WHAT DOES DIALYZER THINK OF ~~ME~~ YOU?

Stavros Aronis
Developer & Trainer @ Erlang Solutions

@Vahnatai
stavros.aronis@erlang-solutions.com

Erlang SOLUTIONS

#CodeBEAMSTO

About me:

- Student of Kostis Sagonas
- Master thesis on Dialyzer
- PhD in CS (**not** type systems)
- 9yrs Erlang, 1yr “industrially”
- https://github.com/parapluu/Concuerror

Goals

Therapy

How to control hatred towards a piece of software?

Dialyzer

How Dialyzer thinks?

Productivity

Why should you add Dialyzer to your build pipeline, today?

Dialyzer?

- Static analysis tool, included in OTP
- Works on single modules or entire applications
- Produces sound warnings (no false positives)
  - “Dialyzer is never wrong”
- Finds definite type errors
- Finds unreachable code
- Finds unnecessary tests
- etc.

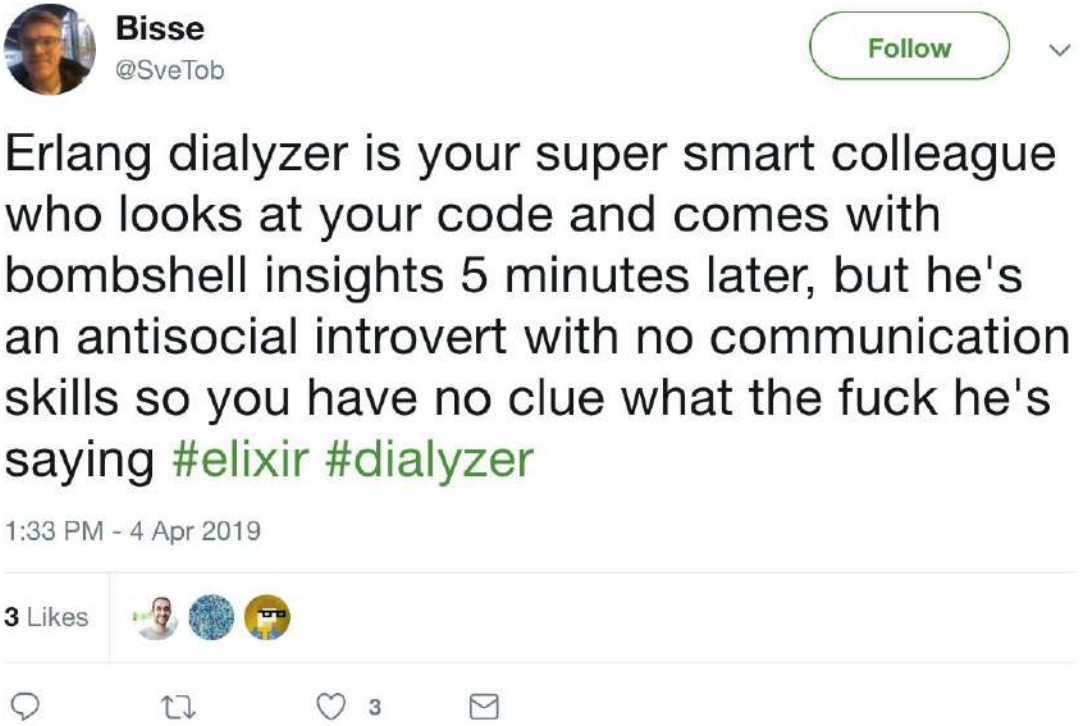
1.

WHAT DO YOU THINK OF DIALYZER?

Erlang SOLUTIONS

Charisma

Charisma



Charisma

Charisma

How does Dialyzer think?

