A PropEr Creation

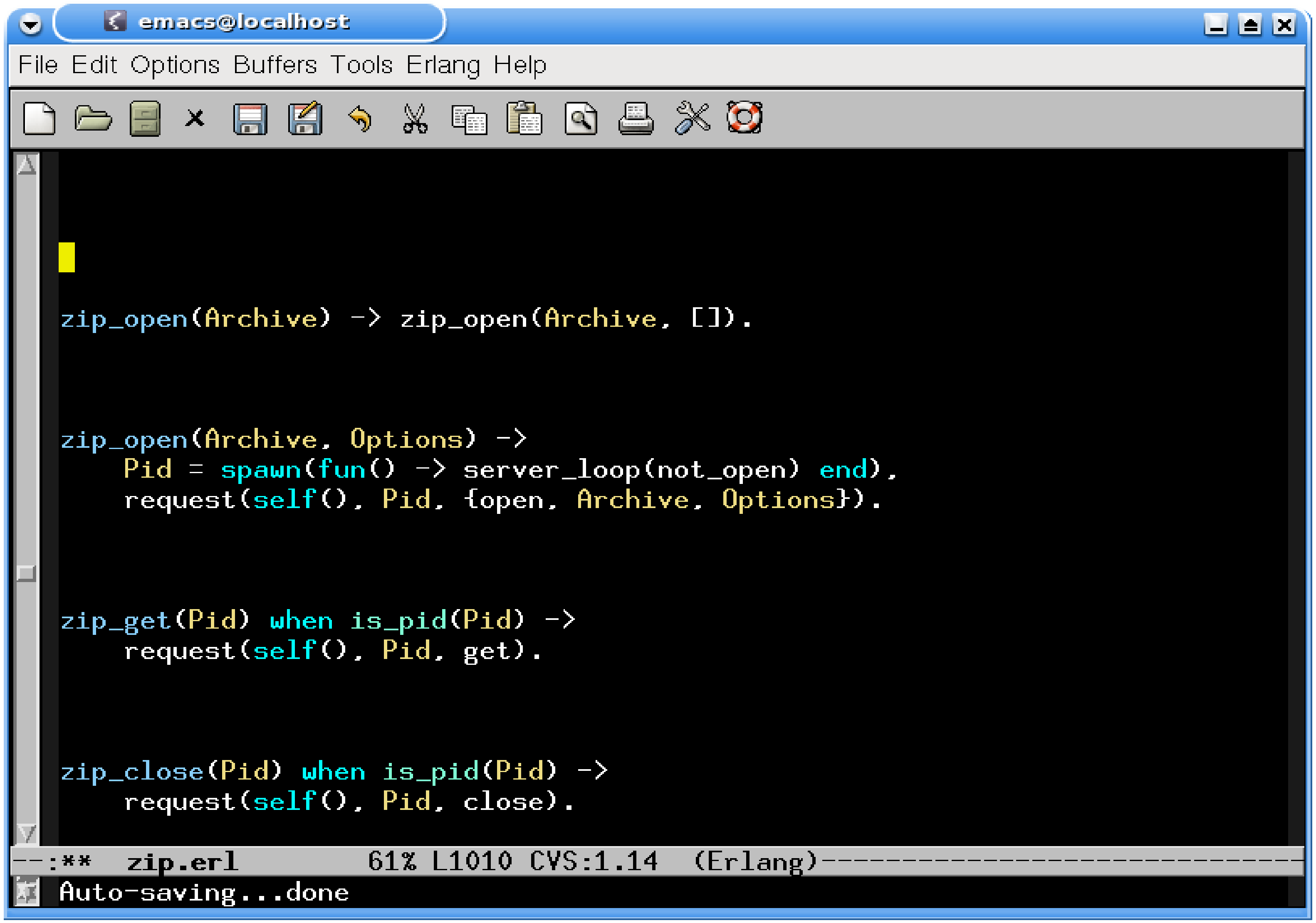
Kostis Sagonas

Overview

- Introductory rambling part:
  - How PropEr was created
  - Myths vs. reality
- A taste of PropEr technology
- Some PropEr advice
- Current uses and exciting projects using PropEr
- PropEr thoughts on open- vs. closed-source software development in Erlang
- PropEr thoughts on licenses

