Gradual Typing of Erlang Programs: A Wrangler Experience

Kostis Sagonas
(paper co-authored with Daniel Luna)
Overview

This talk/paper:

- Aims to document and promote a different mode of Erlang program development:
  
  • one where most typos, interface abuses, type errors, etc. are identified automatically using static analyzers
  
  • one where type information becomes part of the code and checked for definite violations after program modifications
  
  • one where all the above are optional, can take place gradually, and can be refined at any point to the extent desired by the programmer
Practice and experience

- We have been practicing this development mode in large Erlang code bases:
  - dialyzer
  - typer
  - hipe (a very large part)
  - stdlib & kernel (many key modules)
- But wanted to also try it in code with which we were not familiar – hence this paper
Why Wrangler?

• Open source with many releases
• Developed by experts in typed FP
  − Expected it would be written in a type disciplined manner
  − Expected it would not contain (m)any type errors
  − Many higher-order functions – challenging for tools
• Authors of Wrangler aware of our tools
Step #1

Use Dialyzer
Wrangler 0.1

- Released January 2007
- 25 modules
- 35,000 lines of code
- Many modules are slight modifications or clones of Erlang/OTP ones – mainly of `syntax_tools`
Dialyzer on Wrangler 0.1

• Run as simply as

```
> cd distel-wrangler-0.1/wrangler
> dialyzer --src -c *.erl
```

• 67 warnings in less than 2 minutes
• about 50 of them due to abuse of `file:open/2`

```
file:open(Name,read) VS. file:open(Name,[read])
```

• After fixing this and one similar interface abuse, 15 warnings remain
  - all genuine bugs
handle_call(Call, DefinedVars, State) ->
  ...  
  case is_c_atom(Mod) andalso is_c_atom(Fun) of  
    true ->
      M = atom_val(Mod),
      ...
    case {{M_Loc, Call_Loc} of
        {{L1, C1}, {L2, C2}} ->
          if (L1 < L2) or
            ((L1==L2) and ((C2-C1) > length(M)))
        ...
    end

refac_atom_info.erl:715:
Guard test length(M::atom()) can never succeed
Can you spot the bug?

get_new_name(Sub, NewRegExp) ->
    Index = string:indexOf(NewRegExp, "*") - 1,
    case Index of
        0 -> NewRegExp;
        N ->
            Prefix = string:substr(NewRegExp, 1, N-1),
            case Sub of
                [] -> exit(error, "Cannot infer ..."),
                _ -> Sub1 = hd(Sub),
                    get_new_name(tl(Sub), Prefix++Sub1++...)
            end
    end.

refac_batch_rename_mod.erl:161:
The call erlang:exit('error',string()) will fail since it differs in argument 1 from the success typing arguments (pid() | port(),string())
Can you spot the bug?

```erlang
expand_files([File|Left], Ext, Acc) ->
  case filelib:is_dir(File) of
    true ->
      ...
    false ->
      case filelib:is_regular(File) and
              filename:extension(File) == Ext of
        true -> expand_files(Left, Ext, [File|Acc]);
        false -> expand_files(Left, Ext, [File])
      end
  end;
end;
```

**refac_util.erl:1322:**

The call `erlang:and(bool(),[integer()])` will fail since it differs in argument position 2 from the success typing arguments: `(bool(),bool())`
Wrangler 0.3

- Released January 2008 – one year after 0.1
- 25 modules
- 27,000 lines of code
Dialyzer on Wrangler 0.3

• Run as simply as

```
> cd distel-wrangler-0.3/wrangler/erl
> dialyzer --src -I ../hrl -c *.erl
```

• Analysis takes 50 secs – produces many warnings

• Many due to `file:open/2` and due to confusing `lists:concat/1` with `lists:append/1`

• After fixing these, 10 warnings remain
  - all genuine bugs
  - two of them are remains from Wrangler 0.1
  - not very surprising: they are in uncommon code paths
Step #2

Expose type information: make it part of the code
Exposing type information

Can happen in either of the following ways:

- **Add explicit type guards in key places in the code**
  - Ensures the validity of the information
  - Has a runtime cost – typically small
  - Programs may not be prepared to handle failures

- **Add type declarations and contracts**
  - Documents functions and module interfaces
  - Incurs no runtime overhead
  - Can be used by dialyzer to detect contract violations
Turning @specs into -specs

Often Edoc @spec annotations

```erlang
%% @spec batch_rename_mod(OldNamePattern::string(),
%%                       NewNamePattern::string(),
%%                       SearchPaths::[string()]) ->
%%                      ok | {error, string()}
```

Can easily be turned into -spec declarations

```erlang
-spec batch_rename_mod(OldNamePattern::string(),
                       NewNamePattern::string(),
                       SearchPaths::[string()]) ->
                      'ok' | {'error', string()}. 
```
Turning @specs into -specs

In some other cases

```erlang
%% @spec duplicated_code(FileName ::filename(),
% %              MinLines ::integer(),
%%              MinClones::integer()) -> term()
```

Type declarations are also required

```erlang
-type filename() :: string().
-spec duplicated_code(FileName ::filename(),
            MinLines ::integer(),
            MinClones::integer()) -> term().
```
Turning \texttt{@specs} into \texttt{-specs}

A problem with Edoc annotations is that often they are not in accordance with the code

- Not surprising – they are comments after all!

