeX à développer

.

1. qui référencie un bloc (avec une `ref{#1}` dedans)
2. qui positionne un titre (avec un `label{#1}` dedans)
3. qui positionne un commentaire (pour ceux isolés sur une ligne)
4. un backslash
5. un string

6. le symbol d'avant le code
7. le symbol d'avant la section
8. pour pointer sur les sections utilisées et / où utilisant la version en cours
9. appelée en début de fichier
10. appelée en fin de fichier

La feuille de style ocamaweb.sty doit être dans le PATH $\LaTeX$. D'autre part, les section doivent être référencées par `{refcode:#}`. Elle contient en outre diverses définition correspondant aux opérateurs (=, ¬, etc).

TODO list:

- si il n'y a pas de titre de section : plantage String.sub
- au début du fichier, les tout est mis dans le code tant que le premier "%%" n'est pas rencontré (problème des commentaires de code?)
- il faut vérifier les 'String.sub problems' dans les logs
- impossible d'utiliser les 'pipes' dans un titre
- commentaires après du code (regexp)
- incorporer les listes de pères et fils dans l'arbre
- "morceler" le code :
    - un objet pour le parsing : un objet "abstrait" qui soit indépendant du type de commentaire (par lignes ou par blocs) + une implémentation pour MATLAB
    - un objet pour le pretty printing avec une implémentation pour $\LaTeX$
    - externaliser certains paramétrages dans un `.ini` : clefs spéciales, remplacements de symbols, etc.
- améliorer le traitement des chaînes de caractères (pour $\LaTeX$)
- léger problème : les remplacements sont effectués dans les chaînes de charactères (par exemple les = deviennent des $\longleftarrow$)

Pour compiler:

```
set OCAMLLIB=%OCAML_HOME%/lib
set PATH=%OCAML_HOME%/bin;%PATH%
%OCAML_HOME%/bin/ocamlc Str.cma -o ocamaweb.exe ocamaweb.ml

ocamaweb sinuso.m first.tex
```

Table of Contents

2 Small guide

This section contains a small guide to OCAMAWEB, a software in ocaml allowing "literate programming"[1] for MATLAB. This software can be easily generalized to other programming languages. Its goal is to become a generic literate programming tool, called OCAXXWEB.

2.1 Literate programming

This documentation is produced my a modified version of mldoc which is a literate programming tool for ocaml.
The principle of literate programming is to allow inclusion of comments into software sources so that the given sources can be compiled to produce a clear documentation for the software.

The produced documentation is a technical one (explaining the code and the algorithms used) but can include some informations to deliver to the final user of the software.
The key concept in literate programming is the use of "sections" of code.

Section of code.

A section of code is a given part of a software code with a title, and an explanation of the features of the code. Sections can be imbricated.
The power of the literate programming tools is that the order the section into the documentation produced has not to be their order into the code.

Example.

In MATLAB the comments are marked with the character % :

```
function [M, v, l] = a_lot_of_PCAs(n, p, nb)
%< checking the argument number
% there can be 2 or 3 arguments
if nargin < 3
 nb = 1
end;
%>

figure; hold on;
%< performing ACPs
% the number of PCAs to be preformed is nb
for i=1:nb
 %< randomized input
 % I will perform a PCA on a gaussian n X p matrix
 M = randn(n,p);
 %>
 %< PCA
 % I use eig but I could used SVD too
 [v,l] = eig(M' * M);
 l = diag(l);
 csl = cumsum(l);
 %>
 plot(csl/csl(end));
end;
%>
```

Here we have 4 sections :

```
SECTION 1 :=
%< PCA
% I use eig but I could used SVD too
[v,l] = eig(M' * M);
l = diag(l);
csl = cumsum(l);
%>

SECTION 2 :=
%< randomized input
% I will perform a PCA on a gaussian n X p matrix
M = randn(n,p);
%>

SECTION 3 :=
%< performing ACPs
% the number of PCAs to be preformed is nb
for i=1:nb
<<SECTION 2>>
<<SECTION 1>>
plot(csl/csl(end));
end;
%>

SECTION 4 :=
%< checking the argument number
% there can be 2 or 3 arguments
if nargin < 3
nb = 1
end;
%>
```

And the code :

```
function [M, v, l] = a_lot_of_PCAs(n, p, nb)
<<SECTION 4>>
figure; hold on;
<<SECTION 3>>
```

As one can see, if you replace the section by their titles, the code is very simple :

```
function [M, v, l] = a_lot_of_PCAs(n, p, nb)
<<Checking the argument number (SECTION 4)>>
figure; hold on;
<<Performing ACPs (SECTION 3)>>
```

A literate programming software allows to reorganize your code in such a more "readable" way.

2.2 OCAMAWEB syntax

The OCAMAWEB syntax is inspired of the CWEB one. The comment character for MATLAB is "%", so all the lines adressed to OCAMAWEB will begin by a "%".

Section definition.

The sections will began with "%%" or "%<". The sections begining with "%%" will be called "*global sections*" and the section begining with "%<" will be called "*arbitrary sections*". The strings "%%" and "%<" are called "*section marks*".

Global sections.

Global sections begin with "%%" and end with the next "%%" or the end of the file.

Arbitrary sections.

Arbitrary sections begin with "%<" and end with "%>". Arbitrary sections can be nested.

Section title.

The title of a section is the text between the section mark and the first "`.`" (dot) or the end of the line :

source	title
`%% alpha beta. gamma`	alpha beta
`%% alpha beta`	alpha beta
`%< alpha beta. gamma`	alpha beta
`%< alpha beta`	alpha beta

Section captions.

The section captions will be printed in LATEX ; the section comment is :

- the text after the first "`.`" (dot) on the line of section mark,
- each comment line after the line of the section mark,
- each comment line after a caption line.

For instance :

code	caption
`%% alpha beta. gamma`	gamma
`% blah blah`	blah blah
	<empty line : end of this section caption>
`% blah blah`	*<nothing : this is regular comments>*

or:

code	caption
`%< alpha beta. gamma`	gamma
`% blah blah`	blah blah
`for i=1:n`	*<end of this section caption>*
`% blah blah`	*<nothing : this is regular comments>*

Section code

All the source code between the end of the section caption and the end of the section are in the "*code*" part of the section. If an section is defined into a code part then only a reference to this section will be written and the complete section will be put after the current one.

Levelized sections.

Any section can be "*levelized*" : that means that the section mark will contain a special character like "`*N`" (where `N` is a number) or "`**`".
Section with "`**`" will have an higher level than classical ones, and section with "`*N`" will have a "`N`" times higher level than "`**`" ones.
The more a section has an high level, the more it is important.

2.3 OCAMAWEB LATEX macro

Some LATEX macro are defined into tha `ocamaweb.sty` file that is included for compilation of outputs of OCAMAWEB. They can be used in the comment part of the source files.

As with CWEB, you can use the macro 'pipe' () to "quote" some code into your comments, but actually the "piping" is not allowed into the section titles.

Wringing: do not use tabbing into your code but only spaces if you want to have a pretty print.