How PropEr was created

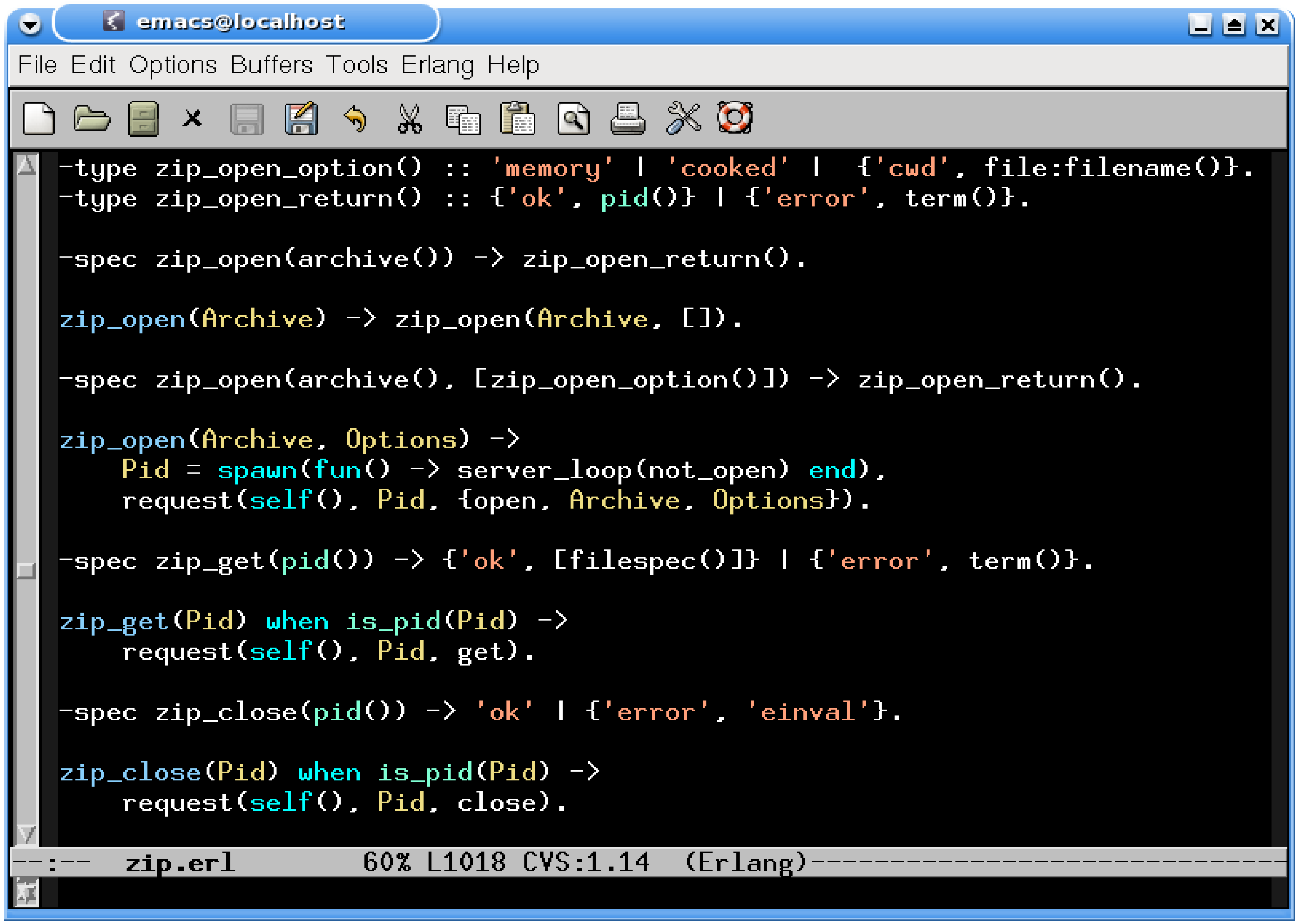
- A Canada trip: Montreal, September 2000



```
zip_open(Archive) -> zip_open(Archive, []).


zip_open(Archive, Options) ->
    Pid = spawn(fun() -> server_loop(not_open) end),
    request(self(), Pid, {open, Archive, Options}).


zip_get(Pid) when is_pid(Pid) ->
    request(self(), Pid, get).


zip_close(Pid) when is_pid(Pid) ->
    request(self(), Pid, close).
--:**  zip.erl        61% L1010 CVS:1.14  (Erlang)------------------------------
Auto-saving...done
```

How PropEr was created

- Another Canada trip: Victoria, September 2008



```
-type zip_open_option() :: 'memory' | 'cooked' |  {'cwd', file:filename()}.
-type zip_open_return() :: {'ok', pid()} | {'error', term()}.

-spec zip_open(archive()) -> zip_open_return().

zip_open(Archive) -> zip_open(Archive, []).

-spec zip_open(archive(), [zip_open_option()]) -> zip_open_return().

zip_open(Archive, Options) ->
    Pid = spawn(fun() -> server_loop(not_open) end),
    request(self(), Pid, {open, Archive, Options}).

-spec zip_get(pid()) -> {'ok', [filespec()]} | {'error', term()}.

zip_get(Pid) when is_pid(Pid) ->
    request(self(), Pid, get).

-spec zip_close(pid()) -> 'ok' | {'error', 'einval'}.

zip_close(Pid) when is_pid(Pid) ->
    request(self(), Pid, close).
--:--  zip.erl        60% L1018 CVS:1.14  (Erlang)------------------------------
```

How PropEr was created

From John Hughes (20/10/2008):

… By the way, we'd be happy to supply an academic licence for QuickCheck for your student in Greece, on the same terms as to our ProTest partners. The advantage over doing something yourself based on … <IDEA> ... would be that you would get our compositional approach to shrinking built-in, which makes it very easy to define, use, and combine shrinking strategies. In fact, it would be very interesting to see if he can come up with some smart ways of deriving generators automatically, or more easily. This is one of the most important areas for improvement in this kind of testing.

How PropEr was created

My reply:

