Chemanalysis: Dialyzing Elixir

Sean Cribbs

1

What is Dialyzer?

2.1

Did you know…

2.2

Elixir Has Types!

2.3

Elixir Has Types!

They're in the Elixir guides!

2.3

Elixir Has Types!

They're in the Elixir guides!

```
@spec days_since_epoch(year :: integer, month :: integer, day :: integer) :: integer
@type color :: {red :: integer, green :: integer, blue :: integer}
```

2.3

Types Help You Find Bugs!

```
Unknown type: Elixir.Map:t/0

lib/myapp/event_logger.ex:38: Expression produces a value of type 'ok' | {'error',_},
but this value is unmatched

lib/collectable.ex:1: The specification for 'Elixir.Collectable':'__protocol__'/1
states that the function might also return 'true' but the inferred return is
'Elixir.Collectable' | 'false' | [{'into',1},...]
```

2.4

Type Annotations are Optional!

👍 No compilable programs are rejected

2.5

Type Annotations are Optional!

👍 No compilable programs are rejected

😭 Bugs slip through!

2.5

Types can be inferred!

```
defmodule Inference do
  def ignore_return do
    two_types()
    :ok
  end

  def two_types do
    if rem(System.monotonic_time(), 2) == 0 do
      {:ok, 2}
    else
      :odd
    end
  end
end
```

```
lib/inference.ex:3: Expression produces a value of type 'odd' | {'ok',2},
but this value is unmatched
```

2.6

Dialyzer automates checking types

2.7

Dialyzer automates checking types

- Figures out what your code calls

2.7

Dialyzer automates checking types

- Figures out what your code calls
- Computes the "success typing" of your program

2.7

Dialyzer automates checking types

- Figures out what your code calls
- Computes the "success typing" of your program
- Checks for known classes of type errors

2.7

Error Classes

Defaults (`-Wno_`)	Optional (`-W`)
`behaviours`	`error_handling`
`contracts`	`race_conditions`
`fail_call`	✅ `underspecs`
`fun_app`	✅ `unknown`
`improper_lists`	✅ `unmatched_returns`
`match`	`overspecs`
`missing_calls`	`specdiffs`
`opaque`	
`return`	
`undefined_callbacks`	
`unused`	

2.8

underspecs

Have you ever seen or written a spec like this?

```
@spec myfun(any()) :: any()
```

2.9

unknown

```
Unknown type: Elixir.Map:t/0
```

2.10

unmatched_returns

```
lib/inference.ex:3: Expression produces a value of type 'odd' | {'ok',2},
but this value is unmatched
```

2.11

unmatched_returns

```
lib/inference.ex:3: Expression produces a value of type 'odd' | {'ok',2},
but this value is unmatched
```

```
Logger.debug("result of call was #{inspect(result)}")
```

2.11

Dialyzer + Elixir

How do I run dialyzer against my Elixir code?

3.1

Dialyzer + Elixir

How do I run dialyzer against my Elixir code?

```
$ mix dialyzer
```

3.1

The Process

3.2

The Process

- Compile the project

3.2

The Process

- Compile the project
- Check or build PLT(s)

3.2

The Process

- Compile the project
- Check or build PLT(s)
- Analyze your code

3.2

The Process

- Compile the project
- Check or build PLT(s)
- Analyze your code
- Print warnings (possibly filtered)

3.2

Persistent Lookup Tables (PLTs)

MFA	Type
`Kernel.round/1`	`((integer() | float()) -> integer())`
`Map.key/0`	`any()`
`:queue.queue/0`	`:queue.queue(term())`

📦 Function + Type Index

3.3

Persistent Lookup Tables (PLTs)

MFA	Type
`Kernel.round/1`	`((integer() | float()) -> integer())`
`Map.key/0`	`any()`
`:queue.queue/0`	`:queue.queue(term())`

