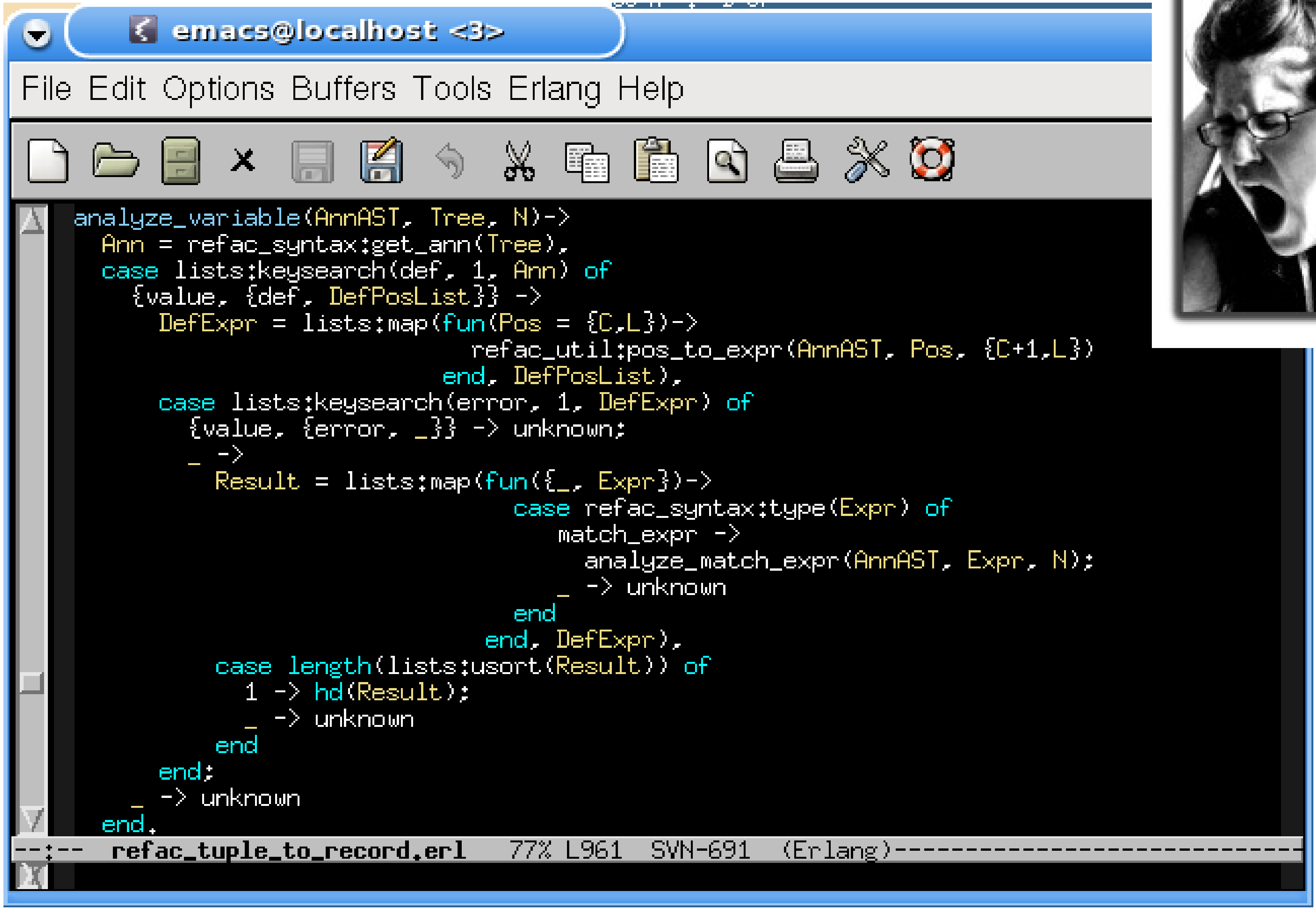
Cleaning up Erlang code is a dirty job but somebody's gotta do it

Kostis Sagonas

(joint work with Thanassis Avgerinos)

Erlang program development



```
analyze_variable(AnnAST, Tree, N)->
  Ann = refac_syntax:get_ann(Tree),
  case lists:keysearch(def, 1, Ann) of
    {value, {def, DefPosList}} ->
      DefExpr = lists:map(fun(Pos = {C,L})->
                              refac_util:pos_to_expr(AnnAST, Pos, {C+1,L})
                          end, DefPosList),
      case lists:keysearch(error, 1, DefExpr) of
        {value, {error, _}} -> unknown;
        _ ->
          Result = lists:map(fun({_, Expr})->
                                 case refac_syntax:type(Expr) of
                                   match_expr ->
                                     analyze_match_expr(AnnAST, Expr, N);
                                   _ -> unknown
                                 end
                             end, DefExpr),
          case length(lists:usort(Result)) of
            1 -> hd(Result);
            _ -> unknown
          end
      end;
    _ -> unknown
  end.
--:--  refac_tuple_to_record.erl    77% L961   SVN-691   (Erlang)-----------------------------------
```

