Cleaning up Erlang code is a dirty job but somebody’s gotta do it

Kostis Sagonas

(joint work with Thanassis Avgerinos)
This talk

• Describes **tidier**, a software tool that:
  - Cleans up Erlang source code
  - Modernizes outdated language constructs
  - Eliminates certain bad code smells from programs
  - Improves performance of applications

• The paper:
  - Documents what we believe are good coding practices in Erlang
  - Reports experiences from real code bases
Characteristics of tidier

• Fully automatic
  - No user interaction required
    (Confirmation available as an option)

• Reliable – never wrong
  - Semantics-preserving transformations

• Universal and easy to use
  - Not tied to some particular editor or IDE

• Flexible
  - Transformations are selectable by the user

• Fast
Properties of the transformations

- **Semantics preserving**
  - Transformations are conservative (more on that later)

- **Code improving**
  - Newer instead of an older/obsolete constructs
  - Smaller and/or more elegant code
  - Redundancy elimination
  - Performance improvement

- **Syntactically pleasing and natural**
  - Similar to what an expert Erlang programmer would have written if transforming the code by hand
Current set of transformations

• Simple transformations (inherited from \texttt{erl\_tidy})
• Record transformations
• List comprehension transformations
• Code simplifications and specializations
• Redundancy elimination transformations
• List comprehension simplifications
• Zip, unzip and deforestations
• Transformations improving runtime performance
Modernizing old guards & functions

- `atom(X)` $\Rightarrow$ `is_atom(X)`
- `integer(X)` $\Rightarrow$ `is_integer(X)`
- `unix:cmd(Cmd)` $\Rightarrow$ `os:cmd(Cmd)`
- `lists:append(L1,L2)` $\Rightarrow$ `L1 ++ L2`
- `lists:subtract(L1,L2)` $\Rightarrow$ `L1 -- L2`
Record transformations

process(St, Pid) when is_record(St, st),
    St#st.status =:= open,
    is_pid(Pid) ->
    inet_tcp:controlling_process(St#st.proxysock, Pid).

⇓

process(#st{} = St, Pid) when St#st.status =:= open,
    is_pid(Pid) ->
    inet_tcp:controlling_process(St#st.proxysock, Pid).

⇓

process(#st{status=Status, proxysock=Proxysock}, Pid)
    when Status =:= open, is_pid(Pid) ->
    inet_tcp:controlling_process(Proxysock, Pid).

⇓

process(#st{status = open, proxysock=Proxysock}, Pid)
    when is_pid(Pid) ->
    inet_tcp:controlling_process(Proxysock, Pid).
List comprehension transformations

```
lists:map(fun dig_to_hex/1, lists:reverse(R))
```

⇓

```
[dig_to_hex(V) || V <- lists:reverse(R)]
```

```python
lists:map(fun (X) -> X + 42 end, L)
```

⇓

```
[X + 42 || X <- L]
```
List comprehension transformations

```erlang
lists:filter(fun (X) ->
    is_integer(X) andalso X > 0
    end, L)
```

\[\Rightarrow\]

```erlang
[X || X <- L, is_integer(X), X > 0]
```

```erlang
lists:filter(fun ({N,_,_}) when N == Name -> true;
    (_) -> false
    end, L)
```

\[\Rightarrow\]

```erlang
[T || T = {N,_,_} <- L, N == Name]
```
Transformations avoiding redundancy

- Specialization of size/1
- Simplifying guard sequences
- Structure reuse
- Straightening case expressions
- Simplifying case expressions
f(Rec, Fields, Key) when is_tuple(Rec), is_list(Fields),
    size(Rec) - 1 =:= length(Fields) ->
    lists:zip([Key | Fields], tuple_to_list(Rec)).

↓

f(Rec, Fields, Key) when tuple_size(Rec) - 1 =:= length(Fields) ->
    lists:zip([Key | Fields], tuple_to_list(Rec)).
case get_value(binary, Opts, case get(read_mode) of
    binary -> true;
    _    -> false
end) of
  true -> ...
is_pure_op(N, A) ->
    case is_bool_op(N, A) of
        true -> true;
        false ->
            case is_comp_op(N, A) of
                true -> true;
                false -> is_type_test(N, A)
            end
    end.

