Validation using Erlang's type system

with Sheriff

Loïc Hoguin

Dev:Extend

Good Erlang code

```
-type year() :: 1900..2011.
-type age() :: 0..111.

-spec main() -> no_return().
main() ->
    Year = 1984,
    Age = calculate_age(Year),
    io:format("~b years old~n", [Age]),
    main().

-spec calculate_age(year()) -> age().
calculate_age(Year) ->
    2011 - Year.
```

Dialyzer is awesome, isn’t it?

- Dialyzer statically checks function specifications
- All type errors will be reported by Dialyzer
- Okay, code isn’t that good, it’s just a dumb example

Good Erlang code receiving external data

```
-type year() :: 1900..2011.
-type age() :: 0..111.

-spec main() -> no_return().
main() ->
    receive {year_of_birth, Year} ->
        Age = calculate_age(Year),
        io:format("~b years old~n", [Age])
    end,
    main().

-spec calculate_age(year()) -> age().
calculate_age(Year) ->
    2011 - Year.
```

We lost Dialyzer

- Dialyzer doesn’t help here
- `Year` could be any integer in this code!
  - Even year 3001
  - Or it could be a binary, a list, a tuple, a pid…
- This value wouldn’t match the type we defined

What is external data?

- Anything that doesn’t come directly from your process
- This includes:
  - Process messages
  - Shared resources
  - Sockets, files
  - Return values from NIFs, ports, linked-in drivers
- Real-world applications are all about external data!

Why check external data? Just let it crash!

- WHAT?!
- Are you sure it’ll crash?
- Maybe it’s going to crash an unrelated process
  - Like a central gen_server of your application
- Maybe it’s going to be stored in a database or a file
- Maybe it’s going to be sent directly to connected clients

Really?

- A person registering on your website today can’t be born in 1492!
- Think about it, are you really crashing on this kind of data?
- Also think about XSS, SQL injection, and friends

Always validate external data

- Either print a nice error message to the user
  - HTML forms, for example
- Or crash as soon as possible
  - Don’t crash anywhere! Crash on the system boundaries
  - Don’t let bad data crash your core processes
  - Don’t let external attacks or user error bring down your app

Data validation without Sheriff

```
-type year() :: 1900..2011.

-spec is_valid_year(year()) -> boolean().
is_valid_year(Y)
        when is_integer(Y), Y >= 1900, Y =< 2011 ->
    true;
is_valid_year(_Y) ->
    false.
```

All this has happened before…

- I feel like I’m repeating myself there

And all this will happen again

- I did write the same constraint twice
- Dialyzer already checks it for most of the program
- Why not use the `year()` type directly?

I can’t type

- But I can’t use Erlang’s types from runtime code!

Who do you call when you need help?

I am the law

- Sheriff is a runtime type checker
- It uses Erlang’s type system for validation
- You don’t need to duplicate constraints to validate data anymore!
- So just be lazy and validate all external data with a single LoC

Data validation with Sheriff

```
true = sheriff:check(Y, year()).
```

Tuple validation with Sheriff

```
-type subject() :: 'I' | you | he.
-type verb() :: like | ignore | hate.
-type object() :: me | you | him.

-type grammar() :: {subject(), verb(), object()}.

%% ...

sheriff:check({you, like, me}, grammar()). %% true
sheriff:check({'I', love, you}, grammar()). %% false
```

Recursive type validation with Sheriff

```
-type rtype() :: {leaf | rtype(), leaf | rtype()}.

%% ...

sheriff:check({leaf, {leaf, leaf}}, rtype()). %% true
sheriff:check({{flower, flower}, leaf}, rtype()). %% false
sheriff:check(<<"flower">>, rtype()). %% false
```

Parameterized type validation with Sheriff

```
-type a() :: 0..65535.
-type b(T) :: undefined | T.
-type c() :: b(a()).

%% ...

sheriff:check(undefined, c()). %% true
sheriff:check(42, c()). %% true
sheriff:check(-1, c()). %% false
sheriff:check(1234567890, c()). %% false
sheriff:check(defined, c()). %% false
```

Record validation with Sheriff

```
-record(packet, {
 id :: 1 | 2 | 3,
 num = 0 :: non_neg_integer(),
 data = <<>> :: binary()
}).
-type packet() :: #packet{}.

%% ...

sheriff:check(#packet{id=1, data= <<0:32>>}, packet()). %% true
sheriff:check(#packet{id=undefined}, packet()). %% true
sheriff:check({packet, 2, 1, <<>>}, packet()). %% true
sheriff:check(#packet{id=0, num=7}, packet()). %% false
sheriff:check(#http_req{}, packet()). %% false
```

Digging in

- Sheriff is a parse_transform
- It first generates validation functions for all the types you defined
- It then replaces the `sheriff:check/2` calls with the proper validation calls
- It’s fast and is only a compilation option away

Don’t fall into lava

- There are limitations
- Exported types can only work on modules that were compiled using the sheriff parse_transform
  - Excluding basic types like `integer()` of course
- It’s only as good as Erlang’s type system
  - It can’t check element order in lists
  - It can’t check the content of binaries, only size
  - This can probably be fixed later
- Dialyzer will print out some warnings if analyzing from source

Sheriff got deputies

- Sheriff, today, is only a proof of concept
- Code was written by two *Dev:Extend* interns
  - William Dang
  - Hamza Mahmood

- They only had one month to learn Erlang and do the project
- So there’s probably many bugs!

To infinity, and beyond!

- We’ll cleanup the codebase
- We’ll add PropEr tests
- We’ll add a few missing features
- First release is planned for December 2011

Wanted

- You can help!
- Source code is already available on https://github.com/extend/sheriff
- Try it out
- Suggest improvements
- File bug reports
  - Wait for the code cleanup though

Suit up!

- We’ll ping Kostis Sagonas and the red ties at some point
- They have experience, they can probably help

Fin

- Any questions?