For example, to be correct, let alone precise, the previous case should read:

\begin{verbatim}
-type filename() :: string().
-spec duplicated_code(FileNames::[filename()],
                     MinLines :: [integer()],
                     MinClones::[integer()]) -> term().
\end{verbatim}
How to turn @specs into −specs

Option 1: Convert @specs into −specs in one go
- Brave and quick
- Typically not a good idea: results in many Dialyzer warnings which may be hard to debug

Experiment: 162 warnings on the code of Wrangler 0.3

Option 2: Convert @specs gradually and fix the erroneous ones using Dialyzer
- First locally (on a module-by-module basis)
- Then globally

→ We strongly recommend Option 2
Wrong `@specs` in Wrangler 0.3

<table>
<thead>
<tr>
<th>module</th>
<th>@specs</th>
<th>local</th>
<th>global</th>
</tr>
</thead>
<tbody>
<tr>
<td>refac_batch_rename_mod</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>refac_duplicated_code</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>refac_expr_search</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>refac_fold_expression</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>refac_gen</td>
<td>7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>refac_move_fun</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>refac_new_fun</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>refac_rename_fun</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>refac_rename_mod</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>refac_rename_var</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>refac_util</td>
<td>21</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>wrangler</td>
<td>11</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Wrong `@specs` in Wrangler 0.3; blank entries denote 0
Step #3

Fix bugs exposed by `-spec` declarations
Step #4

Strengthen and factor -type declarations
Strengthening -type declarations

• Type declarations can be refined to the extent desired by the programmer

- type pos() :: any().
- type pos() :: tuple().
- type pos() :: {any(), any()}.  
- type pos() :: {number(), number()}. 
- type pos() :: {integer(), integer()}. 
- type pos() :: {0..1000000, 0..200}. 
Step #5

Strengthen underspecified `-spec declarations
Strengthening underspecified \texttt{-specs}

Can take place semi-automatically using Dialyzer

\begin{verbatim}
> dialyzer -Wunderspecs --src -I ../hrl -c *.erl
\end{verbatim}

\texttt{refac\_duplicated\_code.erl:53:}

Type specification for \texttt{duplicated\_code/3} ::

\begin{verbatim}
([filename()], [integer()], [integer()]) \rightarrow \texttt{term()}
\end{verbatim}

is a supertype of the success typing:

\begin{verbatim}
([string()], [integer()], [integer()]) \rightarrow \{'\texttt{ok}',\texttt{string()}\}
\end{verbatim}
Step #6

Add `-spec` declarations for all exported functions
Adding missing \texttt{-specs}

Can take place semi-automatically using Typer

\begin{verbatim}
> erlc +warn_missing_spec -I../hrl refac_rename_var.erl
./refac_rename_var.erl:166: Warning:
  missing specification for function pre_cond_check/4

> typer --show-exported -I../hrl refac_rename_var.erl

%% File: "refac_rename_var.erl"
%% ----------------------------
-spec pre_cond_check(tuple(),integer(),integer(),atom()) -> bool().
-spec rename(syntaxTree(),pos(),atom()) -> {syntaxTree(),bool()}.  
-spec rename_var(filename(),...,[string()]) ->
    {'ok',string()} | {'error',string()}.  
\end{verbatim}
## Missing `@specs` in Wrangler 0.3

<table>
<thead>
<tr>
<th>module</th>
<th>@specs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>present</td>
</tr>
<tr>
<td>refac_batch_rename_mod</td>
<td>1</td>
</tr>
<tr>
<td>refac_duplicated_code</td>
<td>1</td>
</tr>
<tr>
<td>refac_expr_search</td>
<td>1</td>
</tr>
<tr>
<td>refac_fold_expression</td>
<td>2</td>
</tr>
<tr>
<td>refac_gen</td>
<td>2</td>
</tr>
<tr>
<td>refac_module_graph</td>
<td></td>
</tr>
<tr>
<td>refac_move_fun</td>
<td>2</td>
</tr>
<tr>
<td>refac_new_fun</td>
<td>1</td>
</tr>
<tr>
<td>refac_rename_fun</td>
<td>1</td>
</tr>
<tr>
<td>refac_rename_mod</td>
<td>1</td>
</tr>
<tr>
<td>refac_rename_var</td>
<td>2</td>
</tr>
<tr>
<td>refac_util</td>
<td>21</td>
</tr>
<tr>
<td>wrangler</td>
<td>11</td>
</tr>
<tr>
<td>wrangler_distel</td>
<td></td>
</tr>
<tr>
<td>wrangler_options</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.** Number of existing and missing `specs` for all exported functions of Wrangler 0.3 modules; blank entries denote 0
Step #7

Test the validity of contracts using runtime monitoring
Testing for contract violations

Out of the 106 -spec declarations of Wrangler
- 55 were exercised by the test suite
- 4 of them were detected as erroneous
Concluding remarks

• Described a methodology for how to:
  – use static analysis for detecting definite type errors
  – add type information to existing Erlang applications
  – become confident about the validity of that information

• Showed both the benefits and common pitfalls of the approach on a non-trivial case study

• Type information is not a panacea but makes code more robust, easier to understand and maintain