↓

is_pure_op(N, A) ->
    is_bool_op(N, A) orelse is_comp_op(N, A)
    orelse is_type_test(N, A).
t_charset(Fun, In) ->
    case lists:all(Fun, In) of
        true ->
            true;
        _ ->
            false
    end.
Simplifying list comprehensions

- Simplifying uses of `filter`
- Simplifying uses of `map`
- Simplifying `map + filter` combinations
- Simplifying uses of `zip` and `unzip`
Simplifying list comprehensions

\[
lf(X, \text{List}) \rightarrow \text{lists:filter(fun (Y) \rightarrow \\
    \text{if} \\
    \hspace{1em} X =:= Y \rightarrow \text{true}; \\
    \hspace{1em} \text{true} \rightarrow \text{false} \\
    \hspace{1em} \text{end} \\
    \hspace{1em} \text{end,} \\
    \hspace{1em} \text{List}).}
\]

\[
\downarrow
\]

\[
lf(X, \text{List}) \rightarrow \quad \lfloor \text{Y } | | \text{ Y <- List, X =:= Y} \rfloor.
\]
lists:filter(fun(Pid) when node(Pid) =:= Node -> false; (_,) -> true end, Pids)

\[ [\text{Pid} \mid \text{Pid} \leftarrow \text{Pids}, \text{node(Pid)} =/\= \text{Node}] \]
lists:filter(fun (I) ->
    case I of
      {xmlelement, _, _, _} -> true;
      _ -> false
    end
  end,
  Els)

⇓

[I || I = {xmlelement, _, _, _} <- Els]


```
lists:map(fun ({_, X}) -> X end,
    lists:filter(fun (X) ->
        case X of
          {atom, _X} -> true;
          _ -> false
        end
    end,
    R))

⇓

[X || {atom, X} <- R]
```
get_all_tracing_nodes_rtstates(RTStates) ->
    lists:map(fun ({N,_,_}) -> N end,
        lists:filter(fun ({_,{tracing,_,_}}) ->
            true;
            (_) -> false
        end,
            RTStates)).
event_filter(Key, EvList) ->
    Fun = fun ({{K, _}}) when K == Key ->
        true;
        (_, _) ->
            false
    end,
    {_, R} = lists:unzip(lists:filter(Fun, EvList)), R.

⇓

event_filter(Key, EvList) ->
    [V || {K, V} <- EvList, K == Key].
Transformations improving performance

Transforming uses of \texttt{length/1}
star(_Rule,XML,_,_WSa,Tree,_S) when length(XML) == 0 ->
  {[Tree],[]};
star(Rule,XMLS,Rules,WSaction,Tree,S) ->
  {WS,XMLS1} = whitespace_action(XMLS,WSaction),
  case parse(Rule,XMLS1,Rules,WSaction,S) of
    {error, _E, {{next,N},{act,A}}} -> {WS++Tree++A,N};
    {error, _E} ->
      case whitespace_action(XMLS,...)) of
        {[],_} -> {WS++[Tree],XMLS};
        {WS2,XMLS2} -> {WS2++[Tree],XMLS2}
      end;
    {Tree1,XMLS2} ->
      star(Rule,XMLS2,Rules,WSaction,Tree++WS++[Tree1],S)
  end.
star(_Rule,XML,_,_WSa,Tree,_S) when length(XML) == 0 ->  
 [[Tree],[[]]]; 
 star(Rule,XMLS,Rules,WSaction,Tree,S) ->  
 ... % recursive case of star function here ...  
    star(Rule,XMLS2,Rules,WSaction,Tree++WS++[Tree1],S) 
 end.