2.4 Install and use

To install OCAMAWEB, just put the `ocamaweb.exe`, `ocamlrun.exe` and `dllstr.dll` in your `PATH`, and define two variables : OCAMAWEB to specify where the `ocamaweb.log` file (the log file) has to been put and where the `ocamaweb.sty` file is, and OCAMAWEB_DEST to specify where the transformed files have to been put by default. Those two variables contain directory names and must end by the file separator (\ for Windows and / for Unixes).
The use of OCAMAWEB is : `ocamaweb matlab_file.m [destination_file.tex]`. if the destination file is ommitted, it is named after the input file (`matlab_file.tex`) and put into the OCAMAWEB_DEST directory **annd** in this case the `.tex` file is produced and processed with $\LaTeX$ and dvipdfm.

When the second argument is specified, the `.tex` file is produced and nothing else is done.

2.5 PAramertization of OCAMAWEB

The versions 5.0 and after of OCAMAWEB allow its parametrization of the software using an xml file : `ocamaweb.xml`. It allow to change the tags used by OCAMAWEB to recognize keywors as version, author's name and email, etc.

Another section of the xml file if dedicated to the declaration of MATLAB keywords (that are sent to the ocamawebdefinition LaTeX macro).

```
<ocamaweb>
<keypatterns>
   <keypattern name="author" key="node.author" before="'" after="'"/>
   <keypattern name="author" key="% author" before="'" after="'"/>
   <keypattern name="author" key="% auteur" before="'" after="'"/>
   <keypattern name="title" key="% title" before="'" after="'"/>
   <keypattern name="title" key="% titre" before="'" after="'"/>
   <keypattern name="project" key="% project" before="'" after="'"/>
   <keypattern name="project" key="% projet" before="'" after="'"/>
   <keypattern name="mailto" key="node.mailto" before="'" after="'"/>
   <keypattern name="mailto" key="% mailto" before="'" after="'"/>
   <keypattern name="date" key="node.date" before="'" after="'"/>
   <keypattern name="date" key="% date" before="'" after="'"/>
   <keypattern name="version" key="% version" before="'" after="'"/>
   <keypattern name="version" key="VERSION" before="=" after=";"/>
</keypatterns>
<matlabwords>
   <keyword name="function"/>
   <keyword name="while"/>
   <keyword name="continue"/>
   <keyword name="try"/>
   <keyword name="catch"/>
   <keyword name="for"/>
   <keyword name="end"/>
```

```
        <keyword name="persistent"/>
        <keyword name="switch"/>
        <keyword name="case"/>
        <keyword name="if"/>
        <keyword name="else"/>
        <keyword name="elseif"/>
        <keyword name="return"/>
        <keyword name="break"/>
        <keyword name="otherwise"/>
    </matlabwords>
    <info/>
    </ocamaweb>
```

That means for instance that the author's name it recongnized as :

- anything on the same line as 'node.author', between ' and '
- or anything on the same line as '% author', between ' and '
- or anything on the same line as '% auteur' (french version), between ' and '

```
open Str;;
open Xml;;

(*> *)
```

3 Fonction globales et définitions

La variable d'environnement OCAMAWEB est utilisée comme chemin pour le fichier de log (son nom est `"ocamaweb.log"`) et pour le fichier `ocamaweb.sty`.

La variable d'environnement OCAMAWEB_DEST est utilisée comme repository de fichiers de doc.

Attention: j'ai besoin du slash de fin!

```
let log_file =
  try
```

```
      open_out ((Sys.getenv "OCAMAWEB") ^ "ocamaweb.log")
    with Not_found ->
      Printf.fprintf stderr "OCAMAWEB: WARNING! WARNING! WARNING!\n          environment variable
'OCAMAWEB' not defined!\n";
      open_out "ocamaweb.log"
;;

(* to be used as output for undefined special keys *)
let not_defined = "***NOT DEFINED***";;

let root_string = "***ROOT***";;

let empty_string = "***EMPTY***";;

let relations     = Hashtbl.create 20;;

let sub_relations = Hashtbl.create 20;;
```

3.1 parsing d'une ligne

Lors du parcours du fichier de CODE, chaque ligne va se voir attribuer un type [state].
Les différents éléments de ce type sont :

- `Into_comments` : il s'agit d'une ligne de commentaire "classique" (elle fera partie soit du code soit des commentaires du bloc suivant sa position relative p/r aux autres lignes)
- `Begin_section` : il s'agit de la déclaration d'une section (par (de cette ligne on va tirer le titre -d'où la référence sera déduite- et le début des commentaires de bloc)
- `Begin_subsection` : il s'agit d'une section incluse (par (de cette ligne on va tirer le titre -d'où la référence sera déduite- et le début des commentaires de bloc)
- `End_subsection` : il s'agit d'une ligne de code
- `Begin_file` : c'est l'état en début de fichier
- `Unknown` : c'est un état piégeant destiné aux erreurs

```
type state = Into_comments | Begin_section | Begin_subsection |
  End_subsection | Into_code | Begin_file | Unknown;;

type action = Add_comment | Add_code | Add_comments_into_code | New_section |
  New_subsection | Into_first_section | Close_subsection | Stop;;

(* used to attribute a level to sections *)
```

```
type state_level = Not_relevant | Star | Double_star | Level of string;;

(* j'ai le titre X le nombre d'espaces X un code de référence (numéro d'ordre) *)
type indented_ref = Indented_ref of string * int * int;;

(* quelques utilitaires pour clarifier le code *)
let next1 s   =
  try
    (String.sub s 1 ((String.length s) - 1))
  with Invalid_argument( ia_s) -> Printf.fprintf log_file "next1: String.sub problem on %s\n" s;
    (* flush log_file; *)
    ""
;;

let next2 s   =
  try
    (String.sub s 2 ((String.length s) - 2))
  with Invalid_argument( ia_s) -> Printf.fprintf log_file "next2: String.sub problem on %s\n" s;
    (* flush log_file; *)
    ""
;;

let char1 s   =
  try
    (String.sub s 0 1)
  with Invalid_argument( ia_s) -> Printf.fprintf log_file "char1: String.sub problem on %s\n" s;
    (* flush log_file; *)
    ""
;;

let char1_c s = (String.get s 0);;

let char2 s   =
  try
     (String.sub s 0 2)
  with Invalid_argument( ia_s) -> Printf.fprintf log_file "char2: String.sub problem on %s\n" s;
    (* flush log_file; *)
    ""
;;
```

3.2 Récupération de variables globales

Les mots clefs suivants sont associés à des variables "globales" qui seront utilisées pour l'en-tête de la documentation.

clef	déclencheur	début	fin
`author`	`node.author`	'	'
	`% author`	'	'

	% auteur	'	'
title	% titre	'	'
	% title	'	'
project	% project	'	'
	% projet	'	'
mailto	node.mailto	'	'
	% mailto	'	'
date	node.date	'	'
	% date	'	'
version	VERSION	=	;
	% version	'	'

Je vais utiliser la structure suivante :

- une hashtable qui contient une description de chaque association de clef (cf tableau plus haut) avec un entier comme index. Une telle description est une structure de type `key_pattern`, qui contient comme champs :
  - `name_key` : le nom de la clef.
  - `trigger_pat` : le "déclencheur"
  - `before_pat` : le tag de début
  - `after_pat` : le tag de fin
  - `is_found` : un booléen qui signale si cette clef a déjà été trouvée dans le texte
  - `value` : la valeur de la clef

