An Erlang-based Framework for the Automatic Testing of Web Services

Kostis Sagonas

joint work with

Leonidas Lampropoulos

Overview

- Property-based Testing using PropEr
  - Short demo
- “Traditional” Testing of Web Services
- Testing of Web Services using Erlang
  - Based on PropEr, xmerl, and Yaws
- Automatic Response Testing of Web Services
  - Demo
- Property-based Testing of Web Services
  - Short demo
- Future Work

Property-based testing

- **Basic idea:**
  - express the properties that a program must satisfy in the form of input-output relations
  - try to find counter-examples for the property
  - … by automatically generating progressively more involved random test cases
  - … based on a general description of the structure of the tests

PropEr

A Property-based Testing Tool for Erlang

- Freely available as open source

  http://proper.softlab.ntua.gr

- Provides 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)

```
-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

```
%% 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]).
```

```
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))).
```

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,leaf,{node,6,leaf,leaf}}}
false
```

Traditional testing of web services

Similar to other forms of software testing:

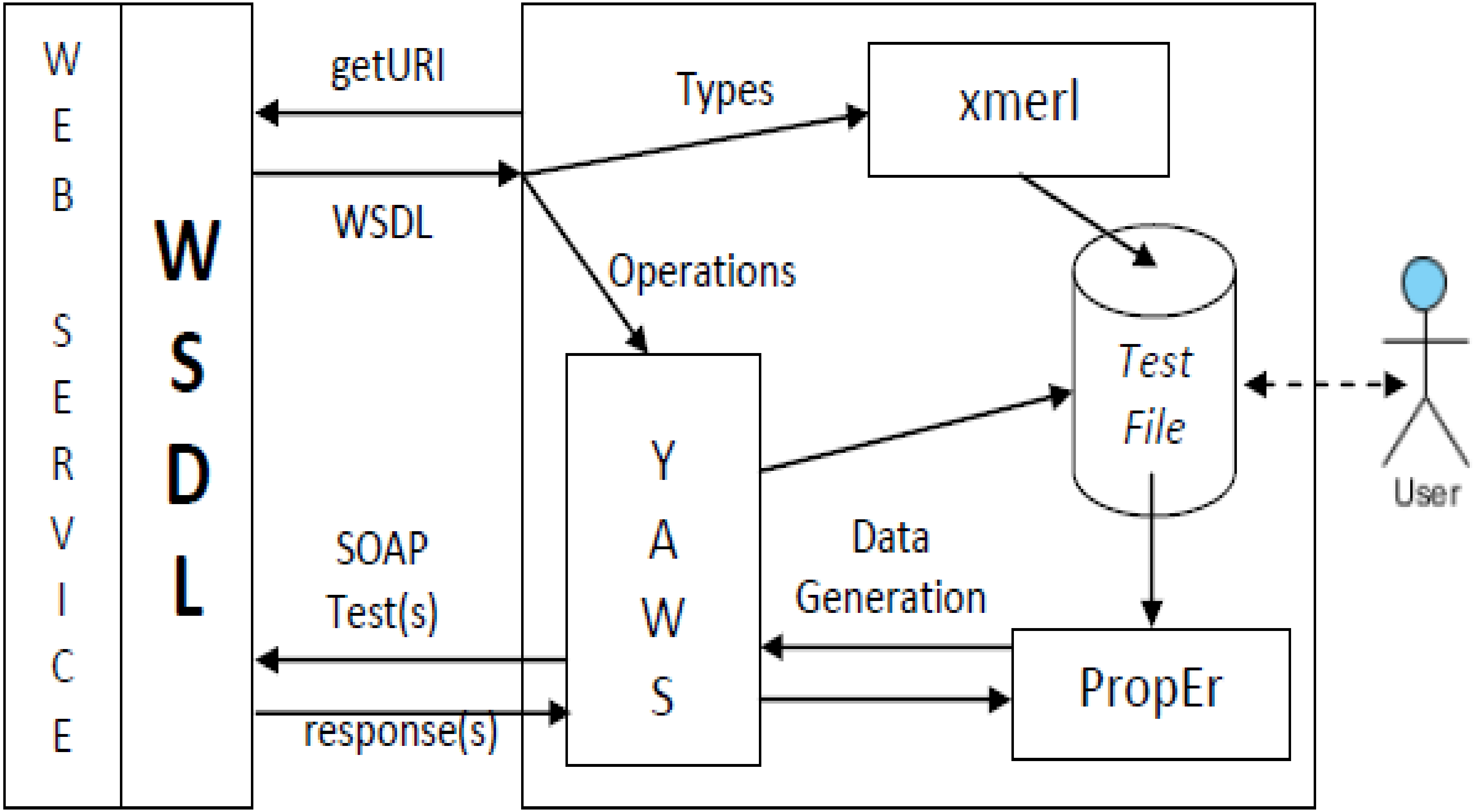
- Aquire valid input
  - User provides this
    (following the WSDL specification)
- Invoke operation
  - Automatically
    (using some existing framework, e.g. Yaws)
- Examine output
  - User checks this