📦 Function + Type Index

🚀 Greatly speeds up analysis

3.3

`dialyxir`

Most popular, most mature

3.4

`dialyxir`

Most popular, most mature

Sophisticated PLT management

3.4

`dialyxir`

Most popular, most mature

Sophisticated PLT management

Elixir-friendly warnings

3.4

`dialyxir`

Most popular, most mature

Sophisticated PLT management

Elixir-friendly warnings

🙅‍♀️ Transitive dependencies not default in PLT

3.4

`dialyxir`

Most popular, most mature

Sophisticated PLT management

Elixir-friendly warnings

🙅‍♀️ Transitive dependencies not default in PLT

🤷‍♀️ Filter on strings or type/file/line

3.4

`dialyzex`

Strictest set of default warnings

3.5

`dialyzex`

Strictest set of default warnings

PLTs include all of OTP, Elixir, deps

3.5

`dialyzex`

Strictest set of default warnings

PLTs include all of OTP, Elixir, deps

Filter warnings using match-patterns

3.5

`dialyzex`

Strictest set of default warnings

PLTs include all of OTP, Elixir, deps

Filter warnings using match-patterns

😉 I wrote it

3.5

`dialyzex`

Strictest set of default warnings

PLTs include all of OTP, Elixir, deps

Filter warnings using match-patterns

😉 I wrote it

📦 Big PLTs

3.5

`dialyzex`

Strictest set of default warnings

PLTs include all of OTP, Elixir, deps

Filter warnings using match-patterns

😉 I wrote it

📦 Big PLTs

🙈 Not very popular

3.5

`mix_dialyzer`

🚨 Not production-ready, GSoC 2018

Elixir-friendly warnings (from dialyxir)

3.6

`mix_dialyzer`

🚨 Not production-ready, GSoC 2018

Elixir-friendly warnings (from dialyxir)

`.dialyzer.exs` config file

3.6

`mix_dialyzer`

🚨 Not production-ready, GSoC 2018

Elixir-friendly warnings (from dialyxir)

`.dialyzer.exs` config file

`dialyzer.info` task 💁‍♀️

3.6

.dialyzer.exs

```
[
  apps: [remove: [], include: []],
  warnings: [
    ignore: [
      {"non_existing", :*, :unknown_type},
      {"lib/dialyzer/config/config.ex", 73, :invalid_contract},
      {"lib/dialyzer/config/config.ex", 129, :invalid_contract},
    ],
    active: [
      :unmatched_returns,
      :error_handling,
      :unknown
    ]
  ],
  extra_build_dir: []
]
```

3.7

mix dialyzer.info

```
12:34 $ mix dialyzer.info

Welcome to mix dialyzer! A tool for integrating dialyzer into a project and analyzing discrepances.
Here are some infos about your system:

## Application name

* mix_dialyzer

## Applications included into analysis


  ### Erlang applications

  * erts
  * kernel
  * stdlib
  * crypto

  ### Elixir applications

  * elixir
  * mix

  ### Project applications

  * compiler
  * erlex
  * kernel
  * logger
  * pane
  * scribe
  * stdlib
  * erts
  * crypto
```

3.8

```
mix dialyzer.info
```

```
## Applications removed from analysis

□

## Warnings currently active

* unmatched_returns - Include warnings for function calls that ignore a structured return value or do not match
against one of many possible return value(s).

* error_handling - Include warnings for functions that only return by an exception.

* unknown - Let warnings about unknown functions and types affect the exit status of the command-line version. T
he default is to ignore warnings about unknown functions and types when setting the exit status. When using Dial
yzer from Erlang, warnings about unknown functions and types are returned; the default is not to return these wa
rnings.

## Warnings ignored

* error_handling - Include warnings for functions that only return by an exception.

* no_behaviours - Suppress warnings about behavior callbacks that drift from the published recommended interface
s.

* no_contracts - Suppress warnings about invalid contracts.

* no_fail_call - Suppress warnings for failing calls.

* no_fun_app - Suppress warnings for fun applications that will fail.

* no_improper_lists - Suppress warnings for construction of improper lists.

* no_match - Suppress warnings for patterns that are unused or cannot match.

* no_missing_calls - Suppress warnings about calls to missing functions.

* no_opaque - Suppress warnings for violations of opacity of data types.

* no_return - Suppress warnings for functions that will never return a value.
```

3.9

Chemanalysis

or… bugs I found in Elixir with Dialyzer

4 . 1

Bug 1

Fix dialyzer underspec warning with __protocol__/1: PR #5679

Fixed January 19, 2017 (released in Elixir 1.5)

4.2

Protocol code generation: `def`

```
quote do
  name  = unquote(name)
  arity = unquote(arity)

  @functions [{name, arity} | @functions]

  # Generate a fake definition with the user
  # signature that will be used by docs
  Kernel.def unquote(name)(unquote_splicing(args))

  # Generate the actual implementation
  Kernel.def unquote(name)(unquote_splicing(call_args)) do
    impl_for!(t).unquote(name)(unquote_splicing(call_args))
  end

  # ...
end
```

4.3

Protocol code generation: dispatch

```
Kernel.def impl_for!(data) do
  impl_for(data) || raise(Protocol.UndefinedError, protocol: __MODULE__, value: data)
end

Kernel.def impl_for(%{__struct__: struct}) when :erlang.is_atom(struct) do
  struct_impl_for(struct)
end

Kernel.defp struct_impl_for(struct) do
  target = Module.concat(__MODULE__, struct)
  case impl_for?(target) do
    true  -> target.__impl__(:target)
    false -> any_impl_for()
  end
end

Kernel.defp impl_for?(target) do
  Code.ensure_compiled?(target) and
    function_exported?(target, :__impl__, 1)
end
```