How does Dialyzer think?

```
foo(Arg) ->
  …
```

Types: sets of values

(e.g. “3.5”,
“1 | 2 | 3”,
“atom()”,
“term()”,
“{term(), list()}”,
)

How does Dialyzer think?

```
foo(Arg) ->
  …
```

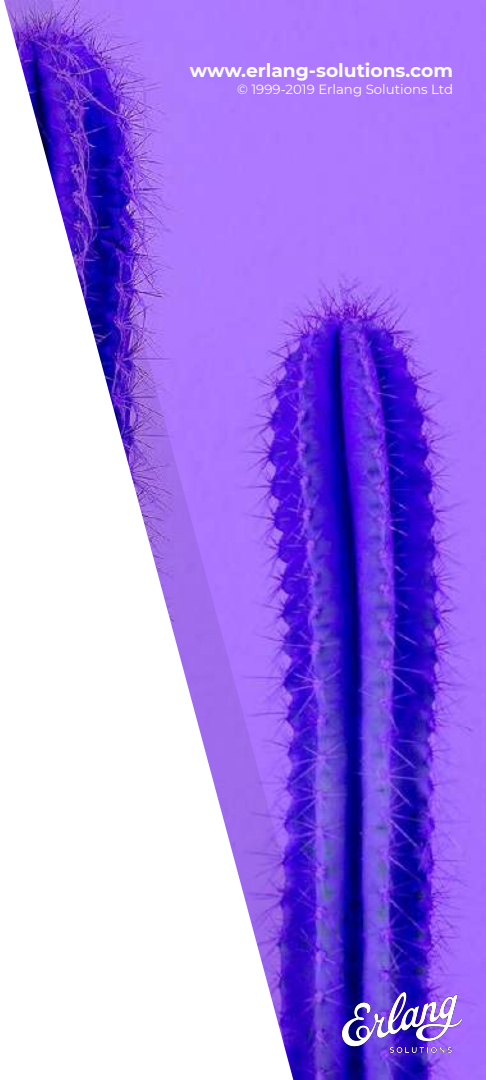
Subtype relationship:

```
'foo' ⊆ 'foo' | 'bar'
      ⊆ atom()
      ⊆ atom() | list()
      ⊆ term()
```

How does Dialyzer think?

```
foo(Arg) ->
  ...
```

`Arg`'s type: `term()`.



How does Dialyzer think?

```
foo(Arg) ->
  self() ! Arg,
  …
```

`Arg`'s type: `term()`.

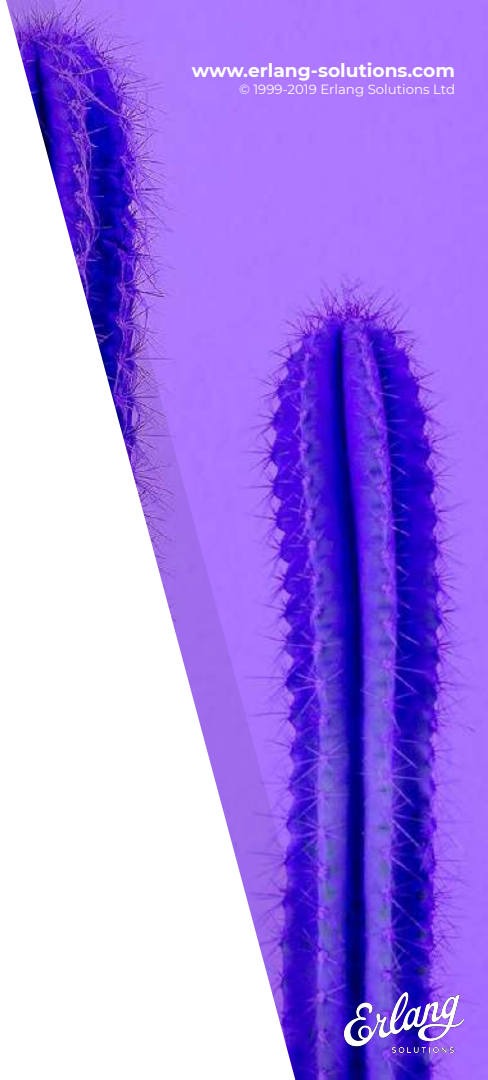
Any value can be sent as a message.