  Pour résumer, la clef `name_key` prend sa valeur dans la zone crochetée du pattern :
  `... trigger_pat ... before_pat [ ... ] after_pat`
- un type `Line( string, bool, int)` qui permet de décrire l'état de la ligne en cours de recherche de clef.
  `Line( S, B, K)` décrit la ligne de contenu S dans laquelle une clef a déjà été trouvée si B est à *true*, et ne l'a pas si B est à *false*. Lorsque B vaut *false* : K vaut -1, et sinon il vaut le numéro de la clef trouvée.

```
type key_pattern = { name_key    : string;
                     trigger_pat : string;
                     before_pat  : string;
                     after_pat   : string;
                     mutable is_found    : bool;
                     mutable value       : string;
                   } ;;

let build_key_pattern name trigger before after =
  { name_key    = name;
    trigger_pat = trigger;
    before_pat  = before;
    after_pat   = after;
    is_found    = false;
    value       = "";
```

```
          } ;;

        let specify_key_pattern name value =
          { name_key    = name;
            trigger_pat = not_defined;
            before_pat  = not_defined;
            after_pat   = not_defined;
            is_found    = true;
            value       = value;
          } ;;

        let keys_counter = ref 0;;
        let keys_hashtbl = Hashtbl.create 20;;

        let add_key_pattern name trigger before after =
          Hashtbl.add keys_hashtbl !keys_counter
            (build_key_pattern name trigger before after);
          keys_counter := !keys_counter + 1
        ;;

        let add_specified_key_pattern name value =
          Hashtbl.add keys_hashtbl !keys_counter
            (specify_key_pattern name value);
          keys_counter := !keys_counter + 1
        ;;

        (* one line is the line string X a boolean (found or not) X the key into the hashtable
           a "neutral  line" is (string, false, -1)
           a "resolved line" is (string, true, nb) with nb a key in keys_hashtbl
           *)
        type line_state_for_keys = Line of string * bool * int;;
```

3.2.1 Recherche de sous chaînes de charactères

Je n'ai pas trouvé en ocaml de fonction `String.index_substring str1 str2` qui renvoie la position de `str2` dans `str1`.

J'ai donc implémenté les fonctions `compare_strings` et `contains_substring` qui émulent cette fonctionnalité. J'ai recours pour cela à `contains_substring_rec` qui cherche récursivement la position d'une chaîne dans un string.

```
        let compare_strings str1 str2 =
          let len = min (String.length str1) (String.length str2) in
            try
              String.sub str1 0 len = String.sub str2 0 len
            with Invalid_argument( ia_s) -> Printf.fprintf log_file "compare_...: String.sub problem on
        %s\n" str1;
```

```
          (* flush log_file; *)
          false
;;

(* use: contains_substring str substr (String.length str) (String. length substr) 0
   return the index of the end of substr into str, raises Not_found if not found *)
let rec contains_substring_rec str sub_str l sl n =
  if char1 str = char1 sub_str then
    begin
      if compare_strings str sub_str then
        n + sl
      else
        if l > sl then
          contains_substring_rec (next1 str) sub_str (l - 1) sl (n + 1)
        else
          raise Not_found;
    end
  else
    if l > sl then
      contains_substring_rec (next1 str) sub_str (l - 1) sl (n + 1)
    else
      raise Not_found;
;;

(* I do not understand why this does not exist in ocaml!!
    returns the index of the end of substr into str or raises Not_found *)
let contains_substring str substr =
  contains_substring_rec str substr (String.length str) (String.length substr) 0;;
(*> *)
```

3.2.2 Recherche des occurences des clefs dans un string

J'utilise la fonction `contains_substring` pour déterminer si une des clefs contenues dans la hashtable `keys_hastbl` est dans un string donné.

```
(* to be used as : (confront_key_to_string k d str) *)
let confront_key_to_string k key one_line =
  match one_line with
      Line( content, is_found, _) ->
        begin
          if is_found || key.is_found then
            one_line
          else
            try
              (* If the key trigger is in the line *)
```

```
                    let deb = contains_substring content key.trigger_pat in
                      begin
                        try
                          Line( String.sub content deb ((String.length content) - deb) , true, k)
                        with Invalid_argument( ia_s) ->
                          Printf.fprintf log_file "confront_...: String.sub problem on %s\n" content;
                          (* flush log_file; *)
                          Line( content , false, -1)
                      end
                  with
                      Not_found ->
                        Line( content, false, -1)
              end
        ;;
        (*> *)

        (* to find which key is in the string *)
        let has_key one_string =
          if String.length one_string = 0 then
            false
          else
            let line_s = Line( one_string, false, -1) in
            let is_it_a_key = Hashtbl.fold confront_key_to_string keys_hashtbl line_s in
              match is_it_a_key with
                  Line( str, is_found, nb) ->
                    if is_found then
                      let this_key = Hashtbl.find keys_hashtbl nb in
                        try
                          (* extraction du morceau important *)
                          let find_first = contains_substring str this_key.before_pat in
                          let first_part = String.sub str find_first ((String.length str) - find_first) in
                          (* en fait c'est bon :
                             - si c'est le dernier
                             - ou si le suivant n'est pas le même  *)
                          let find_last  = contains_substring first_part this_key.after_pat in
                          let this_value = String.sub first_part 0 (find_last - (String.length
        this_key.after_pat)) in
                            this_key.is_found <- true;
                            this_key.value    <- this_value;
                            Printf.fprintf log_file "KEY: %s -> %s\n" this_value this_key.name_key;
                            Hashtbl.replace keys_hashtbl nb this_key;
                            true;
                        with
                            Not_found -> false
                          | Invalid_argument( ia_s) ->
                              Printf.fprintf log_file "has_key: String.sub problem on %s\n" str;
                              (* flush log_file; *)
                              false
                    else
                      false
        ;;

        (* a key value is N x B x S, (N is a key name) where B is false is S is not relevant, true
        otherwise. *)
        type key_value = Key_value of string * bool * string;;

        (* to be used on the keys hashtable to get the value of a key *)
        let get_key_value_fold index key current_key_value =
          match current_key_value with
              Key_value( name, is_found, value) ->
                if is_found then
                  current_key_value
                else
                  if (key.name_key = name) && key.is_found then
                    Key_value( name, true, key.value)
                  else
                    current_key_value
        ;;

        (* to get the value of a given key *)
        let get_key_value k =
          let init_key_value = Key_value( k, false, "") in
          let found_result   = Hashtbl.fold get_key_value_fold keys_hashtbl init_key_value in
            match found_result with
                Key_value( s, b, value) ->
                  if b then
                    value
                  else
                    not_defined
        ;;

        (*> *)

        (*> *)
```