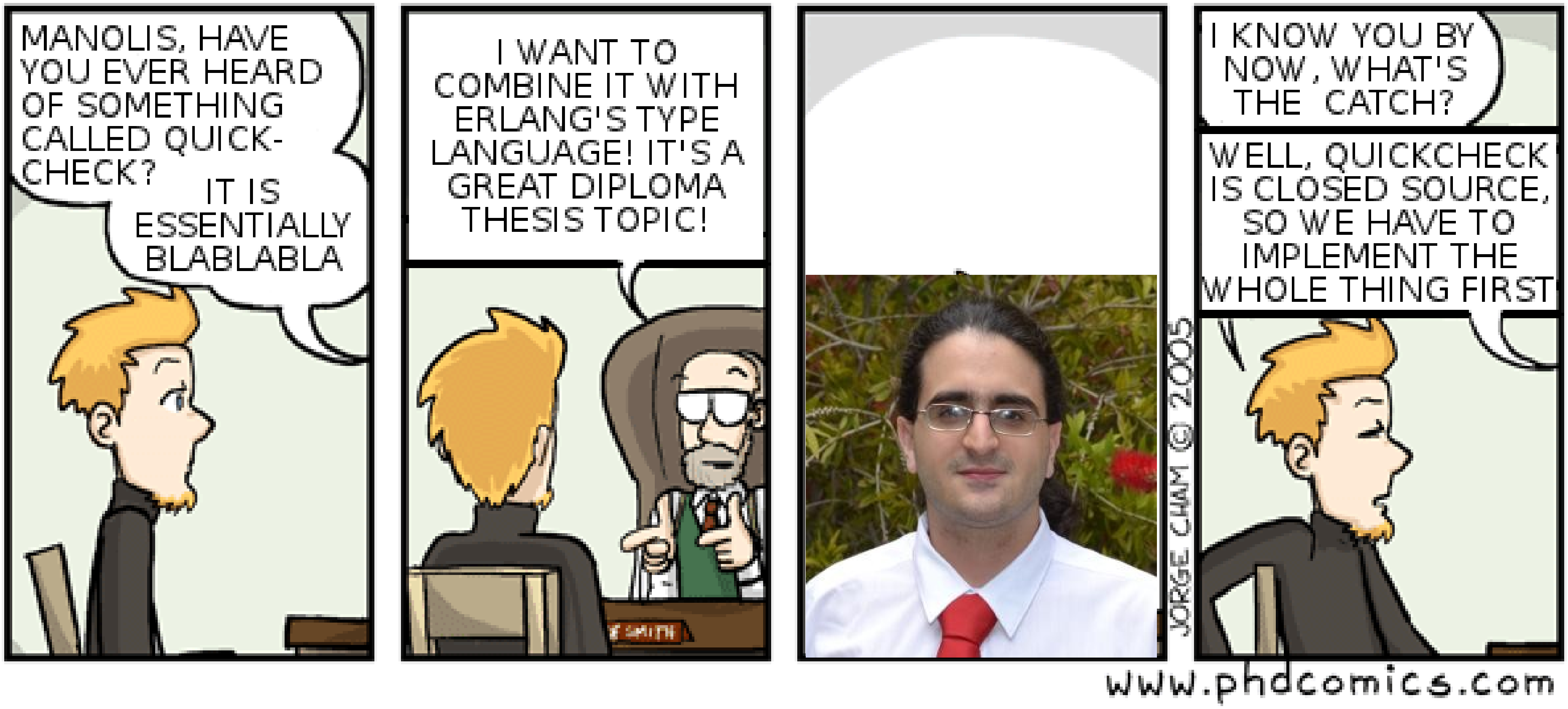
Indeed it is. I'll think about it ... that particular student is not committed yet. One thing is for certain though: I have no grants here, so even one Euro for a licence of any sort is one Euro too much :-(

From John Hughes (21/10/2008):

Well, I suppose an alternative is that the student deliver something of value to us... like a report or whatever, so that we could "sponsor" the project with a licence. It does cost us a little bit to provide one, but if there's something useful that comes out of the project, then we could take that cost. I guess we'd want the right to make use of any results royalty-free in return (although presumably you'd be planning to publish any interesting results anyway, so the question is really pretty moot).

How PropEr was created

- A trip at the Erlang Factory, London, June 2009
  - QuickCheck tutorial
- Back in Athens, October/November 2009


MANOLIS, HAVE YOU EVER HEARD OF SOMETHING CALLED QUICK-CHECK?
IT IS ESSENTIALLY BLABLABLA
I WANT TO COMBINE IT WITH ERLANG'S TYPE LANGUAGE! IT'S A GREAT DIPLOMA THESIS TOPIC!
I KNOW YOU BY NOW, WHAT'S THE CATCH?
WELL, QUICKCHECK IS CLOSED SOURCE, SO WE HAVE TO IMPLEMENT THE WHOLE THING FIRST
JORGE CHAM © 2005
www.phdcomics.com

How PropEr was created

- Athens, November 2009 – March 2010
  - Complete support for properties and generators

    `?FORALL/3, ?LET, ?IMPLIES, ?SUCHTHAT/3,`
    `?SHRINK/2, ?LAZY/1, ?SIZED/2, …`
    `aggregate/2, choose2, oneof/1, …`
- Athens, March 2010 – May 2010
  - Complete intregration with types and specs
- Stockholm, April 2010
  - I hear about Trifork's Triq implementation
- London, June 2010
  - QuviQ announces EQC Mini

How PropEr was created

- Athens, September/October 2010
  - Plans to extend PropEr with component for testing stateful systems
- Athens, November 2010 – March 2011
  - Implementation of ‘statem’
- San Francisco, March 2011
  - Informal announcement of PropEr
- Athens, April/May 2011
  - Implementation of ‘fsm’
- London, June 2011
  - A PropEr announcement of the tool

Myth vs. reality

“We reversed-engineered QuviQ QuickCheck by access to its code or from its bytecode”

- None of us ever used EQC or owned a copy
- EQC Mini was released after we already had more (all?) of its functionality in PropEr
- None of the PropEr developers ever used EQC Mini since then (not even to check compatibility with PropEr)
- **A clean-room implementation from scratch**
- **“Inspiration” from open-source projects (github)**
- **Help from some EQC users**

Myth vs. reality

"... it is quite easy to be in a position where you are funded by external bodies, in the case of PropEr being funded by a large university and also EU grants, and then afford to give the software away for free..."

- We received absolutely no funding for working on PropEr
- There is actually NO MONEY at all in Greece, let alone research grants from the University or government !
- The National Technical University of Athens has NOT been part of the ProTest EU project

Myth vs. reality

“... I also think it makes it a tad rich to release it under something as restrictive as the GPL given that it's the profits of the commercial companies, by virtue of paying taxes, that have in effect funded the software...”

We received absolutely no funding for working on PropEr

Whether EU funded reseach projects should give back their research outcome back to society for free under unrestricted licenses is a matter of debate...

PropEr

A Property-based Testing Tool for Erlang

- Available open source under GPL v3

  (more on that later...)

- Has support for
  - Writing properties and test case generators
  - Concurrent/parallel “statem” and “fsm” testing
- Full integration with the language of types and function specifications
  - Generators often come for free!