4.4

Protocol code generation: boilerplate

```
@doc false
@spec __protocol__(:module) :: __MODULE__
@spec __protocol__(:functions) :: unquote(Protocol.__functions_spec__(@functions))
@spec __protocol__(:consolidated?) :: boolean
Kernel.def __protocol__(:module), do: __MODULE__
Kernel.def __protocol__(:functions), do: unquote(:lists.sort(@functions))
Kernel.def __protocol__(:consolidated?), do: false
```

4.5

Protocol code generation: boilerplate

```
@doc false
@spec __protocol__(:module) :: __MODULE__
@spec __protocol__(:functions) :: unquote(Protocol.__functions_spec__(@functions))
@spec __protocol__(:consolidated?) :: boolean
Kernel.def __protocol__(:module), do: __MODULE__
Kernel.def __protocol__(:functions), do: unquote(:lists.sort(@functions))
Kernel.def __protocol__(:consolidated?), do: false
```

```
lib/collectable.ex:1: The specification for
  'Elixir.Collectable':'__protocol__'/1 states that the function might also
  return 'true' but the inferred return is 'Elixir.Collectable' | 'false' |
  [{'into',1},...]
```

4.5

Fixing underspecified `__protocol__/1`

```
-  @spec __protocol__(:consolidated?) :: boolean
+  @spec __protocol__(:consolidated?) :: false
```

4.6

Fixing underspecified `__protocol__/1`

```
-  @spec __protocol__(:consolidated?) :: boolean
+  @spec __protocol__(:consolidated?) :: false
```

```
defp change_impl_for([{:attribute, line, :spec, {{:__protocol__, 1}, funspecs}} | t],
                     protocol, types, structs, is_protocol, acc) do
  newspecs = for spec <- funspecs do
    case spec do
      {:type, line, :fun, [{:type, _, :product, [{:atom, _, :consolidated?}]}, _]} ->
        {:type, line, :fun,
         [{:type, line, :product, [{:atom, 0, :consolidated?}]},
          {:atom, 0, true}]} # CHANGE HERE
      other -> other
    end
  end
  change_impl_for(t, protocol, types, structs, is_protocol,
                  [{:attribute, line, :spec, {{:__protocol__, 1}, newspecs}}|acc])
end
```

4.6

Bug 2

Remove underspecification warning for consolidated protocols: PR #6627

Fixed October 2, 2017 (released in Elixir 1.6)

4.7

Updated Protocol boilerplate

```
@spec __protocol__(:module) :: __MODULE__
@spec __protocol__(:functions) :: unquote(Protocol.__functions_spec__(@functions))
@spec __protocol__(:consolidated?) :: false
@spec __protocol__(:impls) :: :not_consolidated
Kernel.def __protocol__(:module), do: __MODULE__
Kernel.def __protocol__(:functions), do: unquote(:lists.sort(@functions))
Kernel.def __protocol__(:consolidated?), do: false
Kernel.def __protocol__(:impls), do: :not_consolidated
```

4.8

Changing the spec during consolidation

```
defp change_impl_for([{:attribute, line, :spec, {{:__protocol__, 1}, funspecs}} | tail]
                     protocol, types, structs, protocol?, acc) do
  new_specs = for spec <- funspecs do
    case spec do
      # ...
      # ADDED:
      {:type, line, :fun, [{:type, _, :product, [{:atom, _, :impls}]}, _]} ->
        {:type, line, :fun,
         [{:type, line, :product, [{:atom, 0, :impls}]},
          {:type, 0, :tuple,
           [{:atom, 0, :consolidated},
            {:type, 0, :list, [{:type, 0, :module, []}]}]}]}
      # ...
```

4.9

Changing the spec during consolidation

```
defp change_impl_for([{:attribute, line, :spec, {{:__protocol__, 1}, funspecs}} | tail]
                     protocol, types, structs, protocol?, acc) do
  new_specs = for spec <- funspecs do
    case spec do
      # ...
      # ADDED:
      {:type, line, :fun, [{:type, _, :product, [{:atom, _, :impls}]}, _]} ->
        {:type, line, :fun,
         [{:type, line, :product, [{:atom, 0, :impls}]},
          {:type, 0, :tuple,
           [{:atom, 0, :consolidated},
            {:type, 0, :list, [{:type, 0, :module, []}]}]}]}
      # ...
```

```
@spec __protocol__(:impls) :: {:consolidated, [module()]}
```