This talk

- Documents what we believe are good coding practices in Erlang
- 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  - Modernizes outdated language constructs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
- 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 **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

Transformations from **erl_tidy**

Modernizing old guards and functions

Turning **`apply`**'s to remote calls

Turning **`fun`**'s into functions (lambda lifting)

Turning **`map`**'s and **`filter`**'s to list comprehensions

Modernizing old guards & functions

`atom(X)`	⇒	`is_atom(X)`
`integer(X)`	⇒	`is_integer(X)`
`unix:cmd(Cmd)`	⇒	`os:cmd(Cmd)`
`lists:append(L1,L2)`	⇒	`L1 ++ L2`
`lists:subtract(L1,L2)`	⇒	`L1 -- L2`

Modernizing `lists:keysearch/3`

- New function `lists:keyfind/3` in R13B instead of `{'value', tuple()} | 'false'` returns `tuple() | 'false'`

```
case lists:keysearch(copies, 1, Options) of
    {value, {copies, Copies}} ->
         lists:sublist(util:get_nodes(), Copies);
    false ->
          util:get_nodes()
end
```

⇓

```
case lists:keyfind(copies, 1, Options) of
    {copies, Copies} ->
         lists:sublist(util:get_nodes(), Copies);
    false ->
          util:get_nodes()
end
```

Modernizing `lists:keysearch/3`

- The transformation is not so straightforward (code from **lib/stdlib/src/supervisor.erl:800**)

```
case lists:keysearch(Child#child.name, Pos, Res) of
  {value, _} -> {duplicate_child, Child#child.name};
  _ -> check_startspec(T, [Child|Res])
end
```

⇓

```
case lists:keyfind(Child#child.name, Pos, Res) of
  false -> check_startspec(T, [Child|Res]);
  _ -> {duplicate_child, Child#child.name}
end
```

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 comprehensions of **erl_tidy**

```
mp(L) ->
    lists:map(fun ({X, Y}) -> X + Y;
                  (X) when is_integer(X) -> 2 * X
              end, L).
```

⇓

```
mp(L) -> [mp_1(V) || V <- L].

mp_1({X, Y}) -> X + Y;
mp_1(X) when is_integer(X) -> 2 * X.
```

List comprehensions of **tidier**

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

⇓

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

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

⇓

```
[X + 42 || X <- L]
```

List comprehensions of **erl_tidy**

```
flt(L) ->
    lists:filter(fun ({X, Y}) -> true;
                     (X) -> is_atom(X)
                 end, L).
```

⇓

```
flt(L) -> [V || V <- L, flt_1(V)].

flt_1({X, Y}) -> true;
flt_1(X) -> is_atom(X).
```

List comprehensions of **tidier**

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

⇓

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

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

⇓

```
[T || T = {N, _, _} <- L, N == Name]
```

Transformations avoiding redundancy

Specialization of **`size/1`**

Simplifying guard sequences

Structure reuse

Straightening **`case`** expressions

Simplifying **`case`** expressions

couchdb/src/mochiweb/mochiweb_util.erl:422

```
f(Rec, Fields, Key) when is_tuple(Rec), is_list(Fields),
                         size(Rec)-1 =:= length(Fields) ->
    lists:zip([Key|Fields], tuple_to_list(Rec)).
```

$\Downarrow$

```
f(Rec, Fields, Key) when tuple_size(Rec)-1 =:= length(Fields) ->
    lists:zip([Key|Fields], tuple_to_list(Rec)).
```

Structure reuse

```
t({X, [3, Y]}) ->
  case m:foo(X) of
    true ->
      [3, Y];
    false ->
      {X, [3, Y]}
 end.
```

⇒

```
t({X, [3, _Y] = L} = T) ->
  case m:foo(X) of
    true ->
      L;
    false ->
      T
 end.
```

lib/kernel/src/group.erl:368

```
case get_value(binary, Opts, case get(read_mode) of
                                  binary -> true;
                                  _ -> false
                                end) of
    true -> ...
```

⇓

```
case get_value(binary, Opts, get(read_mode) =:= binary) of
    true -> ...
```

lib/hipe/cerl/cerl_to_icode.erl:2370

```erlang
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.
```