PropEr testing of web services

Mostly automatic – goes as follows:

- Aquire valid input
  - Automatic using some PropEr generator (following the WSDL specification)
- Invoke operation
  - Automatically (using Yaws)
- Examine output
  - Automatic (for response testing)
  - Semi-automatic by writing some PropEr property (for property-based testing)

PropEr testing of web services



WSDL specification

A WSDL specification contains all the necessary information to invoke an operation

- Ports
- Bindings
- Messages
- Parts
- Most importantly (for us): Types!

WSDL types

- Included in a `<types>` XML tag
- Simple primitives
  - `int, long, string, boolean, ...`
- Aggregates
  - `list, union`
- Complex types
  - `sequence, choice, ...`
- Enumerations

A <types> example (www.webservicex.net)

```
<s:element name="ChangeCookingUnit">
```

```
<wsdl:types>
  <s:schema elementFormDefault="qualified“  targetNamespace="http://www.webserviceX.NET/">
    <s:element name="ChangeCookingUnit">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="CookingValue" type="s:double"/>
          <s:element minOccurs="1" maxOccurs="1" name="fromCookingUnit" type="tns:Cookings"/>
          <s:element minOccurs="1" maxOccurs="1" name="toCookingUnit“ type="tns:Cookings"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:simpleType name="Cookings">
      <s:restriction base="s:string">
        <s:enumeration value="drop"/>
        <s:enumeration value="dash"/>
             …
        <s:enumeration value="pinch”/>
        <s:enumeration value="TenCan"/>
       </s:restriction>
     </s:simpleType>
   </s:schema>
</wsdl:types>
```

A `<types>` example explained (1)

The "**`Cookings`**" simple type:

```
<s:restiction base="s:string">
```

is a restriction of the primitive type **`string`**

```
<s:enumeration value="drop"/>
```

adds a value to the enumeration

A <types> example explained (2)

The "**ChangeCookingUnit**" complex type:

```
<s:sequence>
```

is a sequence of the nested elements

```
<s:element minOccurs="1" maxOccurs="1"
   name="CookingValue" type="s:double"/>
```

adds a field "**CookingValue**" of type **double** that appears exactly once

Invoking web services with Yaws

```
yaws_soap_lib:call(WSDL_uri, Op, Args)
```

- The **`Args`** argument can become really complex
- Yaws needs most arguments converted to strings – but not all!
- For large WSDL specifications, writing the input by hand is error-prone

Automatic creation of generators

- Parse the WSDL specification
- Extract all type information
- Break types into primitives
- Handle Yaws string conversions
- Output Yaws records as a `.hrl` file
- Output PropEr generators!