How does Dialyzer think?

```
foo(Arg) ->
  self() ! Arg,
  self() ! Arg + 1,
  …
```

`Arg`'s type: `number()`.

For "+" to succeed, `Arg` needs to be a number.



How does Dialyzer think?

```
foo(Arg) ->
  self() ! Arg,
  self() ! Arg + 1,
  Bar = Arg rem 2,
  …
```

Arg's type: `integer()`.

For "`rem`" to succeed, `Arg` needs to be an integer.

`Bar`'s type: `integer()`.

How does Dialyzer think?

```
foo(Arg) ->
  self() ! Arg,
  self() ! Arg + 1,
  Bar = Arg rem 2,
  Baz = Arg + Bar,
  …
```

Arg's type: `integer()`.

Bar's type: `integer()`.

Baz's type: `integer()`.

And not number!

How does Dialyzer think?

```
foo(Arg) ->
  self() ! Arg,
  self() ! Arg + 1,
  Bar = Arg rem 2,
  Baz = Arg + Bar,
  Baz ! Bar,
  …
```

Arg's type: `integer()`.

Bar's type: `integer()`.

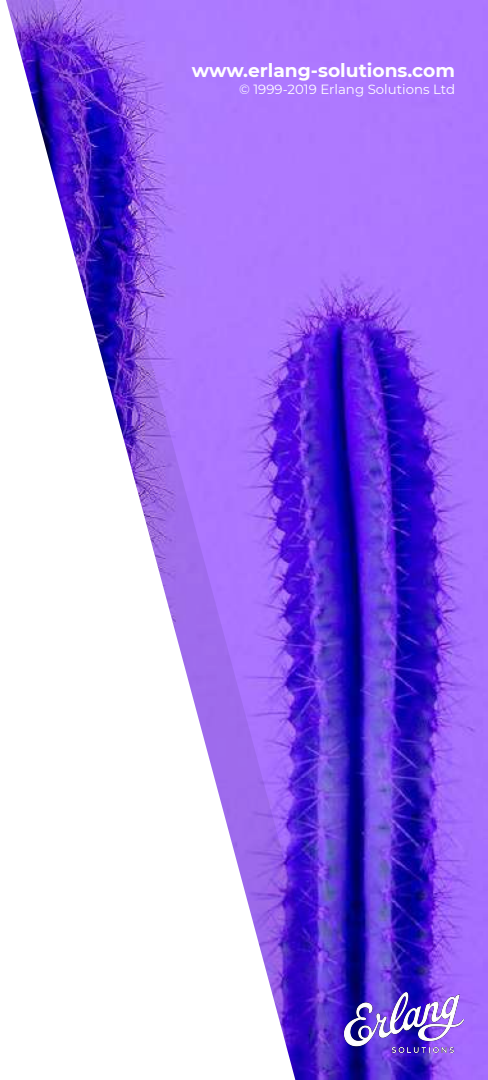
Baz's type: `none()`.

There's no "common" subtype of integer and any acceptable argument for erlang:"!".

How does Dialyzer think?

foo.erl:5: Function foo/1 has no local return
foo.erl:10: The call erlang:'!'(Baz::integer(),Bar::-1 | 0 | 1) will never return since it differs in the 1st argument from the success typing arguments: (atom() | pid() | port() | {atom(),atom()},any())

```
foo(Arg) ->
  self() ! Arg,
  self() ! Arg + 1,
  Bar = Arg rem 2,
  Baz = Arg + Bar,
  Baz ! Bar,
  …
```

How does Dialyzer know types?

Dialyzer trusts none!