⇓

```erlang
is_pure_op(N, A) ->
    is_bool_op(N, A) orelse is_comp_op(N, A)
                     orelse is_type_test(N, A).
```

lib/xmerl/src/xmerl_ucs.erl:549

```
t_charset(Fun, In) ->
    case lists:all(Fun, In) of
        true ->
            true;
        _ ->
            false
    end.
```

⇓

```
t_charset(Fun, In) ->
    lists:all(Fun, In).
```

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, List) ->
  lists:filter(fun (Y) ->
                 if
                    X =:= Y -> true;
                    true -> false
                 end
               end,
               List).
```

$\Downarrow$

```
lf(X, List) ->
  [Y || Y <- List, X =:= Y].
```

lib/kernel/src/pg2.erl:280

```
lists:filter(fun(Pid) when node(Pid) =:= Node -> false;
                (_) -> true
             end,
             Pids)
```

⇓

```
[Pid || Pid <- Pids, node(Pid) =/= Node]
```

src/web/ejabberd_http_bind.erl:956

```
lists:filter(fun (I) ->
                case I of
                  {xmlelement, _, _, _} -> true;
                  _ -> false
                end
             end,
             Els)
```

⇓

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

wrangler/src/refac_rename_fun.erl:344

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

⇓

```
[X || {atom, X} <- R]
```

lib/inviso/src/inviso_tool_sh.erl:1638

```erlang
get_all_tracing_nodes_rtstates(RTStates) ->
  lists:map(fun ({N,_,_}) -> N end,
            lists:filter(fun ({_,{tracing,_},_}) ->
                                true;
                             (_) -> false
                         end,
                         RTStates)).
```

⇓

```erlang
get_all_tracing_nodes_rtstates(RTStates) ->
  [N || {N,{tracing,_},_} <- RTStates].
```

wrangler/src/refac_annotate_pid:274

```
lists:map(fun ({A, P}) -> F(A, P) end
               lists:zip(Args, ParSig))
```

$\Downarrow$

```
[F(A, P) || {A, P} <- lists:zip(Args, ParSig)]
```

$\Downarrow$

$\Downarrow$

```
[F(A, P) || A <- Args && P <- ParSig]
```

disco-0.2/master/src/event_server.erl:123

```
event_filter(Key, EvLst) ->
    Fun = fun ({K, _}) when K == Key ->
                    true;
                (_) ->
                    false
            end,
    {_, R} = lists:unzip(lists:filter(Fun, EvLst)),
    R.
```

⇓

```
event_filter(Key, Event) ->
    [V || {K, V} <- EvList, K == Key].
```

Transformations improving performance

Transforming uses of **`length/1`**

lib/xmerl/src/xmerl_validate.erl:542

```
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.
```

lib/xmerl/src/xmerl_validate.erl:542

```
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.
```

refactorerl/src/referl_expression.erl:384

```
side_effect(Expr)->
  Children  = ?Query:exec(Expr, ?Expr:deep_sub()),
  SideEffs  = [Node || Node <- Children,
                       (?Expr:kind(Node) == send_expr) orelse
                       (?Expr:kind(Node) == receive_expr)],
  Funs      = ?Query:exec(Expr, ?Expr:functions()),
  DirtyFunc = [Fun || Fun <- Funs, ?Fun:dirty(Fun)],
  UnKnown   = [Fun || Fun <- Funs, ?Fun:dirty(Fun) == unknown],
  case {length(SideEffs) /= 0, length(UnKnown) /=0,
        length(DirtyFunc) /= 0} of
    {true, _, _} -> true;
    {_, true, _} -> true;
    %% egyenlore ha nem tudjuk eldonteni, hogy van-e mellekha..
    %% akkor ugy tekintunk ra mintha lenne, kesobb esetleg meg
    %% lehet kerdezni a felhasznalaot, hogy szerinte van-e
    {_, _, true} -> true;
    {_, _,    _} -> false
  end.
```

org.erlide.core/erl/pprint/erlide_pperl.erl:171

```
splice(L) ->
  Res = splice(L, [], []),
  case (length(Res) == 1) and is_list(hd(Res)) of
    true -> no;
    _ -> {yes, Res}
  end.
```

$\Downarrow$

```
splice(L) ->
  Res = splice(L, [], []),
  case Res of
    [Res1] when is_list(Res1) -> no;
    _ -> {yes, Res}
  end.
```

lib/hipe/cerl/cerl_typean.erl:446

```
'case' ->
  {X, St1} = visit(case_arg(T), Env, St),
  Xs =
    case t_is_any(X) orelse t_is_none(X) of
      true ->
        lists:duplicate(length(case_clauses(T)), X);
      false ->
        t_to_tlist(X)
    end
```

- The call

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

- Can be written more compactly and efficiently as