Testing simple properties (1)

```erlang
-module(simple_props).

%% Properties are automatically exported.
-include_lib("proper/include/proper.hrl").

%% Functions that start with prop_ are considered properties
prop_t2b_b2t() ->
  ?FORALL(T, term(), T =:= binary_to_term(term_to_binary(T))).
```

```
1> c(simple_props).
{ok,simple_props}
2> proper:quickcheck(simple_props:prop_t2b_b2t()).
...................................................
.................................................
OK: Passed 100 test(s)
true
```

Testing simple properties (2)

```
%% Testing the base64 module:
%%   encode should be symmetric to decode:

prop_enc_dec() ->
  ?FORALL(Msg, union([binary(), list(range(1,255))]),
      begin
        EncDecMsg = base64:decode(base64:encode(Msg)),
        case is_binary(Msg) of
          true  -> EncDecMsg =:= Msg;
          false -> EncDecMsg =:= list_to_binary(Msg)
        end
      end).
```

PropEr integration with simple types

```
%% Using a user-defined simple type as a generator
-type bl() :: binary() | [1..255].

prop_enc_dec() ->
  ?FORALL(Msg, bl(),
      begin
        EncDecMsg = base64:decode(base64:encode(Msg)),
        case is_binary(Msg) of
          true  -> EncDecMsg =:= Msg;
          false -> EncDecMsg =:= list_to_binary(Msg)
        end
      end).
```

PropEr shrinking

```erlang
%% A lists delete implementation
-spec delete(T, list(T)) -> list(T).
delete(X, L) ->
  delete(X, L, []).

delete(_, [], Acc) ->
  lists:reverse(Acc);
delete(X, [X|Rest], Acc) ->
  lists:reverse(Acc) ++ Rest;
delete(X, [Y|Rest], Acc) ->
  delete(X, Rest, [Y|Acc]).
```

```erlang
prop_delete() ->
  ?FORALL({X,L}, {integer(),list(integer())},
          not lists:member(X, delete(X, L))).
```

PropEr shrinking

```
41> c(simple_props).
{ok,simple_props}
42> proper:quickcheck(simple_props:prop_delete()).
.........................................!
Failed: After 42 test(s).
{12,[-36,-1,-2,7,19,-14,40,-6,-8,42,-8,12,12,-17,3]}

Shrinking ...(3 time(s))
{12,[12,12]}
false
```

PropEr integration with types

```
-type tree(T) :: 'leaf' | {'node',T,tree(T),tree(T)}.
```

```
%% A tree delete implementation
-spec delete(T, tree(T)) -> tree(T).
delete(X, leaf) ->
  leaf;
delete(X, {node,X,L,R}) ->
  join(L, R);
delete(X, {node,Y,L,R}) ->
  {node,Y,delete(X,L),delete(X,R)}.
```

```
join(leaf, T) -> T;
join({node,X,L,R}, T) ->
  {node,X,join(L,R),T}.
```

```
prop_delete() ->
  ?FORALL({X,L}, {integer(),tree(integer())},
          not lists:member(X, delete(X, L))).
```

What one would have to write in EQC

```
tree(G) ->
  ?SIZED(S, tree(S, G)).

tree(0, _) ->
  leaf;
tree(S, G) ->
  frequency([
    {1, tree(0, G)},
    {9, ?LAZY(
            ?LETSHRINK(
              [L, R],
              [tree(S div 2, G), tree(S div 2, G)],
              {node, G, L, R}
          ))}
  ]).
```

What one has to write in PropEr

This slide intentionally left blank

Integration with recursive types

```
41> c(mytrees).
{ok,mytrees}
42> proper:quickcheck(mytrees:prop_delete()).
.......................!
Failed: After 24 test(s).
{6,{node,19,{node,-19,leaf,leaf},
            {node,6,leaf,{node,6,leaf,leaf}}}}

Shrinking .(1 time(s))
{6,{node,6,{node,6,leaf,leaf}}}
false
```

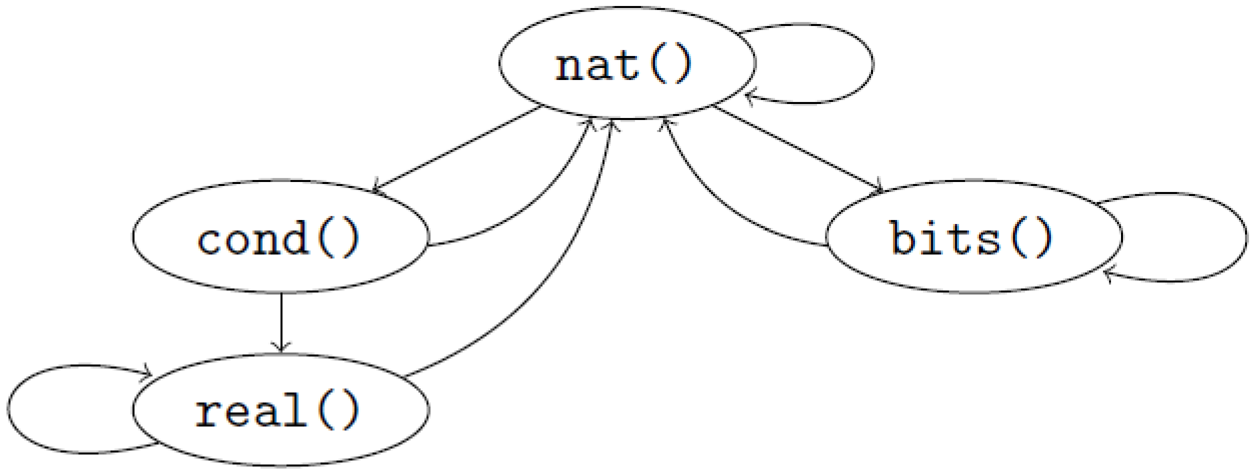
Generators from recursive types

Takes place, roughly, in the following steps

- Detect recursion
- Inline (non-recursive) type definitions
- Normalize by pushing unions to the top level
- Find base cases
- Prepare the recursive calls
- Determine shrinking behavior
- Compose a generator