- … except for Erlang primitives
- Also collects and uses any specs it finds
- Needs to pre-analyse any code that **you** trust
  - e.g. OTP, dependencies, etc.
  - saves the result in a lookup table (PLT)

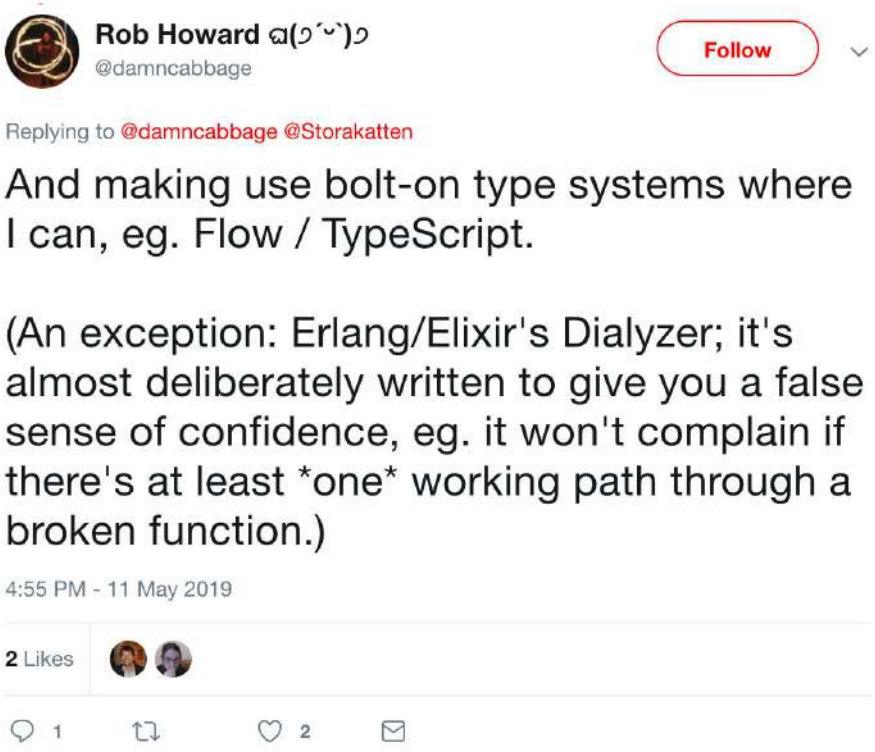
How does Dialyzer think?

- Types: sets of values (e.g. “1 | 2”, “atom()”, “any()”).
- Dialyzer’s type inference:
  - Assumes that variables can have any value
  - Finds where each variable is used
  - Constraints its values so that its use is ok
    - bottom-up based analysis
    - values must be subtypes of “success typings”
  - Refining:
    - dataflow-based analysis
    - for non-exported function arguments
    - based on local calls

2.

MORE THOUGHTS?

Commitments?