4.9

Underspecified list of modules

```
'Elixir.Collectable':'__protocol__'('impls') -> {'consolidated',[module()]}
  ; ('consolidated?') -> 'true'
  ; ('functions') -> [{'into',1},...]
  ; ('module') -> 'Elixir.Collectable'
is a supertype of the success typing:
  'Elixir.Collectable':'__protocol__'('consolidated?' | 'functions' |
                                      'impls' | 'module') ->
    'Elixir.Collectable' |
    'true' |
    [{'into',1},...] |
    {'consolidated',['Elixir.BitString' | 'Elixir.File.Stream' |
                     'Elixir.HashDict' | 'Elixir.HashSet' |
                     'Elixir.IO.Stream' | 'Elixir.List' |
                     'Elixir.Map' | 'Elixir.MapSet',...]}
```

4 . 10

Fixing underspecification AGAIN!

```
            [{:type, line, :product, [{:atom, 0, :consolidated?}]},
             {:atom, 0, true}]}
         {:type, line, :fun, [{:type, _, :product, [{:atom, _, :impls}]}, _]} ->
+          impls = for mod <- types, do: {:atom, 0, mod}
           {:type, line, :fun,
            [{:type, line, :product, [{:atom, 0, :impls}]},
             {:type, 0, :tuple,
              [{:atom, 0, :consolidated},
-              {:type, 0, :list, [{:type, 0, :module, []}]}]}]}
+              {:type, 0, :list, [{:type, 0, :union, impls}]}]}]}
         other -> other
       end
     end
```

4 . 11

Bug 3

Guard test tuple_size(…) can never succeed warning: Issue #8157

Fixed by PR #8196 on September 14, 2018 (Released in Elixir 1.8)

4 . 12

The warning

```
Guard test tuple_size(__@5::#{'__exception__':='true', '__struct__':=_, _=>_})
  can never succeed
```

4 . 13

Prometheus.ex delegation macro

```
defmacro delegate(fun, opts \\ []) do
  fun = Macro.escape(fun, unquote: true)

  quote bind_quoted: [fun: fun, opts: opts] do
    target = Keyword.get(opts, :to, @erlang_module)

    {name, args, as, as_args} = Kernel.Utils.defdelegate(fun, opts)

    def unquote(name)(unquote_splicing(args)) do
      Prometheus.Error.with_prometheus_error(
        unquote(target).unquote(as)(unquote_splicing(as_args))
      )
    end
  end
end
```

4 . 14

Prometheus.ex error-handling macro

```
defmacro with_prometheus_error(block) do
  quote do
    try do
      unquote(block)
    rescue
      e in ErlangError ->
        reraise(
          Prometheus.Error.normalize(e),
          unquote(
            if macro_exported?(Kernel.SpecialForms, :__STACKTRACE__, 0) do
              quote(do: __STACKTRACE__)
            else
              quote(do: System.stacktrace())
            end
          )
        )
    end
  end
end
```

4 . 15

Elixir exception normalization

```
error:#{'__struct__' := __@4,'__exception__' := true} = __@3:___STACKTRACE__@1
    when __@4 == 'Elixir.ErlangError' ->
    _e@1 =
        'Elixir.Exception':normalize(
          error, __@3,
          case __@3 of
              __@5 when tuple_size(__@5) == 2 andalso
                        element(1, __@5) == badkey;
                       __@5 == undef;
                       __@5 == function_clause ->
                   ___STACKTRACE__@1;
              _ ->
                  []
          end),
```

4 . 16

Fixing the guard test

```
 dynamic_normalize(Meta, Var, [H | T]) ->
+  Generated = ?generated(Meta),
+
   Guards =
     lists:foldl(fun(Alias, Acc) ->
-      {'when', Meta, [erl_rescue_stacktrace_for(Meta, Var, Alias), Acc]}
-    end, erl_rescue_stacktrace_for(Meta, Var, H), T),
+      {'when', Generated, [erl_rescue_stacktrace_for(Generated, Var, Alias), Acc]}
+    end, erl_rescue_stacktrace_for(Generated, Var, H), T),

-  {'case', Meta, [
+  {'case', Generated, [
     Var,
     [{do, [
-      {'->', Meta, [[{'when', Meta, [Var, Guards]}], {'__STACKTRACE__', Meta, nil}]},
-      {'->', Meta, [[{'_', Meta, nil}], []]}
+      {'->', Generated, [[{'when', Generated, [Var, Guards]}], {'__STACKTRACE__', Gene
+      {'->', Generated, [[{'_', Generated, nil}], []]}
     ]}]
   ]}.
```

4 . 17

Conclusion

5.1

Conclusion

- Dialyzer is a useful tool for improving your code

5.1

Conclusion

- Dialyzer is a useful tool for improving your code
- Elixir integrates well with Dialyzer

5.1

Conclusion

- Dialyzer is a useful tool for improving your code
- Elixir integrates well with Dialyzer
- You can find bugs in Elixir!

5.1

Thanks!

https://seancribbs.com/presentations/chemanalysis-dialyzing-elixir.html

6.1