Chemanalysis: Dialyzing Elixir Sean Cribbs

What is Dialyzer?

Did you know...

Elixir Has Types!

Elixir Has Types!

They're in the Elixir guides!

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}
```

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},...]
```

Type Annotations are Optional!

de No compilable programs are rejected

Type Annotations are Optional!

de No compilable programs are rejected

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
```

• Figures out what your code calls

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

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

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	

underspecs

Have you ever seen or written a spec like this?

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

unknown

Unknown type: Elixir.Map:t/0

unmatched_returns

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

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)}")

Dialyzer + Elixir

How do I run dialyzer against my Elixir code?

Dialyzer + Elixir

How do I run dialyzer against my Elixir code?

\$ mix dialyzer

• Compile the project

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

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

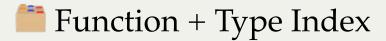
- Compile the project
- Check or build PLT(s)
- Analyze your code
- Print warnings (possibly filtered)

Persistent Lookup Tables (PLTs)

```
MFA Type

Kernel.round/1 ((integer() | float()) -> integer())

Map.key/0 any()
:queue.queue/0 :queue.queue(term())
```

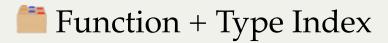


Persistent Lookup Tables (PLTs)

```
MFA Type

Kernel.round/1 ((integer() | float()) -> integer())

Map.key/0 any()
:queue.queue/0 :queue.queue(term())
```



Greatly speeds up analysis

Most popular, most mature

Most popular, most mature Sophisticated PLT management

Most popular, most mature
Sophisticated PLT management
Elixir-friendly warnings

Most popular, most mature Sophisticated PLT management Elixir-friendly warnings

Transitive dependencies not default in PLT

Most popular, most mature

Sophisticated PLT management

Elixir-friendly warnings

Transitive dependencies not default in PLT

Filter on strings or type/file/line

dialyzex

Strictest set of default warnings

dialyzex

Strictest set of default warnings
PLTs include all of OTP, Elixir, deps

dialyzex

Strictest set of default warnings

PLTs include all of OTP, Elixir, deps

Filter warnings using match-patterns

dialyzex

Strictest set of default warnings

PLTs include all of OTP, Elixir, deps

Filter warnings using match-patterns

6 I wrote it

dialyzex

Strictest set of default warnings

PLTs include all of OTP, Elixir, deps

Filter warnings using match-patterns

6 I wrote it



dialyzex

Strictest set of default warnings

PLTs include all of OTP, Elixir, deps

Filter warnings using match-patterns





Not very popular

mix_dialyzer

Not production-ready, GSoC 2018

Elixir-friendly warnings (from dialyxir)

mix_dialyzer

Not production-ready, GSoC 2018

Elixir-friendly warnings (from dialyxir)

.dialyzer.exs config file

mix_dialyzer

Not production-ready, GSoC 2018

Elixir-friendly warnings (from dialyxir)

.dialyzer.exs config file

dialyzer.info task 🎥

.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: []
]
```

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
```

mix dialyzer.info

Chemanalysis

or... bugs I found in Elixir with Dialyzer

Bug 1

Fix dialyzer underspec warning with __protocol__/1: PR #5679 Fixed January 19, 2017 (released in Elixir 1.5)

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
```

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
```

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
```

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},...]
```

Fixing underspecified __protocol___/1

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

Fixing underspecified protocol /1

end

Bug 2

Remove underspecification warning for consolidated protocols: PR #6627

Fixed October 2, 2017 (released in Elixir 1.6)

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
```

Changing the spec during consolidation

Changing the spec during consolidation

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

Underspecified list of modules

Fixing underspecification AGAIN!

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)

The warning

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

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
```

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. Special Forms, : STACKTRACE , 0) do
              quote(do: __STACKTRACE__)
            else
              quote(do: System.stacktrace())
            end
    end
 end
end
```

Elixir exception normalization

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}], []]}
    1}1
  ] } .
```

• Dialyzer is a useful tool for improving your code

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

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

Thanks!

https://seancribbs.com/presentations/chemanalysis-dialyzingelixir.html