3.3 Niveau d'étoile

Comme cela a été exposé plus haut, les spécifications de CWEB précisent différents niveaux de blocs, marqués par un "étoilage" des marques de début de bloc.

```
(* to determine the "star level" of the section mark *)
let star_level str =
  let str_l = String.length str in
    if str_l < 3 then
      (str, Not_relevant)
    else
      let short_str = next2 str in
        (* Printf.fprintf log_file ">>>| %s\n" str; *)
```

```
            Printf.fprintf log_file "***| %s(%c)\n" short_str (char1_c short_str);
            match char1_c short_str with
                '*' ->
                  if str_l < 4 then
                    (next1 short_str, Star)
                  else
                    begin
                      let very_short_str = next1 short_str in
                        Printf.fprintf log_file "***| %s(%c)\n" very_short_str (char1_c very_short_str);
                        match char1_c very_short_str with
                            '*'       -> (next1 very_short_str, Double_star)
                          | '0' ..'9' -> (next1 very_short_str, Level( char1 very_short_str))
                          | _         -> (very_short_str, Star)
                    end
              | _   -> (short_str, Not_relevant)
;;

(* debug tool *)
let get_star_level l =
  match l with
      Not_relevant -> "NR"
    | Star -> "STAR"
    | Double_star -> "2STAR"
    | Level( f) -> "Level(" ^ f ^ ")"
;;

(*> *)
```

3.4 Explosion d'une ligne et attribution de son type

La tâche principale est de na pas tenir compte des blanc en début de ligne et de repérer les 4 types de marqueurs de début de ligne :

- %_ pour les commentaires simples
- %% pour les début de section
- %< pour les débuts de sous-section (avec retour charriot)
- %> pour les fins de sous-section

le reste étant du code

```
let rec explode s =
  if (String.length s) = 0 then
    (Into_code, "", Not_relevant)
  else
    match (char1 s) with
        " " -> explode (next1 s)
      | "%" ->
          begin
            if (String.length s) > 1 then
              (* dans les deux premier cas il faut regarder plus loin pour voir si la section est
étoilée *)
              match (char2 s) with
                  "%%" -> let (next_s, s_level) = star_level s in
                    Printf.fprintf log_file "+++| %s[%s]\n" next_s (get_star_level s_level);
                    (Begin_section, next_s, s_level)
                | "%<" -> let (next_s, s_level) = star_level s in
                    Printf.fprintf log_file "+++| %s[%s]\n" next_s (get_star_level s_level);
                    (Begin_subsection, next_s, s_level)
                | "%>" -> (End_subsection, (next2 s), Not_relevant)
                | _    -> (Into_comments, (next1 s), Not_relevant)
            else
              (Into_comments, (next1 s), Not_relevant)
          end;
      | _  -> (Into_code, s, Not_relevant)
;;
(*> *)
```

3.5 debug tools

Différentes fonctions de parcours de résultats intermédiaires, destinées à du débug.

```
(* pour résumer un string*)
let short_string str = (* str ;;*)
  try
    String.sub str 0 (min (String.length str) 10)
  with Invalid_argument( ia_s) ->
    Printf.fprintf log_file "short_string: String.sub problem on %s\n" str;
    (* flush log_file; *)
    ""
;;

(* pour décrire l'état *)
let say_state (s, str) =
  let s_str = short_string str in
    match s with
```

```
            Begin_section    -> Printf.fprintf log_file "Debut de section      <-- %s\n" s_str
          | Into_comments    -> Printf.fprintf log_file "Dans les commentaires <-- %s\n" s_str
          | Begin_subsection -> Printf.fprintf log_file "Debut de sous section <-- %s\n" s_str
          | End_subsection   -> Printf.fprintf log_file "Fin de sous section   <-- %s\n" s_str
          | Into_code        -> Printf.fprintf log_file "Dans le code          <-- %s\n" s_str
          | Begin_file       -> Printf.fprintf log_file "Tout début de fichier <-- %s\n" s_str
          | _                -> Printf.fprintf log_file "ERREUR!!!<-- %s\n" s_str
;;

(*> *)
```

3.6 Structure de blocs

Un bloc (une `section`) est composé d'une structure comportant :

- une référence,
- un titre,
- une zone de commentaires,
- une zone de code.

```
(* je vais ranger mon code dans un arbre *)
type 'a tree = Empty | Noeud of 'a * 'a tree list;;
(*type arbre = Empty | Noeud of section * arbre list;; *)


(* fabrication d'un noeud vide *)
type section = { refcode  : indented_ref ;
                 level    : state_level  ;
                 title    : string ;
                 comments : string ;
                 code     : string ;
               };;

let get_refcode_s d =
    match d.refcode with
        Indented_ref( s, _, _) -> s
;;

(* pour plus tard *)
let root_section = { refcode  = Indented_ref( root_string, 0, 0);
                     level    = Not_relevant ;
                     title    = "" ;
                     comments = "" ;
                     code     = "" ;
```

```
                          };;

(* pour plus tard *)
let empty_section = { refcode  = Indented_ref( empty_string, 0, 0);
                      level    = Not_relevant ;
                      title    = "" ;
                      comments = "" ;
                      code     = ""
                    };;
```

Encapsulation --- mise en page.

Il s'agit de l'encapsulation d'une section.

```
(* se débarasser des charactères spéciaux *)
let rec without_escapes s =
  if (String.length s) = 0 then
    ""
  else
    match (char1 s) with
      | "}" -> "\\}" ^ (without_escapes (next1 s))
      | "{" -> "\\{" ^ (without_escapes (next1 s))
      | "_" -> "\\_" ^ (without_escapes (next1 s))
      | "^" -> "\\^\\ " ^ (without_escapes (next1 s))
      | "#" -> "\\#" ^ (without_escapes (next1 s))
      | "\\" -> "\\bs{}" ^ (without_escapes (next1 s))
      | _  -> (char1 s) ^ (without_escapes (next1 s))
;;
```

En ce qui concerne les inclusions.