Generators for the cooking example

```
generate_ChangeCookingUnit_1_CookingValue() ->
  ?LET(Gen, float(), float_to_list(Gen)).

generate_ChangeCookingUnit_1_fromCookingUnit_Cookings() ->
  elements(["drop","dash","pinch",...,"TenCan"]).

generate_ChangeCookingUnit_1_toCookingUnit_Cookings() ->
  elements(["drop","dash","pinch",...,"TenCan"]).

generate_ChangeCookingUnit_1() ->
  ?LET({Pr_ChangeCookingUnit_1_CookingValue,
        Pr_ChangeCookingUnit_1_fromCookingUnit_Cookings,
        Pr_ChangeCookingUnit_1_toCookingUnit_Cookings},
       {generate_ChangeCookingUnit_1_CookingValue(),
        generate_ChangeCookingUnit_1_fromCookingUnit_Cookings(),
        generate_ChangeCookingUnit_1_toCookingUnit_Cookings()},
       [Pr_ChangeCookingUnit_1_CookingValue,
        Pr_ChangeCookingUnit_1_fromCookingUnit_Cookings,
        Pr_ChangeCookingUnit_1_toCookingUnit_Cookings]).
```

Automatic response testing

- When an error occurs (server error, exceptions, out-of-bounds, etc.) a SOAP fault message is returned
- Conservatively accept every other response
- In this case the property creation is fully automatic

Property for the cooking example

```
prop_ChangeCookingUnit_responds() ->
  ?FORALL(Args, generate_ChangeCookingUnit_1(),
      case call_ChangeCookingUnit(Args) of
        {ok, _Attribs, [#'soap:Fault'{}]} -> false;
        {ok, _Attribs, _Result_record} -> true;
        _ -> false
      end).
```

Property-based testing of web services

- Use the tool to create a file with generators and properties
- Can use the created generators “as is”
- Simple to change them in order to refine them or add semantic information
- Can use the property with for response testing as our guide

Web service with delete example

```
-module(myDelete).
-export([handler/4]).

-include("myDelete.hrl").  % .hrl file generated by erlsom

handler(_Header, [#'p:delete'{'list'=List,'x' = X}],
        _Action, _SessionValue) ->
  {ok, undefined, get_response(List, X)}.

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]).

get_response(List, X) ->
  [#'p:deleteResponse'{anyAttribs = [],
                       deleteReturn = delete(X,List)}].
```

Automatic response test for delete

```
generate_delete_1_list() ->
  ?LET(Len, range(1, inf),
       vector(Len, integer(-2147483648, 2147483647))).

generate_delete_1_x() ->
  integer(-2147483648, 2147483647).

generate_delete_1() ->
  ?LET({Pr_delete_1_list, Pr_delete_1_x},
       {generate_delete_1_list(), generate_delete_1_x()},
       [Pr_delete_1_list, Pr_delete_1_x]).
```

```
prop_delete_responds() ->
  ?FORALL(Args, generate_delete_1(),
          case call_delete(Args) of
            {ok, _Attribs, [#'soap:Fault'{}]} -> false;
            {ok, _Attribs, _Result_record} -> true;
            _ -> false
          end).
```