```
[X || _ <- case_clauses(T)]
```

Conservatism of transformations

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

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

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

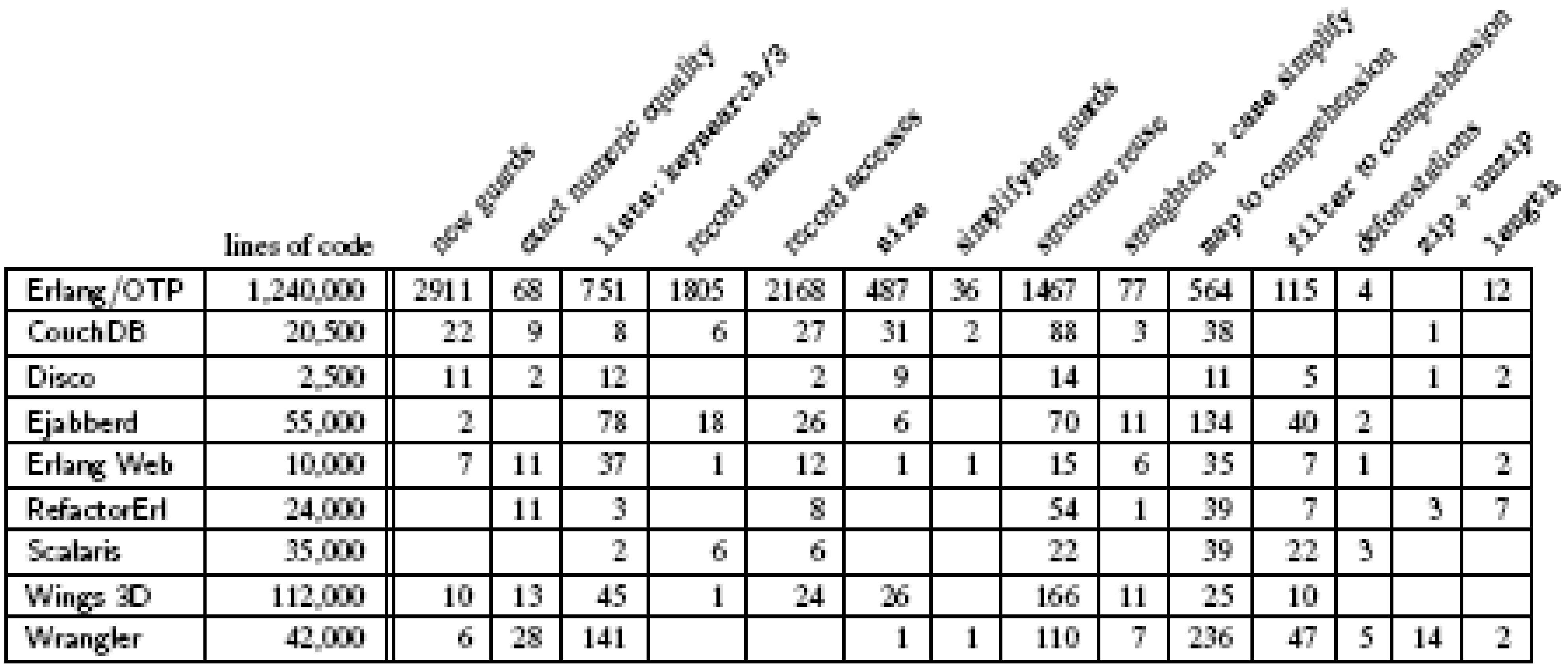
```
foo(Ps) -> lists:map(fun ({X,Y}) -> X + Y end, Ps)
```

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

Now what?

Demo time!

Some numbers (by now old)



	lines of code	new guards	exact numeric equality	lists:keysearch/3	record matches	record accesses	size	simplifying guards	structure reuse	straighten + case simplify	map to comprehension	filter to comprehension	deforestations	zip + unzip	length
Erlang/OTP	1,240,000	2911	68	751	1805	2168	487	36	1467	77	564	115	4		12
CouchDB	20,500	22	9	8	6	27	31	2	88	3	38			1	
Disco	2,500	11	2	12		2	9		14		11	5		1	2
Ejabberd	55,000	2		78	18	26	6		70	11	134	40	2		
Erlang Web	10,000	7	11	37	1	12	1	1	15	6	35	7	1		2
RefactorErl	24,000		11	3		8			54	1	39	7		3	7
Scalaris	35,000			2	6	6			22		39	22	3		
Wings 3D	112,000	10	13	45	1	24	26		166	11	25	10			
Wrangler	42,000	6	28	141			1	1	110	7	236	47	5	14	2

Table 1. Number of tidier's transformations on various Erlang source code bases.

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