Ca se passe là:

```
let get_relation f =
  try
    Hashtbl.find relations (get_refcode_s f)
  with Not_found ->
    Printf.fprintf log_file "--[%s:notfound]\n" (get_refcode_s f);
    empty_string
;;

let get_subrelations f =
  try
    Hashtbl.find sub_relations (get_refcode_s f)
  with Not_found ->
    Printf.fprintf log_file "--[%s:notfound]\n" (get_refcode_s f);
    [ empty_string ]
;;

let rec pretty_print_rel_list l=
    match l with
        []           -> ""
      | head :: tail ->
          if (head = root_string) || (head = empty_string) then
            Printf.sprintf "%s" (pretty_print_rel_list tail)
          else
            Printf.sprintf "\\refangles{refcode:%s} %s" head (pretty_print_rel_list tail)
;;

let pretty_print_relations s empty_comments=
  Printf.fprintf log_file "PPR>\n";
  let prefix_relations = ref "" in
    if empty_comments then
      prefix_relations := ""
    else
      prefix_relations := "~\\\\\n";
    let p = get_relation s in
    let f = get_subrelations s in
      if p = "0" then
        if (List.length f) = 1 && (((List.hd f) = empty_string) || ((List.hd f) = root_string)) then
          ""
        else
          Printf.sprintf "%s{\\footnotesize \\uses %s}" !prefix_relations (pretty_print_rel_list f)
      else
        if f = [] then
          Printf.sprintf "%s{\\footnotesize \\usedby \\refangles{refcode:%s}}" !prefix_relations p
        else
          begin
            if (List.length f) = 1 && (((List.hd f) = empty_string) || ((List.hd f) = root_string))
then
              Printf.sprintf "%s{\\footnotesize \\usedby \\refangles{refcode:%s}}" !prefix_relations
p
            else
              Printf.sprintf "%s{\\footnotesize \\usedby \\anglesref{refcode:%s} --- \\uses %s}"
                !prefix_relations p (pretty_print_rel_list f)
          end
```

```
;;
(*> *)

let pretty_print_level l =
  match l with
      Not_relevant -> "0"
    | Star         -> "10"
    | Double_star  -> "100"
    | Level( f)    -> f
;;

let before_code_symbol = "\\bcodesymbol\n%END LATEX\n\\begin{quote}\n\\begin{alltt}\n";;
let after_code_symbol = "\\end{alltt}\n\\end{quote}\n%BEGIN LATEX\n}\n";;
let before_section_symbol = "\\bsecsymbol";;

let pretty_print_section sec =
  let after_comments_carriage = ref "" in
    if sec.comments = "" then
      after_comments_carriage := "\n"
    else
      after_comments_carriage := "~\\\\\n";
    match sec.refcode with
        Indented_ref( s, n, c) ->
          "\\noindent" ^ before_section_symbol ^ "\\nopagebreak\\\\\n" ^
          "\\sectitle{" ^ (pretty_print_level sec.level) ^ "}{" ^ sec.title ^ "}" ^
          "{" ^ (string_of_int c) ^ "}\n" ^ sec.comments ^ !after_comments_carriage ^
          (pretty_print_relations sec (sec.comments = "")) ^
          "\n\n" ^
          "\\nopagebreak\n{\\footnotesize\n" ^
          before_code_symbol ^
          sec.code ^
          after_code_symbol ;;

(* to gain CPU time*)
let process_strings_regexp = (regexp "\([, (\[]'\)\([^']*\)\('\)");;

(*
  Pas mieux pour l'instant:
  Test:
  # get_strings "ceci est un 'test' '''' assez 'étrange': A' + ['test''m''test', 'test','a']
('al').";;
*)
let process_strings str = global_replace process_strings_regexp "\1\\stringc{\2}\3" str;;

(* let process_cmts str = global_replace (regexp "\(%[^']*\)\(\\n\)") "\\cmt{\1}\\2" str;; *)
(* for the "end of line" comments *)

(* to gain CPU time*)
let colorize_regexp_1 = (regexp "|");;
let colorize_regexp_2 = (regexp "&");;
let colorize_regexp_3 = (regexp "=");;
let colorize_regexp_4 = (regexp "==");;
let colorize_regexp_5 = (regexp "~");;
let colorize_regexp_6 = (regexp "<=");;
let colorize_regexp_7 = (regexp ">=");;
let colorize_regexp_oo = ref (regexp
"\\b\(function\|while\|continue\|try\|catch\|for\|end\|persistent\|switch\|case\|if\|else\|elseif\|return\|break\|otherwise\)\\b");;
```

3.7 Déclarations de clefs globales spécifiques à OCAMAWEB

Je déclare ici les clefs de la section 3.2 :

clef	déclencheur	début	fin
author	node.author	'	'
	% author	'	'
	% auteur	'	'
title	% titre	'	'
	% title	'	'
project	% project	'	'
	% projet	'	'
mailto	node.mailto	'	'
	% mailto	'	'
date	node.date	'	'
	% date	'	'
version	VERSION	=	;
	% version	'	'

```
try
  let xml_inits = Xml.parse_file ((Sys.getenv "OCAMAWEB") ^ "ocamaweb.xml") in
  let caracts = (get_childs xml_inits) in
    (* caracts in a list of xml structs, I want the one with 'keypatterns' tag *)
  let is_keypatterns xml_struct =
    (get_name xml_struct) = "keypatterns"
  in
  let key_patterns = (get_childs (List.find is_keypatterns caracts)) in
    (* now I have to put the keypatterns into my hashtable *)
  let xml_key_atomic_value xml_list n =
    let couple = List.nth xml_list n in
      match couple with
          a,b -> b
  in
  let add_key_pattern_from_xml xml_struct =
    match xml_struct with
        name,attribs,childs ->
          let key_name    = xml_key_atomic_value attribs 0 in
          let key_sign    = xml_key_atomic_value attribs 1 in
          let before_name = xml_key_atomic_value attribs 2 in
          let after_name  = xml_key_atomic_value attribs 3 in
            add_key_pattern key_name key_sign before_name after_name
  in
  let is_matlabwords xml_struct =
    (get_name xml_struct) = "matlabwords"
  in
  let matlab_words = (get_childs (List.find is_matlabwords caracts)) in
    (* now I have to put the matlabwords into a list *)
  let extract_name_from_matlabwords xml_struct =
    match xml_struct with
        name,attribs,childs ->
          xml_key_atomic_value attribs 0
  in
    (* here I form a list with all the matlab words *)
  let matlabwords_list = List.map extract_name_from_matlabwords matlab_words in
  let matlabwords_concat mw_init mw_next =
    (mw_init ^ "\|" ^ mw_next)
  in
  let matlabwords_regexp = (regexp ("\\b\(" ^ (List.fold_left matlabwords_concat (List.hd 
matlabwords_list) (List.tl matlabwords_list)
                                             ^ "\)\\b"))) in
    (* here I have got the correct regexp, for instance :
       (global_replace matlabwords_regexp "\\ocamawebdefinition{\\0}" "if a==0 then 1 else 2 end;")
       returns a correct answer *)
    colorize_regexp_oo := matlabwords_regexp;

    (* get the keypattern informations *)
    List.iter add_key_pattern_from_xml key_patterns
with e ->
  Printf.fprintf stderr "OCAMAWEB: I cannot find <ocamaweb.xml> file in <%s>\n" (Sys.getenv 
"OCAMAWEB") ;
  add_key_pattern "author"  "node.author" "'" "'";
  add_key_pattern "author"  "% author"    "'" "'";
  add_key_pattern "author"  "% auteur"    "'" "'";
  add_key_pattern "title"   "% title"     "'" "'";
  add_key_pattern "title"   "% titre"     "'" "'";
  add_key_pattern "project" "% project"   "'" "'";
  add_key_pattern "project" "% projet"    "'" "'";
  add_key_pattern "mailto"  "node.mailto" "'" "'";
  add_key_pattern "mailto"  "% mailto"    "'" "'";
  add_key_pattern "date"    "node.date"   "'" "'";
  add_key_pattern "date"    "% date"      "'" "'";
  add_key_pattern "version" "% version"   "'" "'";
  add_key_pattern "version" "VERSION"     "=" ";"
;;

(*> *)
```

```
(* to colorize the code parts *)
let colorize str =
  process_strings
    (global_replace colorize_regexp_1 "{\\vt}"
       (global_replace colorize_regexp_2 "{\\va}"
          (global_replace colorize_regexp_3 "{\\oeq}"
             (global_replace colorize_regexp_4 "{\\ooeq}"
                (global_replace colorize_regexp_5 "{\\otdl}"
                   (global_replace colorize_regexp_6 "{\\(\\leq\\)}"
                      (global_replace colorize_regexp_7 "{\\(\geq\\)}"
                         (global_replace !colorize_regexp_oo "\\ocamawebdefinition{\\0}"
                            (without_escapes str))))))))))
;;

let without_path f =
  try
    let deb_path = 1 + (String.rindex f '/') in
      String.sub f deb_path ((String.length f) - deb_path)
  with Not_found ->
    try
      let deb_path = 1 + (String.rindex f '\\') in
        String.sub f deb_path ((String.length f) - deb_path)
    with Not_found ->
      f
;;

let neutralize_filenames_regexp = (regexp "_");;

(* juste pour neutraliser les _ et \ dans les noms de (fichiers) *)
let neutralize_filenames str_fname =
  let str = without_path str_fname in
    try
      let ridx = (1 + (String.rindex str '/')) in
      let only_filename = String.sub str ridx ((String.length str) - ridx) in
        global_replace neutralize_filenames_regexp "\\_" only_filename
    with
        Not_found -> global_replace neutralize_filenames_regexp "\\_" str
;;

let rec first_defined str_lst =
  match str_lst with
      []           -> not_defined
    | head :: tail ->
        if head = not_defined then
          first_defined tail
        else
          head
;;

let get_email str =
  if str = not_defined then
    ""
  else
```

```
    "\\\\{\\tt mailto:" ^ str ^ "}"
;;

(* to include ocamaweb.sty file into the LaTeX one *)
let windows_path_to_unix str = global_replace (regexp "[\\]") "/" str;;

let get_style_path =
  try
    (windows_path_to_unix (Sys.getenv "OCAMAWEB"))
  with Not_found ->
    ""
;;

let get_first_line _ =
  let project_name_v = (get_key_value "project") in
    if project_name_v = not_defined then
      ""
    else
      project_name_v ^ "\\\\"
;;

let title_or_filename _ =
  let tit = (get_key_value "title") in
  if tit = not_defined then
    (get_key_value "filename")
  else
    tit
;;

let head_of_file _ =
  "\\input{" ^ get_style_path ^ "ocamaweb.sty}\n\n" ^
  "\\def\\ocamawebauthor{" ^ (get_key_value "author") ^ "}\n" ^
  "\\def\\ocamawebtitle{" ^ (title_or_filename ()) ^ "}\n" ^
  "\\def\\ocamawebproject{" ^ (get_key_value "project") ^ "}\n" ^
  "\\def\\ocamawebfilename{" ^ (get_key_value "filename") ^ "}\n" ^
  "\\def\\ocamawebmailto{" ^ (get_key_value "mailto") ^ "}\n" ^
  "\\def\\ocamawebdate{" ^ (get_key_value "date") ^ "}\n" ^
  "\\def\\ocamawebversion{" ^ (get_key_value "version") ^ "}\n\n" ^
  "\\ocamawebstart\n\n"
;;
(*
   "\\title{" ^
   (get_first_line ()) ^
   (first_defined [(get_key_value "title")   ; (get_key_value "filename") ]) ^ "}\n" ^
   "\\author{"  ^ (get_key_value "author") ^ (get_email (get_key_value "mailto")) ^ "}\n" ^
   "\\date{Imprimé le \\today,\\\\dernière modification le " ^ (get_key_value "date") ^ "}\n" ^
   "\\ocamawebstart\n\n";;
*)

let foot_of_file =
  "\n\\noindent\\rule{5cm}{1pt}\\nopagebreak\\\\\n\\ocamawebend\n";;

(*> *)
```