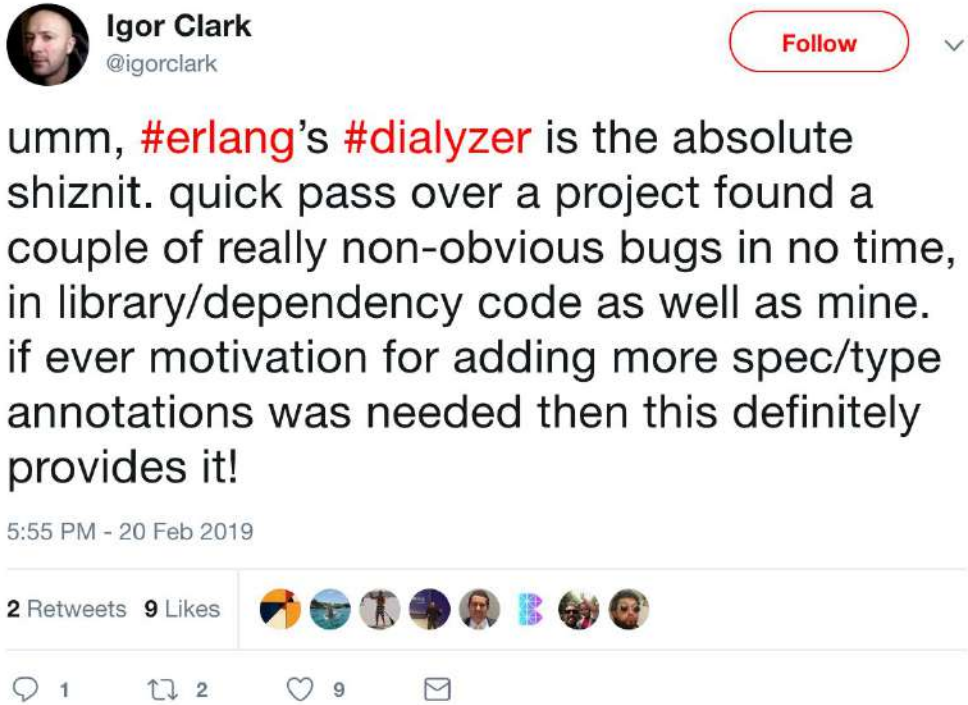


Testing & Verification Tools

Program analyses tools	Sound	Complete
Under-approximate	Report only real bugs (may miss some)	Find all bugs (may report false positives)
Over-approximate	Produce only real safety proofs (may reject safe programs)	Prove safety for all safe programs (may accept unsafe programs)

https://twitter.com/johannes_kinder/status/1105138480303218688

The bright side of life

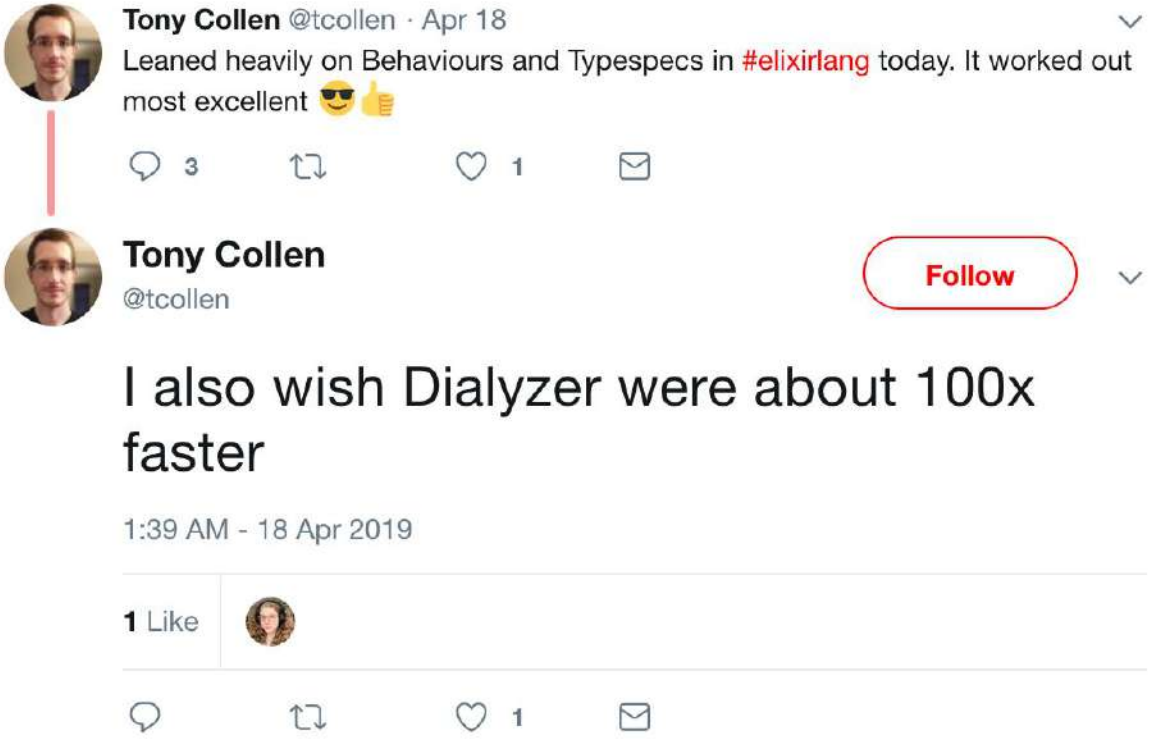


3.

