

erl2latex: Literal Erlang Programming

November 28, 2008

Copyright (c) 2008 Ulf Wiger, John Hughes¹

1 Introduction

This module converts an Erlang source file to latex. The latex file can then be converted to e.g. PDF, using pdflatex or similar tool.

The idea of ‘literal Erlang programming’ is that the source and comments should read as a good paper. Unlike XML markup, Latex markup is also fairly unobtrusive when reading the source directly.

```
-module(erl2latex).  
  
-export([file/1, file/2]).
```

2 file/[1,2]

The interface is:

file(Filename [, Target.filename]) -> ok — {error, Reason}

If no target filename is given, the .erl extension of the source filename will be stripped and replaced with .tex.

¹The MIT License

Copyright (c) 2008 Ulf Wiger <ulf@wiger.net>, John Hughes <john.hughes@quviq.com>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```

file(F) ->
  file(F, latex_target(F)).

file(F, Target) ->
  case file:read_file(F) of
    {ok, Bin} ->
      output(convert_to_latex(Bin), Target);
    Err ->
      Err
  end.

```

The actual conversion function. We separate comments from code, and convert each block to latex separately. We then insert a preamble, if not already present, or insert a small formatting macro for the source code (if not already defined).

```

convert_to_latex(Bin) ->
  Parts = split_input(binary_to_list(Bin)),
  case lists:flatten([convert_part(P) || P <- Parts]) of
    "\\document" ++ _ = Latex0 ->
      {Preamble,Doc} = get_preamble(Latex0),
      [Preamble, source_listing_setup(Preamble),
       Doc, end_doc()];
    Latex0 ->
      [default_preamble(),
       "\\begin{document}\n",
       Latex0, end_doc()]
  end.

```

If a preamble is present, the `\begin{document}` entry must also be present. This is how we know where the preamble ends.

```

get_preamble(Str) ->
  get_preamble(Str, []).

get_preamble("\\begin{document}" ++ Rest, Acc) ->
  {lists:reverse("\n" ++ Acc), "\\begin{document}\n" ++ Rest};
get_preamble([H|T], Acc) ->
  get_preamble(T, [H|Acc]).

default_preamble() ->
  ["\\documentclass[a4paper,12pt]{article}\n",
   source_listing_setup()].

source_listing_setup(Preamble) ->
  case regexp:first_match(Preamble, "begin{mylisting}") of
    {match,_,_} ->
      [];
    nomatch ->
      source_listing_setup()
  end.

source_listing_setup() ->
  ("\\newenvironment{mylisting}\n"

```

```

"{{\\begin{list}}{\\setlength{\\leftmargin}{1em}}"
"\\item\\scriptsize\\bfseries}\\n"
"{{\\end{list}}\\n"
"\\n"
"\\newenvironment{mytinylisting}\\n"
"{{\\begin{list}}{\\setlength{\\leftmargin}{1em}}"
"\\item\\tiny\\bfseries}\\n"
"{{\\end{list}}\\n").

end_doc() ->
"\\n\\end{document}\\n".

split_input(Txt) ->
group([wrap(L) || L <- lines(Txt)]).

lines(Str) ->
lines(Str, []).

lines("\\n" ++ Str, Acc) ->
[lists:reverse(Acc) | lines(Str, [])];
lines([H|T], Acc) ->
lines(T, [H|Acc]);
lines([], Acc) ->
[lists:reverse(Acc)].

wrap("%" ++ S) ->
{comment, strip_comment(S)};
wrap(S) ->
{code, S}.

group([T,C|Tail]) ->
{More,Rest} = lists:splitwith(fun({T1,_C1}) -> T1 == T end, Tail),
[{T,[C|[C1 || {_,C1} <- More]] | group(Rest)}];
group([]) ->
[].
```

In this function, we wrap the different ‘source’ and ‘comment’ blocks appropriately. The weird-looking split between string parts is to keep pdf_latex from tripping on what looks like the end of the verbatim block.

```

convert_part({code,Lines}) ->
["\\begin{mylisting}\\n"
"\\begin{verbatim}\\n",
[[expand(L),"\\n"] || L <- Lines],
"\\end{verbatim}\\n"
"\\end{mylisting}\\n\\n"];
convert_part({comment,Lines}) ->
[[[L,"\\n"] || L <- Lines],"\\n"].
```

The `expand(Line)` function expands tabs for better formatting. The tab expansion algorithm is really too simplistic.

```

expand(Line) ->
  lists:map(fun($\t) -> ["  "];
            (C) -> C
            end, Line).

```

Following edoc convention, comments are excluded if the first non-space character following the leading string of % is another %, for example:
 %% % This comment will be excluded.

```

strip_comment(C) ->
  C1 = strip_percents(C),
  case string:strip(C1, left) of
    "%" ++ _ -> "";
    Stripped ->
      Stripped
  end.

strip_percents("%" ++ C) ->
  strip_percents(C);
strip_percents(C) ->
  C.

```

Finally, just a few utility functions.

```

latex_target(F) ->
  filename:basename(F, ".erl") ++ ".tex".

output(Data, F) ->
  file:write_file(F, list_to_binary(Data)).

```