3.8 fonctions de manipulation de l'arbre

Un arbre est ou bien vide, ou bien un `Noeud` contenant un label et une liste de noeuds.
L'arbre d'un fichier est donc une racine qui a comme fils les blocs principaux. Chaque bloc principal a comme fils les blocs arbitraire qu'il contient et de même chaque bloc arbitraire a comme fils les blocs arbitraire qu'il contient.
Le label d'un noeud est composé d'une section (cf 3.6).

```
(* pour résoudre un problème d'empilage: je vais créer une fonction
   qui inverse l'ordre du premier niveau de l'arbre *)
let reverse_sons t =
  match t with
      Empty -> t
    | Noeud(s, f) -> (* je vais inverser l'ordre des fils |f| et les remettre dans s: *)
        Noeud(s, List.rev f)
;;


(* affiche un titre *)
let print_title s =
  Printf.fprintf stdout "%s%s\n" (String.make 1 ' ') s.title;;

(* listing des titres des noeuds d'un arbre *)
let rec get_section = function
    Empty -> empty_section
  | Noeud(s, _) -> s
;;

(* liste les différents titres d'un arbre *)
let rec get_t nb n =
  match n with
      Empty -> Printf.fprintf stdout "%s\n" (String.make nb '@');
    | Noeud(s, l) ->
        begin
          Printf.fprintf stdout "%s%s\n" (String.make nb '@') s.title;
          ignore (List.map (get_t (2 + nb)) l);
        end;
;;

(* liste les différents commentaires d'un arbre *)
let rec get_c nb n =
  match n with
      Empty -> ignore (Printf.fprintf stdout "%s\n" (String.make nb '@'));
    | Noeud(s, l) ->
```

```
            begin
              Printf.fprintf stdout "%s%s\n%s\n" (String.make nb '@') s.title s.comments;
              ignore (List.map (get_c (2 + nb)) l);
            end;
            ()
;;
```

remplissage d'un vecteur avec la correspondance péres --- fils.

J'utilise deux variables globales pour cela (je sais ce n'est pas très élégant, mais je suis pressé pour l'instant). Ces deux variables sont remplies à la fin du parsing.

Il s'agit de Hashtables qui contiennent respectivement les fils et les pères de chaque section.

```
let concat_subrelation k v =
  try
    let l = Hashtbl.find sub_relations k in
      Hashtbl.replace sub_relations k (List.append l [ v])
  with Not_found ->
    Hashtbl.add sub_relations k [v]
;;

let get_ref_number s =
  match s.refcode with
      Indented_ref(_, _, s) -> (string_of_int s)
;;

let add_to_relations f d =
  (* Printf.fprintf stdout "REL> %s --son of-- %s\n"    (get_refcode_s d) (get_refcode_s f); *)
  Hashtbl.add relations (get_refcode_s d) (get_ref_number f);
  (* Printf.fprintf stdout "REL> %s --father of-- %s\n" (get_refcode_s f) (get_refcode_s d); *)
  concat_subrelation (get_refcode_s f) (get_ref_number d)
;;

(* renseigne récursivement les hashtables d'indexation des pères et fils *)
let rec build_relations_rec f a_tree =
    match a_tree with
        Empty -> ()
      | Noeud(desc, fils) ->
          add_to_relations f desc;
          ignore (List.map (build_relations_rec desc) fils)
;;

(*> *)
```

Affichage itératif des fils de la racine d'un arbre.