Example: detecting recursion

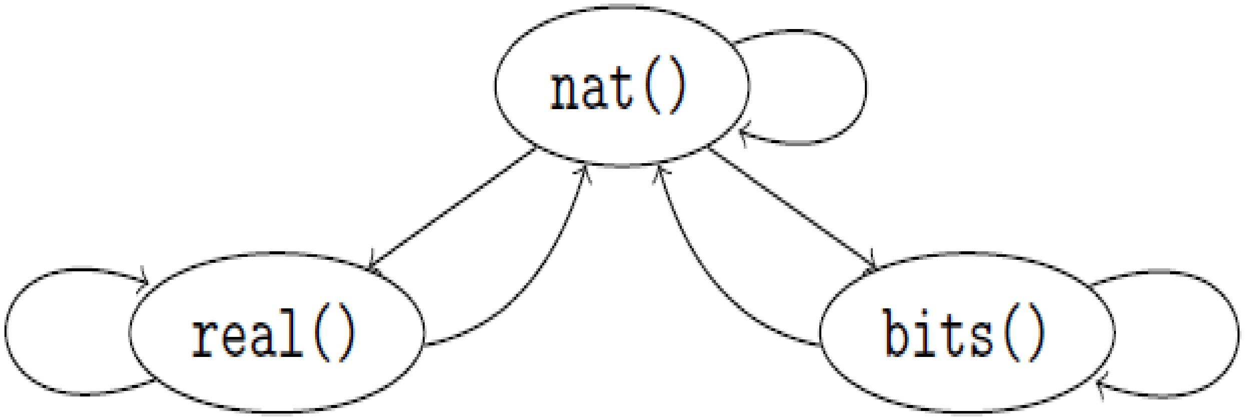
```
-type nat()  :: non_neg_integer()
              | {'+', nat(), nat()}
              | {'if', cond(), nat(), nat()}
              | {'from_bits', bits()}.
-type cond() :: {'=', nat(), nat()}
              | {'=', real(), real()}.
-type real() :: {'from_nat', nat()}
              | {'+', real(), real()}.
-type bits() :: {'from_nat', nat()}
              | {'concat', [bits() | nat()]}.
```



$$\begin{tikzcd}
 & \text{nat()} \arrow[loop right] \arrow[dl] \arrow[dr] & \\
\text{cond()} \arrow[ur, bend right] \arrow[d] & & \text{bits()} \arrow[ul, bend left] \arrow[loop right] \\
\text{real()} \arrow[loop left] \arrow[uur, bend right] & &
\end{tikzcd}$$

Example: after inlining

```
 nat() :: non_neg_integer()
        | {'+', nat(), nat()}
        | {'if',
           {'=', nat(), nat()} | {'=', real(), real()},
           nat(), nat()}
        | {'from_bits', bits()}.
real() :: {'from_nat', nat()} | {'+', real(), real()}.
bits() :: {'from_nat', nat()} | {'concat', [nat() | bits()]}.
```



Example: after normalization

nat() :: non_neg_integer()$_2$
  | {'+', nat(), nat()}$_2$
  | {'if', {'=',nat(),nat()}, nat(), nat()}$_1$
  | {'if', {'=',real(),real()}, nat(), nat()}$_1$
  | {'from_bits', bits()}$_2$.
real() :: {'from_nat', nat()}$_1$ | {'+', real(), real()}$_1$.
bits() :: {'from_nat', nat()}$_1$ | {'concat', [bits_helper()]}$_1$.
bits_helper() :: nat()$_1$ | bits()$_1$.

$$\begin{tikzcd}
& \text{nat()} \arrow[loop right] \arrow[dl] \arrow[dr] & \\
\text{real()} \arrow[loop left] \arrow[ur, bend right] & & \text{bits()} \arrow[ul, bend left] \arrow[d] \\
& & \text{bits\_helper()} \arrow[uul, bend left] \arrow[u, bend right]
\end{tikzcd}$$

Example: the generated generator

```
nat() ->
    ?SIZED(Size, nat(Size)).

nat(0) ->
    non_neg_integer();
nat(S) ->
    weighted_union([
        {2, ?LAZY(nat(0))},
        {2, ?LAZY(non_neg_integer())},
        {2, ?LAZY(?LETSHRINK([X,Y], vector(2,nat(S div 2)),
                {’+’, X, Y}))},
        {1, ?LAZY(?LETSHRINK([X,Y,Z,W], vector(4,nat(S div 4)),
                {’if’, {’=’, X, Y}, Z, W}))},
        {1, ?LAZY(?LETSHRINK([X,Y], vector(2,nat(S div 4)),
                {’if’, {’=’, real(S div 4),real(S div 4)},
                       X, Y}))},
        {2, ?LAZY({’from_bits’,from_bits(S)})}]).

real() ->
    ?SIZED(Size, real(Size)).

real(0) ->
    {’from_nat’,nat(0)};
real(S) ->
    weighted_union([
        {2, ?LAZY(real(0))},
        {3, ?LAZY({’from_nat’,nat(S-1)})},
        {3, ?LAZY(?LETSHRINK([X,Y], vector(2,real(S div 2)),
                {’+’, X, Y}))}]).
```

```
bits() ->
    ?SIZED(Size, bits(Size)).

bits(0) ->
    {’concat’,[]};
bits(S) ->
    weighted_union([
        {2, ?LAZY(bits(0))},
        {3, ?LAZY({’from_nat’,nat(S-1)})},
        {3, ?LAZY({’concat’,resize(S,list(bits_helper

bits_helper() ->
    ?SIZED(Size, nat(Size)).

bits_helper(0) ->
    union([nat(0), bits(0)]);
bits_helper(S) ->
    weighted_union([{2, ?LAZY(bits_helper(0))},
                    {3, ?LAZY(nat(S-1))},
                    {3, ?LAZY(bits(S-1))}]).
```