EVEN MORE THOUGHTS?

Speed?

Speed?



dialyzer --statistics -n --apps <all of OTP>

```
compile   (+0.07s):  18.57s (1410 modules)
clean     (+0.01s):   0.79s
remote    (+0.14s):   9.43s
order     (+0.57s):   4.95s
typesig   (+0.00s):  59.18s (  88558 SCCs)
order     (+0.00s):   4.02s
refine    (+0.00s):  30.97s (1410 modules)
order     (+0.00s):   5.77s
typesig   (+0.00s):  61.03s (  70444 SCCs)
order     (+0.00s):   2.31s
refine    (+0.00s):  30.42s (1340 modules)
order     (+0.00s):   0.16s
typesig   (+0.00s):   8.87s (   3107 SCCs)
order     (+0.00s):   0.06s
refine    (+0.00s):   8.13s ( 247 modules)
order     (+0.00s):   0.00s
typesig   (+0.00s):   1.24s (     11 SCCs)
order     (+0.00s):   0.00s
refine    (+0.00s):   0.40s (   2 modules)
warning   (+1.58s):  34.35s (1410 modules)
(+ 0.45s)
```

done in 4m43.61s

On my 2014 Quad-core Macbook Pro

dialyzer +S 1:1 --statistics -n --apps <all of OTP>

```
compile   (+0.09s):  40.06s (1410 modules)
clean     (+0.01s):   0.81s
remote    (+0.13s):   9.65s
order     (+0.60s):   5.00s
typesig   (+0.00s): 129.66s (  88558 SCCs)
order     (+0.00s):   4.05s
refine    (+0.00s):  65.26s (1410 modules)
order     (+0.00s):   5.86s
typesig   (+0.00s): 137.26s (  70444 SCCs)
order     (+0.00s):   2.38s
refine    (+0.00s):  67.58s (1340 modules)
order     (+0.00s):   0.17s
typesig   (+0.00s):  18.38s (   3107 SCCs)
order     (+0.00s):   0.06s
refine    (+0.00s):  17.36s ( 247 modules)
order     (+0.00s):   0.00s
typesig   (+0.00s):   1.24s (     11 SCCs)
order     (+0.00s):   0.00s
refine    (+0.00s):   0.36s (   2 modules)
warning   (+1.42s):  77.77s (1410 modules)
(+ 0.50s)
```

done in 9m46.04s

If restricted to one core

4.

Conclusions

Why & how I use Dialyzer?

- To check my types & specs
- To simplify my code
- As the first job on CI test runs
- Properly focused (only my code, not dependencies)
    - On sane dependencies, possibly patched
- With a cached PLT
- With HiPE enabled

More reads

- Decoding Dialyzer

  http://devonestes.herokuapp.com/decoding-dialyzer
- Spot The Discrepancies with Dialyzer for Erlang

  http://tech.adroll.com/blog/dev/2019/02/19/erlang-dialyzer.html
- Chemanalysis: Dialyzing Elixir

  https://codesync.global/media/chemanalysis-dialyzing-elixir-sean-cribbs

THANK YOU!
YOUR THOUGHTS?

Stavros Aronis
Developer & Trainer @ Erlang Solutions

@Vahnatai
stavros.aronis@erlang-solutions.com

#CodeBEAMSTO

Erlang SOLUTIONS