C'est une des fonctions importantes du code, elle appelle `son_to_string` sur toutes les branches de l'arbre.

```
(* affichage du contenu d'un élément puis de ses fils comme déclarations de blocs *)
let rec son_to_string str l_tree =
  match l_tree with
      Empty -> str
    | Noeud(t, l) ->
        str ^ (pretty_print_section t ) ^ (List.fold_left son_to_string "" l)
;;

let to_string t =
  match t with
      Empty -> "";
    | Noeud(s, l) -> (* j'ai les noeud principaux,
                        je vais afficher :
                        - le contenu du premier avec un code c
                        - puis ses blocs de référence
                        - puis pour chaque autre élément de la liste :
                        - le contenu du premier
                        - puis ses blocs de référence
                     *)
        let first = List.nth l 0 in
          List.fold_left son_to_string "" l
;;
(*> *)

(*> *)

(*> *)

(*> *)
```

4 file parsing

```
(* récupération de l'état d'une ligne *)
let get_line_state (s, _, _) = s;;
let get_line_str   (_, s, _) = s;;
let get_line_level (_, _, l) = l;;

(* croisement de l'état global et de celui d'une ligne.
   C'est LA grosse machine à états du parseur
*)
let cross_states g_state pline =
  let this_level = get_line_level pline in
  let l_state    = get_line_state pline in
  let str        = short_string (get_line_str   pline) in
    match g_state with
        Begin_section | Into_comments | Begin_subsection   -> begin
          match l_state with
              Begin_section    -> Printf.fprintf log_file "a(1) new section: %s ...\n" str;
                ( Begin_section, New_section, this_level)
            | Into_comments    -> Printf.fprintf log_file "b(2) add comments to current
(sub?)section: %s ...\n" str;
                ( Into_comments, Add_comment, this_level)
            | Begin_subsection -> Printf.fprintf log_file "c(3) new subsection (insert into former
code): %s ...\n" str;
                ( Begin_subsection, New_subsection, this_level)
            | End_subsection   -> Printf.fprintf log_file "d(4) close current (sub)section: %s
...\n" str;
                ( End_subsection, Close_subsection, this_level)
            | Into_code        -> Printf.fprintf log_file "e(5) close comment part and begin code
part: %s ...\n" str;
                ( Into_code, Add_code, this_level)
            | Begin_file       -> Printf.fprintf log_file "f(*) ERROR!!: %s ...\n" str;
                ( Unknown, Stop, this_level)
            | _                -> ( Unknown, Stop, this_level)
        end;
      | End_subsection | Into_code   -> begin
          match l_state with
              Begin_section    -> Printf.fprintf log_file "g(1) new section: %s ...\n" str;
                ( Begin_section, New_section, this_level)
            | Into_comments    -> Printf.fprintf log_file "h(6) stay into code: %s ...\n" str;
                ( Into_code, Add_comments_into_code, this_level)
            | Begin_subsection -> Printf.fprintf log_file "i(3) new subsection (insert into former
code): %s ...\n" str;
                ( Begin_subsection, New_subsection, this_level)
            | End_subsection   -> Printf.fprintf log_file "j(4) close current (sub)section: %s
...\n" str;
                (End_subsection, Close_subsection, this_level)
            | Into_code        -> Printf.fprintf log_file "k(6) stay into code: %s ...\n" str;
                (Into_code, Add_code, this_level)
            | Begin_file       -> Printf.fprintf log_file "l(*) ERROR!!: %s ...\n" str;
                (Unknown, Stop, this_level)
            | _                -> ( Unknown, Stop, this_level)
        end;
      | Begin_file       ->begin
          match l_state with
              Begin_section    -> Printf.fprintf log_file "m(7) compose the first section: %s ...\n"
str;
                (Into_comments, Into_first_section, this_level)
            | Into_comments    -> Printf.fprintf log_file "n(8) stay into Begin_file (add to code):
%s ...\n" str;
                (Begin_file, Add_code, this_level)
            | Begin_subsection -> Printf.fprintf log_file "o(*) ERROR!!: %s ...\n" str;
                (Unknown, Stop, this_level)
            | End_subsection   -> Printf.fprintf log_file "p(*) ERROR!!: %s ...\n" str;
                (Unknown, Stop, this_level)
            | Into_code        -> Printf.fprintf log_file "q(8) stay into Begin_file (add to code):
%s ...\n" str;
                (Begin_file, Add_code, this_level)
            | Begin_file       -> Printf.fprintf log_file "l(*) ERROR!!: %s ...\n" str;
                ( Unknown, Stop, this_level)
            | _                -> ( Unknown, Stop, this_level)
        end;
      | _ -> ( Unknown, Stop, this_level)
;;

(* variable globale *)
let global_state = ref Begin_file ;;
```

5 outils de manipulation de fichier

```
(* renvoie tout avant le premier ["."] rencontré *)
let title_part str_ =
  if String.length str_ = 0 then
    ""
  else
    let spaces_nb = ref 0 in
      try
        while (String.sub str_ !spaces_nb 1= " ") do
          spaces_nb := !spaces_nb + 1
        done;
        let str = String.sub str_ !spaces_nb (String.length str_ - !spaces_nb) in
          begin
            try
              let point_pos = String.index str '.' in
                if point_pos > 0 then
                  String.sub str 0 point_pos
```

```
                    else
                      ""
                with
                    Not_found -> str
              end
          with Invalid_argument( ia_s) ->
            Printf.fprintf log_file "title_part: String.sub problem on %s\n" str_;
            (* flush log_file; *)
            str_
;;

(* renvoie tout après le premier ["."] rencontré *)
let comment_part str =
  try
    let point_pos = String.index str '.' in
    let str_len   = String.length str in
      if str_len > (1 + point_pos) then
        String.sub str (1 + point_pos) ( str_len - point_pos - 1)
      else
        ""
  with
      Not_found -> ""
;;

(* I had my xml parsing functions here *)
```

5.1 Parsing spécifique aux langages sans commentaires par blocs

Cette partie doit être remplacée par une autre pour les langages (comme le `c` ou `ocaml`) qui offrent la possibilité de créer des blocs de commentaires (`/* ... */` ou `(* *)`).

```
let refcode_counter = ref 0;;

let make_generic_refcode str_ dec =
  refcode_counter := !refcode_counter + 1 - dec;
  if String.length str_ = 0 then
    Indented_ref ("", 0, !refcode_counter)
  else
    let spaces_nb = ref 0 in
      while (String.sub str_ !spaces_nb 1= " ") do
        spaces_nb := !spaces_nb + 1
      done;
      let str = String.sub str_ !spaces_nb (String.length str_ - !spaces_nb) in
      let (str_unstar, s) = star_level str in
```

```
          let title = title_part str_unstar in
            (* Printf.fprintf log_file "RRR> %s\n-*-> %s\n" str str_unstar; *)
            if dec = 1 then
              begin
                Printf.fprintf log_file "RR1> %s |%d|%d (%d)\n" title !spaces_nb (!refcode_counter +
dec) dec;
                Indented_ref (title, !spaces_nb, !refcode_counter + dec);
              end
            else
              begin
                Printf.fprintf log_file "RR0> %s |%d|%d (%d)\n" str 0 (!refcode_counter + dec) dec;
                Indented_ref (str, 0, !refcode_counter + dec)
              end
;;
(* renvoie un refcode pour le string :
   un "nettoyage" (pour WEB) de la partie du string avant le 1er ["."] rencontré.
   D'abord j'enlève (et je compte) le nombre d'espaces qu'il y a avant
*)
let make_refcode str_ =
  make_generic_refcode str_ 0;;

(* le même mais dans le vide *)
let make_fake_refcode str_ =
  make_generic_refcode str_ 1;;


(* référence un bloc par refcode (les retours charriot ne sont pas compris) *)
let insert_refcode my_refcode =
  match my_refcode with
    |  Indented_ref( s, n, c) ->
         (String.make n ' ') ^ "\\refcode{" ^ s ^ "}{" ^ (string_of_int c) ^ "}" ;;

(* declare un bloc par refcode (les retours charriot ne sont pas compris)
let declare_refcode str = "@<" ^ str ^ "@>=";;

let end_of_first_comments = "@c\n";;
let end_of_comments       = "@p\n";;
*)

(*> *)

(*> *)
```