PropEr integration with remote types

- We want to test that **`array:new/0`** can handle any combination of options
- Why write a custom generator (which may rot)?
- We can use the remote type as a generator!

```
-type array_opt() :: 'fixed' | non_neg_integer()
                   | {'default', term()}
                   | {'fixed', boolean()}
                   | {'size', non_neg_integer()}.
-type array_opts() :: array_opt() | [array_opt()].
```

```
-module(types).
-include_lib("proper/include/proper.hrl").

prop_new_array_opts() ->
  ?FORALL(Opts, array:array_opts(),
          array:is_array(array:new(Opts))).
```

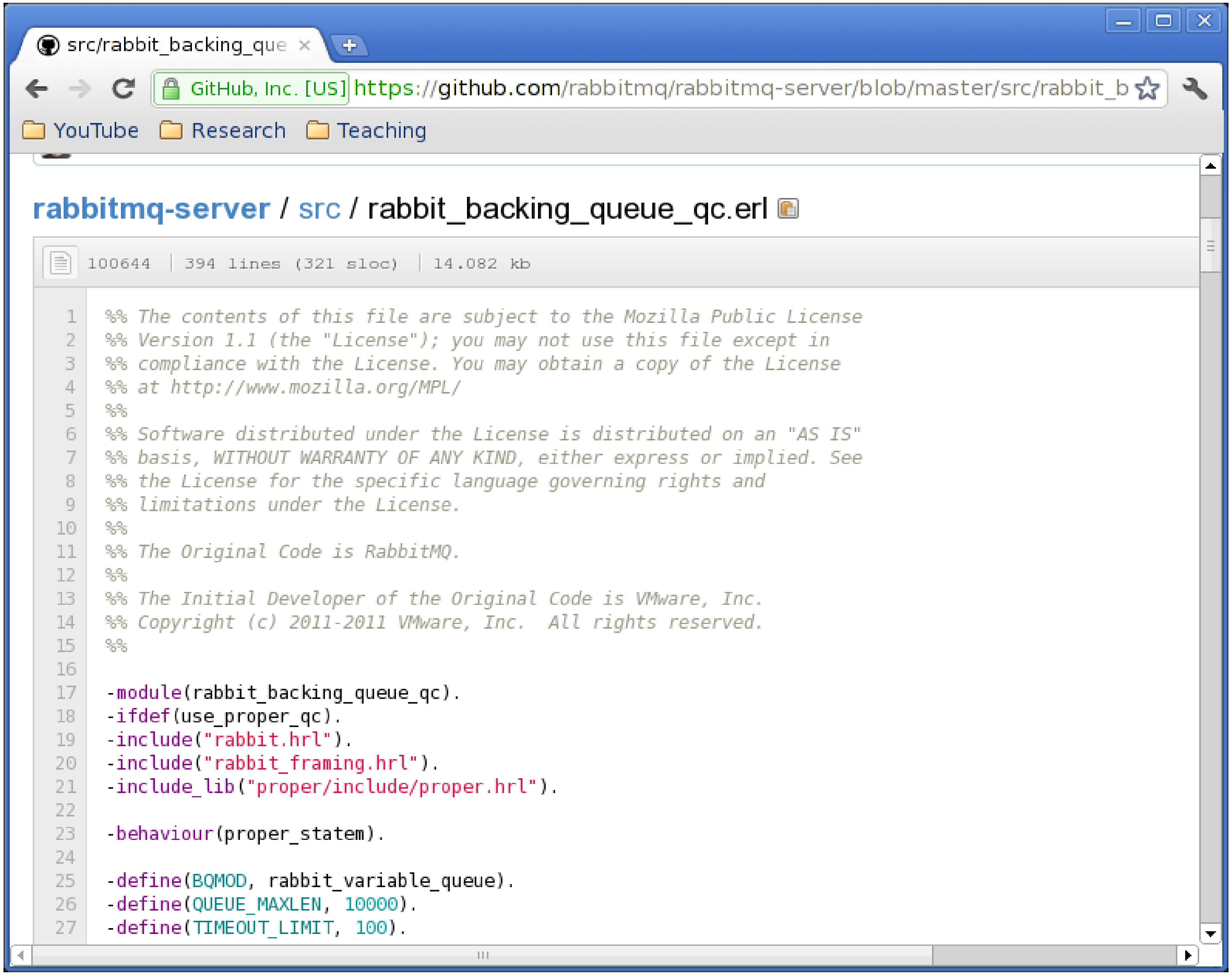
PropEr testing of specs

```erlang
-module(myspecs).

-export([divide/2, filter/2, max/1]).

-spec divide(integer(), integer()) -> integer().
divide(A, B) ->
  A div B.

-spec filter(fun((T) -> term()), [T]) -> [T].
filter(Fun, List) ->
  lists:filter(Fun, List).

-spec max([T]) -> T.
max(List) ->
  lists:max(List).
```

PropEr testing of specs

```
1> c(myspecs).
{ok,myspecs}
2> proper:check_spec({myspecs,divide,2}).
!
Failed: After 1 test(s).
An exception was raised: error:badarith.
Stacktrace: [{myspecs,divide,2}].
[0,0]

Shrinking (0 time(s))
[0,0]
false
      .... AFTER FIXING THE PROBLEMS ....
42> proper:check_specs(myspecs).
```

PropEr uses



src/rabbit_backing_que

GitHub, Inc. [US] https://github.com/rabbitmq/rabbitmq-server/blob/master/src/rabbit_b

