```
open Xml
open ExtLib
open Printf
open Rdf

module G = RdfUtil.G
module Ht = Hashtbl

let space = Pcre.regexp "[ \t]+"

let file_to_list file conv =
  let ic = open_in file in
  let lines = input_list ic in
  close_in ic;
  List.map conv lines

let make_rewrite file =
  let lst = file_to_list file
      (fun l ->
        let ch = Pcre.split ~rex:space l in
        (List.hd ch),(List.nth ch 1)) in
  let rewrite (s,p,o) g =
  let _,s = List.fold_left
      (fun (b,s) (sub,by) -> String.replace ~str:s ~sub ~by)
      (false,s) lst in
  let _,o = List.fold_left
      (fun (b,o) (sub,by) -> String.replace ~str:o ~sub ~by)
      (false,o) lst in
  G.add_edge_e g (s,p,o)
  in
  rewrite

let make_select file =
  let lst = file_to_list file
      (fun l ->
        let ch = Pcre.split ~rex:space l in
        let srex = List.hd ch in
        let prex = List.nth ch 1 in
        let orex = List.nth ch 2 in
        match srex,prex,orex with
        | "_","_",op when op <> "_" -> fun (s,p,o) -> o = op
        | "_",pp,"_" when pp <> "_" -> fun (s,p,o) -> p = pp
        | sp,"_","_" when sp <> "_" -> fun (s,p,o) -> s = sp
        | "_",pp,op when pp <> "_" && op <> "_" ->
            fun (s,p,o) -> p = pp && o = op
        | sp,"_",op when sp <> "_" && op <> "_" ->
            fun (s,p,o) -> s = sp && o = op
        | sp,pp,"_" when sp <> "_" && pp <> "_" ->
            fun (s,p,o) -> s = sp && p = pp
        | sp,pp,op when sp <> "_" && pp <> "_" && op <> "_" ->
            fun (s,p,o) -> s = sp && p = pp && o = op
        | sp,pp,op -> fun (s,p,o) -> true)
  in
  let select ((s,p,o) as t) = List.exists (fun f -> f t) lst in
  select

let get_url rewrite select url =
  fprintf stderr "getting %s...\n" url; flush stderr;
  let url =
    if String.starts_with url "<http:" then RdfUtil.form_url url
    else if String.starts_with url "http:" then url
    else failwith (sprintf "Error: get_url: %s is not URL." url) in
```

```
  try
    Some(RdfUtil.rewrite_graph
      rewrite (RdfUtil.filter_graph select (RdfUtil.graph_of_url url)))
  with exc ->
    fprintf stderr "Lod_crawler.get_url: exception %s (ignored)\n"
      (Printexc.to_string exc);
    flush stderr;
    None

let crawl depth rewrite select url =
  let ldone = Ht.create 1 in
  let get u = Ht.add ldone u (); get_url rewrite select u in
  let rec loop n graph =
    if n = depth then graph
    else
      let nht = Ht.create 1 in
      fprintf stderr "depth %d/%d\n" n depth; flush stderr;
      G.iter_edges_e
        (fun ((s,p,o) as t) ->
          if select t
          then begin
            if RdfUtil.is_url s && not(Ht.mem ldone (RdfUtil.form_url s))
            then Ht.replace nht (RdfUtil.form_url s) ()
            else ();
            if RdfUtil.is_url o && not(Ht.mem ldone (RdfUtil.form_url o))
            then Ht.replace nht (RdfUtil.form_url o) ()
            else ();
          end)
        graph;
      let ngraph = Ht.fold
          (fun k _ ng ->
            match get k with
            | Some g -> RdfUtil.add_graph g ng
            | None -> ng)
          nht graph in
      if G.nb_edges graph = G.nb_edges ngraph
      then graph
      else loop (n+1) ngraph in
  match get url with
  | Some g -> loop 0 g
  | None -> G.empty

let main () =
  let fselect = ref "" in
  let frewrite = ref "frewrite.dat" in
  let dep = ref 1 in
  let url0 = ref "" in
  let specs = [
    ("-select", Arg.Set_string fselect, "patterns of selected triples");
    ("-rewrite", Arg.Set_string frewrite, "patterns of replaced URL's");
    ("-depth", Arg.Set_int dep, "depth of searching");
  ] in
  let usage = sprintf "Usage: %s [options] URL > outfile.n3\n"
      Sys.executable_name in
  Arg.parse specs (fun s -> url0 := s) usage;
  if !fselect = "" || !frewrite = "" || !url0 = "" then
    (Arg.usage specs usage; exit 1);
  let select = make_select !fselect in
  let rewrite = make_rewrite !frewrite in
  let url = !url0 in
  let depth = !dep in
  let graph = crawl depth rewrite select url in
  fprintf stderr "graph size (%d) = %d %d\n"
```

```
    !dep (G.nb_vertex graph) (G.nb_edges graph);
  G.iter_edges_e (fun (s,p,o) -> printf "%s %s %s .\n" s p o) graph;
  ()

let _ =
  try
    main ()
  with
  | Xml.Error e -> fprintf stderr "Xml.Error: %s\n" (Xml.error e)
  | e -> raise e
```