6 Traitements principaux

```
(* j'ai besoin d'une racine :
   une liste de noeuds (je la met à l'extérieur pour que cela ne soit pas trop récursif -mal de tête-
) *)
let root_nodes = ref [];;

let regexp_no_tabs = regexp "[\t]";;
```

6.1 parcours récursif d'un fichier (channel)

```
let rec make_section first_line channel_name first_line_level =
  let refcode_      = ref (Indented_ref("", 0, 0)) in
  let title_        = ref "" in
  let level_        = ref first_line_level in
  let comments_     = ref "" in
  let code_         = ref "" in
  let into_parsing = ref true in
  let fils          = ref [] in
    if !global_state <> Begin_file then
      begin
        title_    := title_part first_line;
        comments_ := comment_part first_line;
        refcode_  := make_refcode first_line;

        Printf.fprintf log_file "**>> %s\n" (get_star_level !level_);
      end;

    (* tant je ne dois pas créer une section de façon récursive *)
    while !into_parsing do
      (* flush log_file; *)
      try
        let this_line =
          (global_replace regexp_no_tabs " " (input_line channel_name)) in
        let has_any_special_keys = has_key this_line in
        let this_parsed_line     = explode this_line in
        let this_str             = (get_line_str this_parsed_line) in
        let carriage             = "\n" in
        let (new_state, this_action, this_state_level) = cross_states !global_state this_parsed_line
in

          global_state := new_state;

          (* j'ai remplacé certains this_str par des this_line car pour l'instant
             j'utiliser le package alltt, j'espère bien y remédier... *)
          match this_action with
              Add_comment            -> comments_ := !comments_ ^ carriage ^ this_str
                (* ATTENTION: je devrais chercher les commentaires de fin de ligne
                    problème des '%'!!! *)
             | Add_code               -> code_     := !code_     ^ carriage ^ (colorize this_line)
             | Add_comments_into_code -> code_     := !code_     ^ carriage ^ "\comments{" ^ this_str
^ "}"
             | New_section            ->
                 begin
                   Printf.fprintf log_file "*>>> (%s)%s\n" this_str (get_star_level
this_state_level);
                   root_nodes := !root_nodes @ [ (make_section this_str channel_name
this_state_level) ];
                   (* root_nodes := [ (make_section this_str channel_name) ] @ !root_nodes; :pas bon
du tout *)
                   into_parsing := false
                 end;
             | New_subsection ->
                 begin
                   code_    := !code_ ^ carriage ^ (insert_refcode (make_fake_refcode this_line)); (*
this_str *)
                   Printf.fprintf log_file "*>>> (%s)%s\n" this_str (get_star_level
this_state_level);
                   fils     := !fils  @ [ (make_section this_str channel_name this_state_level) ]
                 end;
             | Into_first_section ->
                 begin
                   title_    := title_part     this_str;
                   level_    := get_line_level this_parsed_line;
                   comments_ := comment_part   this_str;
                   refcode_  := make_refcode   this_str;
                 end;
             | Close_subsection -> into_parsing := false
             | Stop             -> into_parsing := false

          with End_of_file -> into_parsing := false

    done;

    (* je renvoie un noeud quicontient la description de cette section *)
    let this_section = { refcode  = !refcode_ ;
                         level    = !level_ ;
                         title    = !title_ ;
                         comments = !comments_ ;
                         code     = !code_
                       } in
      Noeud( this_section, !fils )
;;
(*> *)
```

6.2 fonction principale

- ouverture du fichier
- mise à "vide" de l'ensemble des noeuds racine
- mise à `Begin_file` du `global_state`
- lancement du parcours recursif

```
let follow_file fname =
  root_nodes   := [];
  global_state := Begin_file ;
  add_specified_key_pattern "filename" (neutralize_filenames fname) ;
  let this_channel = open_in fname in
    let first_node   = make_section "" this_channel Not_relevant in
      (* attention: je fais ici un List.rev pour supprimer un comportement INEXPLIQUE:
         le premier niveau de liste se retrouve empilé à l'envers! *)
    let all_nodes    = Noeud( root_section, List.rev (!root_nodes @ [ first_node ])) in
      flush log_file;
      all_nodes
;;

(*> *)
```

6.3 for testing purpose

```
let send_to_file a_tree filename =
  build_relations_rec empty_section a_tree;
  let this_chan = open_out_bin filename in
    Printf.fprintf this_chan "%s%s%s" (head_of_file ()) (to_string a_tree) foot_of_file;
    close_out this_chan
;;

let without_ext f =
  let deb_ext = String.rindex f '.' in
    String.sub f 0 deb_ext
;;

let rec move_files name dest exts =
  match exts with
      [] -> 0
    | head :: tail ->
        (Sys.command( "move " ^ name ^ head ^ " " ^ dest)) +
        move_files name dest tail
;;
```

```
let get_dest_path =
  try
    (Sys.getenv "OCAMAWEB_DEST")
  with Not_found ->
    Printf.fprintf log_file "ENV: OCAMAWEB_DEST not found!!!\n";
    ""
;;

let latexize f mode =
  let rez = ref 0 in
  let new_f   = without_ext f in
    rez := !rez + (Sys.command( "del " ^ get_dest_path ^ (without_path new_f) ^ ".* "));
    rez := !rez + 2 * (Sys.command( "latex " ^ (without_path f)));
    rez := !rez + 4 * (Sys.command( "latex " ^ (without_path f)));
    rez := !rez + 8 * (Sys.command( "dvipdfm " ^ (without_path new_f) ^ ".dvi"));
    if mode = 2 then
      begin
        !rez + 100  * (move_files (without_path (without_ext f))
                     (Sys.getenv "OCAMAWEB_DEST")
                     [ ".pdf" ; ".tex" ; ".dvi" ; ".log" ; ".aux" ; ".toc"] );
      end
    else
      !rez + 16;
;;

let ocamaweb_process f1 f2 mode =
  Printf.fprintf stderr "%s --> %s\n" f1 f2;
    let garzol = follow_file f1 in
      if mode < 3 then
        begin
          send_to_file garzol (without_path f2);
          latexize f2 mode
        end
      else
        begin
          send_to_file garzol f2;
          1
        end
;;


let main () =
  Printf.fprintf stderr "OCAMAWEB version-%s\n" version;
  if !Sys.interactive then
    2
  else
    begin
      match Array.length Sys.argv with
            2 ->
              let file_to_comment = Sys.argv.(1) in
              let file_to_write   = (Sys.getenv "OCAMAWEB_DEST") ^
                                    (without_path (without_ext file_to_comment)) ^ ".tex" in
                ocamaweb_process file_to_comment file_to_write 2
                | 3 ->
                   let file_to_comment = Sys.argv.(1) in
                   let file_to_write   = Sys.argv.(2) in
                     ocamaweb_process file_to_comment file_to_write 3
                | _ ->
                   ignore (Printf.fprintf stdout "ocamaweb input_file.m [output_file.tex]\n");

      close_out log_file;
      1;
    end
;;

main ();;


(*> *)

(*> *)
```

1

see `http://www.literateprogramming.com`.