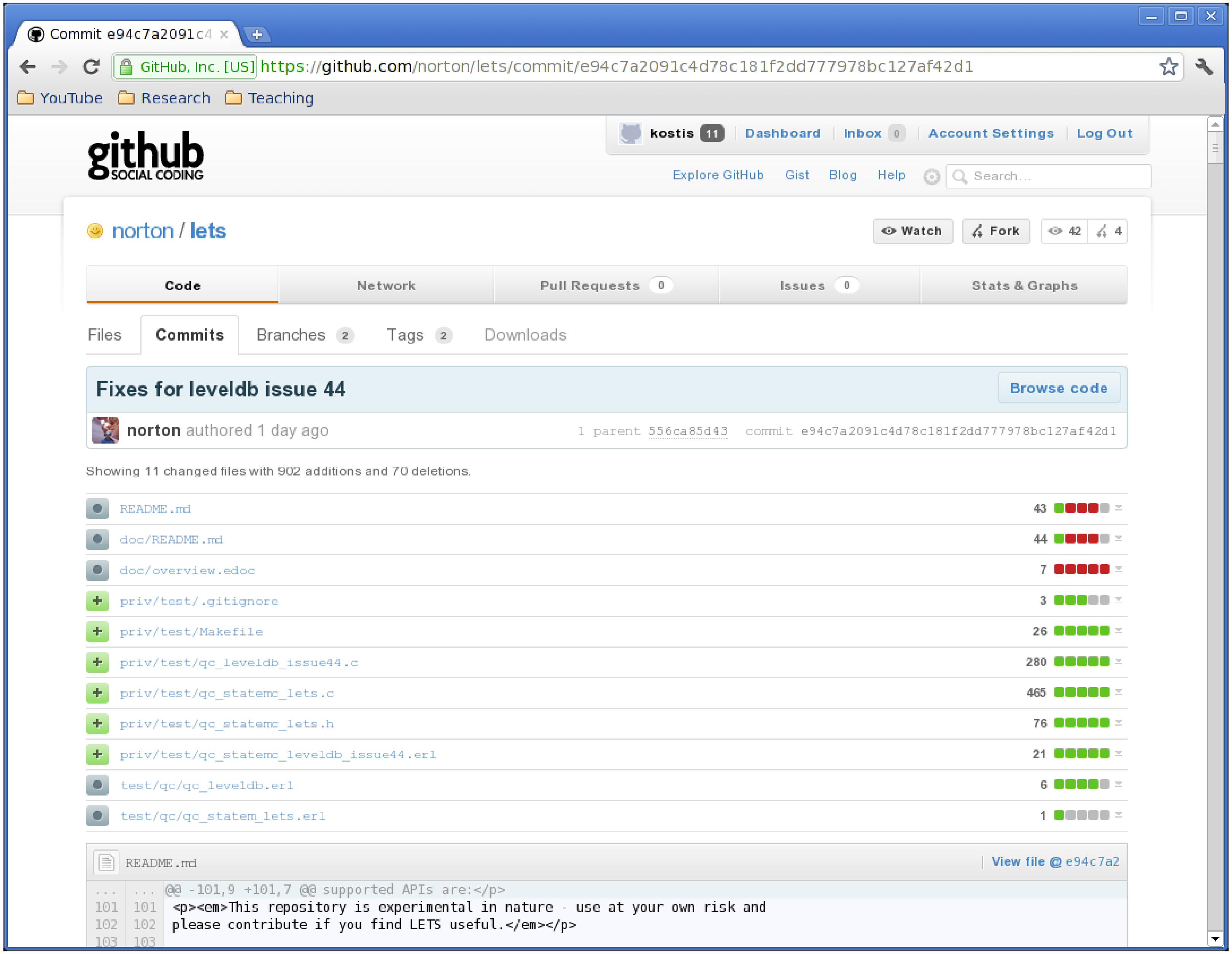
YouTube Research Teaching

rabbitmq-server / src / rabbit_backing_queue_qc.erl

100644 | 394 lines (321 sloc) | 14.082 kb

```erlang
%% The contents of this file are subject to the Mozilla Public License
%% Version 1.1 (the "License"); you may not use this file except in
%% compliance with the License. You may obtain a copy of the License
%% at http://www.mozilla.org/MPL/
%%
%% Software distributed under the License is distributed on an "AS IS"
%% basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See
%% the License for the specific language governing rights and
%% limitations under the License.
%%
%% The Original Code is RabbitMQ.
%%
%% The Initial Developer of the Original Code is VMware, Inc.
%% Copyright (c) 2011-2011 VMware, Inc.  All rights reserved.
%%

-module(rabbit_backing_queue_qc).
-ifdef(use_proper_qc).
-include("rabbit.hrl").
-include("rabbit_framing.hrl").
-include_lib("proper/include/proper.hrl").

-behaviour(proper_statem).

-define(BQMOD, rabbit_variable_queue).
-define(QUEUE_MAXLEN, 10000).
-define(TIMEOUT_LIMIT, 100).
```

PropEr uses



Fixes for leveldb issue 44

norton authored 1 day ago 1 parent 556ca85d43 commit e94c7a2091c4d78c181f2dd777978bc127af42d1

Showing 11 changed files with 902 additions and 70 deletions.

●	README.md	43
●	doc/README.md	44
●	doc/overview.edoc	7
+	priv/test/.gitignore	3
+	priv/test/Makefile	26
+	priv/test/qc_leveldb_issue44.c	280
+	priv/test/qc_statemc_lets.c	465
+	priv/test/qc_statemc_lets.h	76
+	priv/test/qc_statemc_leveldb_issue44.erl	21
●	test/qc/qc_leveldb.erl	6
●	test/qc/qc_statem_lets.erl	1

README.md View file @ e94c7a2

```
@@ -101,9 +101,7 @@ supported APIs are:</p>
 <p><em>This repository is experimental in nature - use at your own risk and
 please contribute if you find LETS useful.</em></p>
```

