erl2latex: Literal Erlang Programming

Ulf Wiger <ulf@wiger.net>

March 11, 2009

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.

See the Makefile for hints on how to integrate erl2latex into your own build system. You can call the emktex script using a symbolic link to the original script, which is located in the erl2latex/src directory.

```
-module(erl2latex).

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

2 file/[1,2]

The interface is:
file(Filename [, Options]) -> ok | {error, Reason}

Options can be specified either when calling file/[1,2], or by adding an attribute, -erl2latex(Options), in the source code. The attribute will not be

[1]The 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.

included in the latex output. Options given in the function call will shadow options embedded in the source code.

mode: ‘normal’ or ‘included’. If mode is ‘included’, preamble and document begin and end markers are removed if found.

```
-spec file/1 :: (Filename::string()) -> ok | {'error',atom()}.

file(F) ->
    file(F, []).

-type option() :: {documentclass, none | auto | string()}
                | {mode, normal | included}
                | {source_listing, auto | string()}.

-spec file/2 :: (Filename::string(), [option()]) ->
                    {ok,string()} | {'error',atom()}.

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

3 Conversion to Latex

Below is 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, Options0) ->
    Parts0 = split_input(binary_to_list(Bin)),
    {Parts1, Embedded_options} = embedded_options(Parts0),
    Parts = rearrange_if_escript(Parts1),
    Options = Options0 ++ Embedded_options,
    Mode = proplists:get_value(mode, Options, normal),
    case lists:flatten([convert_part(P) || P <- Parts]) of
        "\\document" ++ _ = Latex0 ->
            {Preamble,Doc} = get_preamble(Latex0),
            [[[Preamble,
               source_listing_setup(Preamble),
               "\\begin{document}\n"] || Mode == normal],
             Doc,
             end_doc(Mode)];
        Latex0 ->
            [[[default_preamble(Options),
               "\\begin{document}\n"] || Mode == normal],
             Latex0,
            end_doc(Mode)]
    end.
```

If a preamble is present, the \begindocument 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), Rest};
get_preamble([H|T], Acc) ->
    get_preamble(T, [H|Acc]).
```

The following functions output the default latex preamble and document end marker.

```
default_preamble(Options) ->
    [Doc_class,Src_listing] =
        [proplists:get_value(P,Options) ||
            P <- [documentclass, source_listing]],
    document_class(Doc_class) ++ source_listing_setup(Src_listing,"").

document_class(Opt) ->
    if Opt==auto; Opt==undefined ->
            "\\documentclass[a4paper,12pt]{article}\n";
       Opt==none -> "";
       is_list(Opt) -> Opt
    end.

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

source_listing_setup(undefined) ->
    source_listing_setup
      ("\\setlength{\\leftmargin}{1em}}"
       "\\item\\scriptsize\\bfseries");
source_listing_setup(Str) when is_list(Str) ->
    ("\\newenvironment{mylisting}\n"
     "{\\begin{list}{}{")
        ++ Str
        ++ ("}\n"
            "{\\end{list}}\n"
            "\n").


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

end_doc(included) ->
    "";
end_doc(normal) ->
    end_doc().
```

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

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

normalize(["\n","\n"|T]) ->  normalize(["\n"|T]);
normalize([H|T])         ->  [H|normalize(T)];
normalize([])            ->  [].
```

The expand(Line) function expands tabs for better formatting.

```
expand(Line) ->
    expand_tabs(Line).

expand_tabs(Xs) ->
  expand_tabs(0,Xs).

expand_tabs(_N,[]) ->
    [];
expand_tabs(N,[$\t|Xs]) ->
    N1 = 8*(N div 8) + 8,
    [$\s || _ <- lists:seq(N,N1)] ++ expand_tabs(N1,Xs);
expand_tabs(N,[X|Xs]) ->
    [X|expand_tabs(N+1,Xs)].
```

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.
```

4 Utility Functions

```
split_input(Txt) ->
    [{T1,Ls} ||
        {T1,Ls} <-
            [{T,strip_empty(L1)} || {T,L1} <-
                                        group([wrap(L) || L <- lines(Txt)])],
        Ls =/= []].

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

lines("\n" ++ Str, Acc) ->
    [lists:reverse([$\n|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}.

strip_empty(Ls) ->
    Strip = fun(Ls1) ->
                    lists:dropwhile(fun(L) -> strip_space(L) == [] end, Ls1)
            end,
    lists:reverse(Strip(lists:reverse(Strip(Ls)))).
```

Remove leading empty lines, even if they contain whitespace.

```
strip_space(L) ->
    lists:dropwhile(fun(C) when C==$\s; C==$\t; C==$\n -> true;
                       (_) -> false
                    end, L).

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

latex_target(F, Options) ->


    Target_base = strip_extension(F) ++ ".tex",
    Outdir = proplists:get_value(outdir, Options, filename:dirname(F)),
    filename:join(Outdir, Target_base).

strip_extension(F) ->
    case regexp:split(F, "\.[a-z]+$") of
        {ok, [F1,[]]} ->
            F1
    end.

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

embedded_options(Parts) ->
    lists:mapfoldl(
      fun({code,Ls}=Part,Acc) ->
              case scan_for_opts(Ls) of
                  none       -> {Part,Acc};
                  {Opts,Ls1} -> {{code,Ls1}, Acc ++ Opts}
              end;
         (Other, Acc) ->
              {Other, Acc}
      end, [], Parts).

scan_for_opts(Ls) ->
    scan_forms(Ls, [], []).

scan_forms(Ls, Opts0, Acc) ->
    case scan_tokens(Ls) of
        {{ok,[{'-',_},{atom,_,erl2latex},{'(',L}|Ts], _}, Used, Rest} ->
            case erl_parse:parse_term([{'(',L}|Ts]) of
                {ok, Opts} -> scan_forms(Rest, Opts0++Opts, Acc);
                {error,_}  -> scan_forms(Rest, Opts0, Acc ++ Used)
            end;
        {{eof,_}, Used, []} ->
            case Opts0 of
                [_|_] -> {Opts0, Acc ++ Used};
                []    -> none
            end;
        {_, Used, Rest} ->
            scan_forms(Rest, Opts0, Acc ++ Used)
    end.

scan_tokens([L|Ls]) ->
    scan_tokens(erl_scan:tokens([],L,1), Ls, [L]).

scan_tokens({done,Result,Leftover_chars},Rest,Used) ->
    {Result, lists:reverse(Used), [Leftover_chars|Rest]};
scan_tokens({more, Cont}, Ls, Used) ->
    case Ls of
        [] ->
            {{eof,1}, lists:reverse(Used), []};
        [L|Rest] ->
            scan_tokens(erl_scan:tokens(Cont, L, 1), Rest, [L|Used])
    end.

rearrange_if_escript([{code, ["#!" ++ _ = Head]},
                      {comment, ["! -" ++ _ = Xtra]},
                      {comment, _} = Cmt|Rest]) ->
    Code = {code, [Head, "%%" ++ Xtra]},


    [Cmt, Code | Rest];
rearrange_if_escript([{code, ["#!" ++ _|_]} = Head,
                      {comment, _} = Cmt|Rest]) ->
    [Cmt, Head | Rest];
rearrange_if_escript(Other) ->
    Other.
```