Semi-automatic property testing

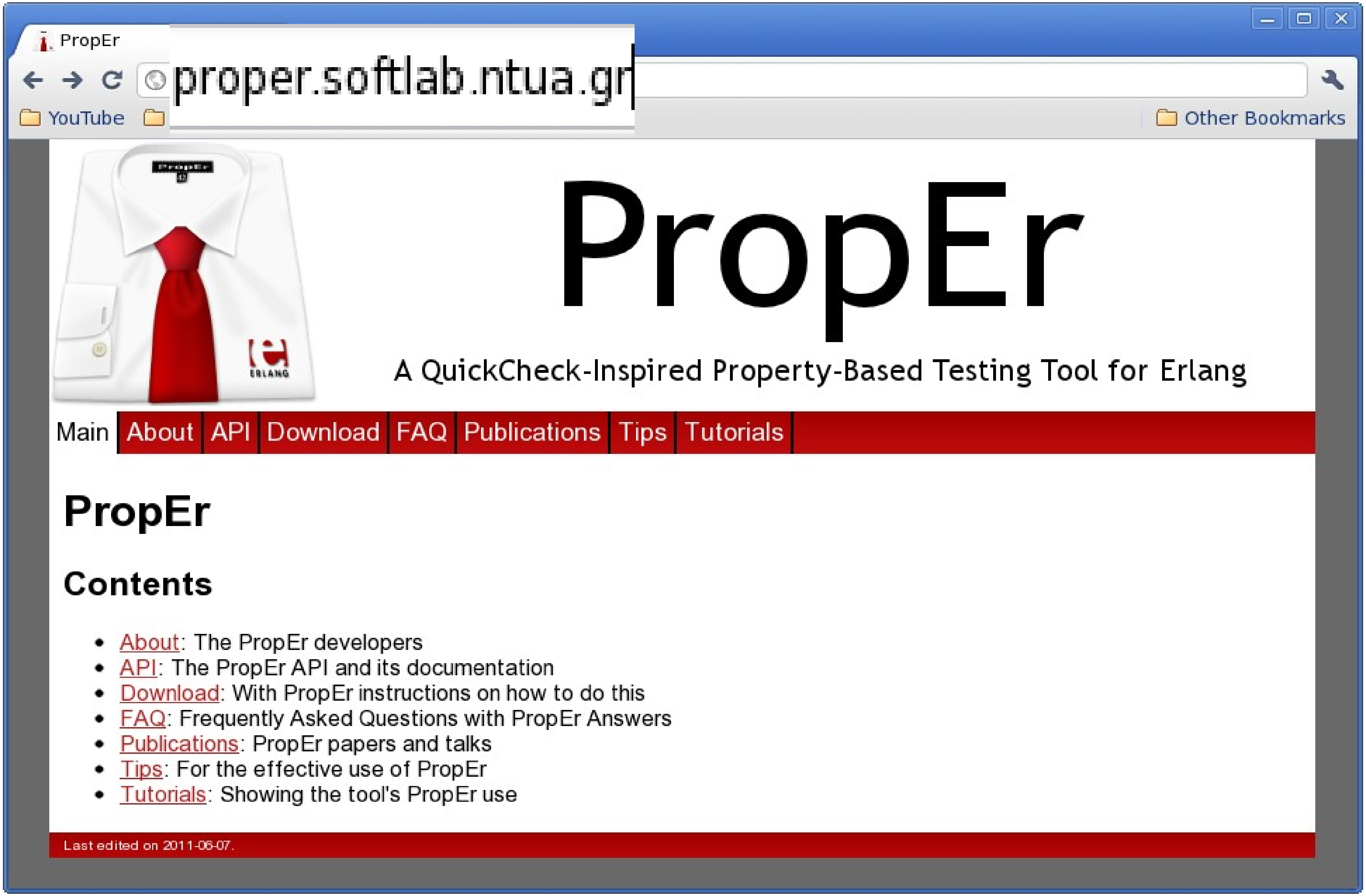
```
prop_delete_responds() ->
  ?FORALL([_L, X] = Args, generate_delete_1(),
          case call_delete(Args) of
            {ok, _Attribs, [#'soap:Fault'{}]} -> false;
            {ok, _Attribs,
              [#'p:deleteResponse'{
                  deleteReturn = undefined}]} -> true;
            {ok, Attribs,
              [#'p:deleteResponse'{
                  deleteReturn = RetList}]} ->
                not lists:member(X, RetList);
            _ -> false
          end).
```

Property-based testing

```
1> proper_ws:generate("file://tmp/myDelete.wsdl",
                      "proper_ws_myDelete").
ok
2> c(proper_ws_myDelete).
{ok,proper_ws_myDelete}
3> proper:quickcheck(
     proper_ws_myDelete:prop_delete_removes_every_x()).
.........................................!
Failed: After 42 test(s).
{[27,-86,-42,-14,90,10,-4,-32,8,44,4,-23,16,-42],-42}

Shrinking ..........(10 time(s))
{[0,0],0}
false
```

More info on our PropEr website



PropEr
proper.softlab.ntua.gr
YouTube
Other Bookmarks
ERLANG

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.