Kostis Sagonas A PropEr Creation

Some observations from PropEr uses

- Erlang's type language is often less expressive than desired for property-based testing
  - e.g. not possible to specify that binaries should contain valid UTF8 characters
- Function specs cannot express argument dependencies
  - e.g. dependencies between args of `lists:nth/2`
- Users often under-specify function domains
- Function signatures can often be used as simple specifications of functions

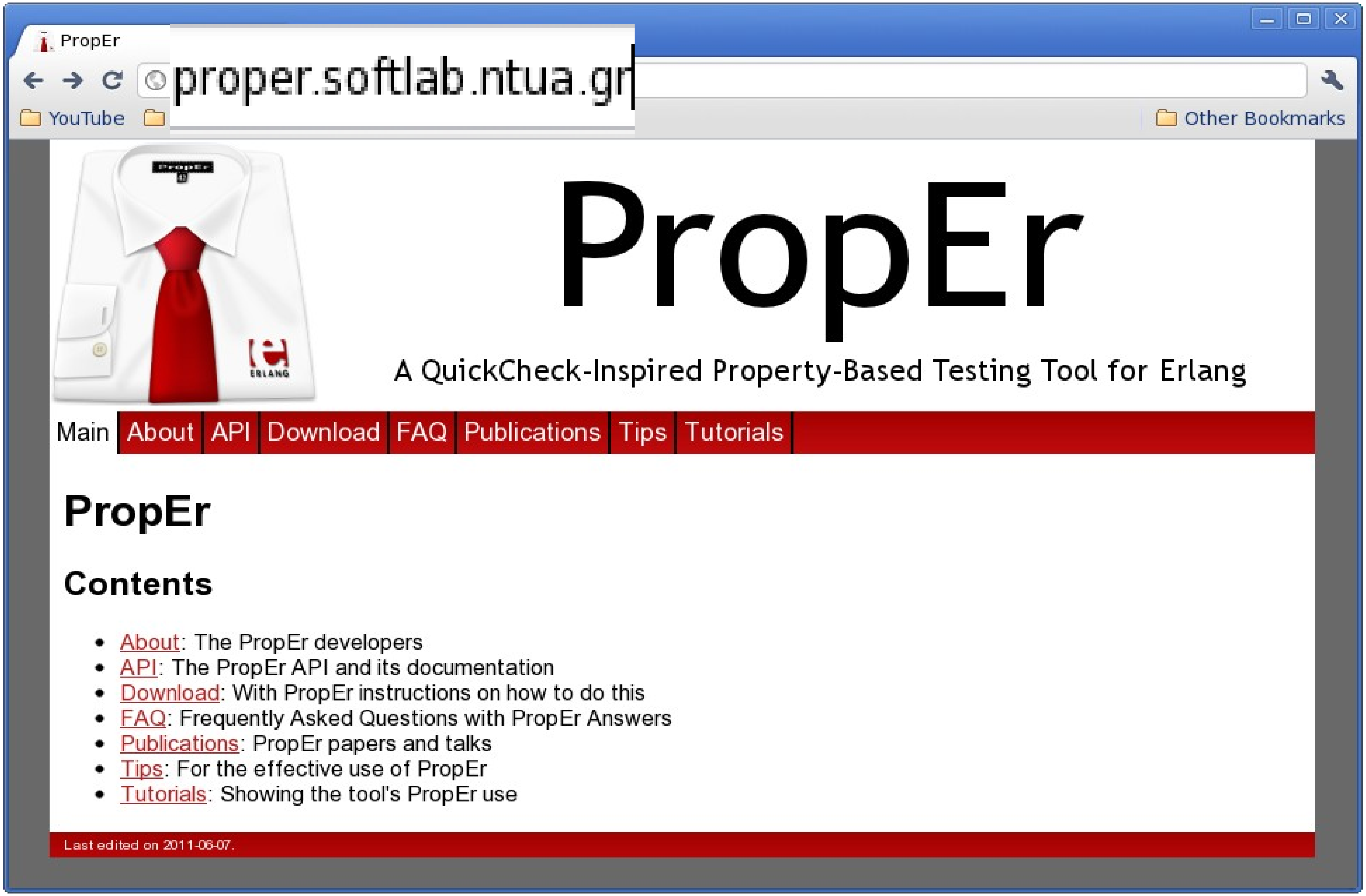
Lessons learned

- Unit testing and property-based testing require different mindsets
  - Difficult to come up with "interesting" properties
  - Tricky to express them
    - often one debugs the property rather than the code
- Writing generators for recursive types is tricky and requires significant time and effort
  - PropEr significantly eases this task

Some PropEr advice

- Start with testing the functional core
- Break the testing into smaller, simpler to express (partial) correctness properties
- Write properties for readability
- For generators of recursive datatypes
  - Just write the data type and rely on PropEr
  - Put a global size bound if the above is not enough
  - Only if the steps above are not enough resort to using `?LAZY/1, ?LETSHRINK/1, resize, …`

More info on our PropEr website



PropEr

A QuickCheck-Inspired Property-Based Testing Tool for Erlang

Main | About | API | Download | FAQ | Publications | Tips | Tutorials

PropEr

Contents

- About: The PropEr developers
- API: The PropEr API and its documentation
- Download: With PropEr instructions on how to do this
- FAQ: Frequently Asked Questions with PropEr Answers
- Publications: PropEr papers and talks
- Tips: For the effective use of PropEr
- Tutorials: Showing the tool's PropEr use

Last edited on 2011-06-07.

Open vs. closed source tools

- All developers of PropEr are firm believers of the value of **open source** development tools
- Many advantages:
  - More eyes that read code $\Rightarrow$ more robust code
  - Easier to understand how the software works
  - Allows for user contributions (hopefully!)
  - ...
- More advantages if software is also free:
  - The Greek population can afford them!

PropEr licence

- GPL v3
- Long series of mail exchanges with the FSF
- …
- For the time being:
  - **GPL v3 but ...**
  - **with a (still implicit) Open-Source Exception**

Thanks from the PropEr developers!