↓

star(_Rule,[][_WSa,Tree,_S) ->  
 [[Tree],[[]]]; 
 star(Rule,XMLS,Rules,WSaction,Tree,S) ->  
 ... % recursive case of star function here ...  
    star(Rule,XMLS2,Rules,WSaction,Tree++WS++[Tree1],S) 
 end.
splice(L) ->
    Res = splice(L, [], []),
    case (length(Res) == 1) andalso is_list(hd(Res)) of
        true -> no;
        _    -> {yes, Res}
    end.
The call

```erlang
lists:duplicate(length(case_clauses(T)), X)
```

Can be written more compactly and efficiently as

```erlang
[X || _ <- case_clauses(T)]
```
Conservatism of transformations

- Tidier preserves the operational semantics of Erlang programs
- The following transformations are not performed

```erlang
Functions = [E || E <- get_content(functions,Es)]

foo(Ps) -> lists:map(fun ({X,Y}) -> X + Y end, Ps)
```

```erlang
Functions = get_content(functions,Es)

foo(Ps) -> [X + Y || {X,Y} <- Ps].
```
Now what?

Demo time!
### Table 1. Number of tidier’s transformations on various Erlang source code bases.

<table>
<thead>
<tr>
<th>Lines of Code</th>
<th>New Guards</th>
<th>Exact Numeric Equality</th>
<th>Lists: <code>keysearch/3</code></th>
<th>Record Matches</th>
<th>Record Accesses</th>
<th>Size</th>
<th>Simplifying Guards</th>
<th>Structured Renaming</th>
<th>Straighten + Case Simplify</th>
<th>Map to Comprehension</th>
<th>Filter to Comprehension</th>
<th>Deforesting</th>
<th>Zip + Unzip</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erlang/OTP</td>
<td>1,240,000</td>
<td>2911</td>
<td>68</td>
<td>751</td>
<td>1805</td>
<td>2168</td>
<td>487</td>
<td>36</td>
<td>1467</td>
<td>77</td>
<td>564</td>
<td>115</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>CouchDB</td>
<td>20,500</td>
<td>22</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>27</td>
<td>31</td>
<td>2</td>
<td>88</td>
<td>3</td>
<td>38</td>
<td>11</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Disco</td>
<td>2,500</td>
<td>11</td>
<td>2</td>
<td>12</td>
<td>2</td>
<td>9</td>
<td>14</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>134</td>
<td>40</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Ejabberd</td>
<td>55,000</td>
<td>2</td>
<td>78</td>
<td>18</td>
<td>26</td>
<td>6</td>
<td>70</td>
<td>11</td>
<td>134</td>
<td>40</td>
<td>20</td>
<td>10</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Erlang Web</td>
<td>10,000</td>
<td>7</td>
<td>11</td>
<td>37</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>15</td>
<td>6</td>
<td>35</td>
<td>75</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RefactorErl</td>
<td>24,000</td>
<td>11</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>54</td>
<td>1</td>
<td>39</td>
<td>7</td>
<td>36</td>
<td>3</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Scalaris</td>
<td>35,000</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>22</td>
<td>39</td>
<td>22</td>
<td>31</td>
<td>20</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Wings 3D</td>
<td>112,000</td>
<td>10</td>
<td>13</td>
<td>45</td>
<td>1</td>
<td>24</td>
<td>26</td>
<td>166</td>
<td>11</td>
<td>25</td>
<td>10</td>
<td>4</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>Wrangler</td>
<td>42,000</td>
<td>6</td>
<td>28</td>
<td>141</td>
<td>1</td>
<td>1</td>
<td>110</td>
<td>7</td>
<td>236</td>
<td>47</td>
<td>44</td>
<td>5</td>
<td>14</td>
<td>2</td>
</tr>
</tbody>
</table>
I just ran a little demo here for ..., ..., ..., and .... Many laughs and comments like "whose code is that? Mine?!!" and a couple of "I didn't know you could write that like that". We're still on R12B-5 here for all our work, so most people don't have a working R13 on their computers. But I'd like to force everyone to set it up and run tidier on the code they are responsible for, as a learning experience for many of the more junior developers (and for some senior ones as well, apparently...).
Concluding remarks

- Described the details of **tidier**, a software tool that
  - Cleans up Erlang source code
  - Modernizes outdated language constructs
  - Eliminates certain bad code smells from programs
  - Improves performance of applications