TARGETED PROPERTY-BASED TESTING

Andreas Löscher, Konstantinos Sagonas

andreas.loscher@it.uu.se, kostis@it.uu.se

Department of Information Technology

Uppsala University

Sweden

Outline

- Random Property-Based Testing
- Motivation
- Targeted Property-Based Testing

- Case Studies
- Concluding Remarks

UPPSALA UNIVERSITET

Property-Based Testing

- High-level, semi-automatic, black-box testing technique.
- Testing user-specified properties of the SUT.
- Examples:
  - QuickCheck (Haskell)
  - ScalaCheck (Scala)
  - PropEr (Erlang)
  - …

PropEr

A QuickCheck-Inspired Property-Based Testing Tool for Erlang

Random Property-Based Testing

- PBT tool provides:
  - Random generators for basic types
  - Language to write more complex generators
- PBT tool automatically tests these properties
  - Generate wide range of random inputs
  - Run the SUT with these inputs
  - Check if the properties hold

Random Property-Based Testing

Generator

```
prop_list_reverse() ->
  ?FORALL(L, proper_types:list(integer()),
      lists:reverse(lists:reverse(L)) == L).
```

Property should hold for all L

General Property

Random Property-Based Testing

```
L = []
L = [2]
L = [-5,-1,-8,1]
L = [16,3,-23]
L = [38,29,-28,12,-11,-3,-28,-6,9,-16,4,4]
…
```

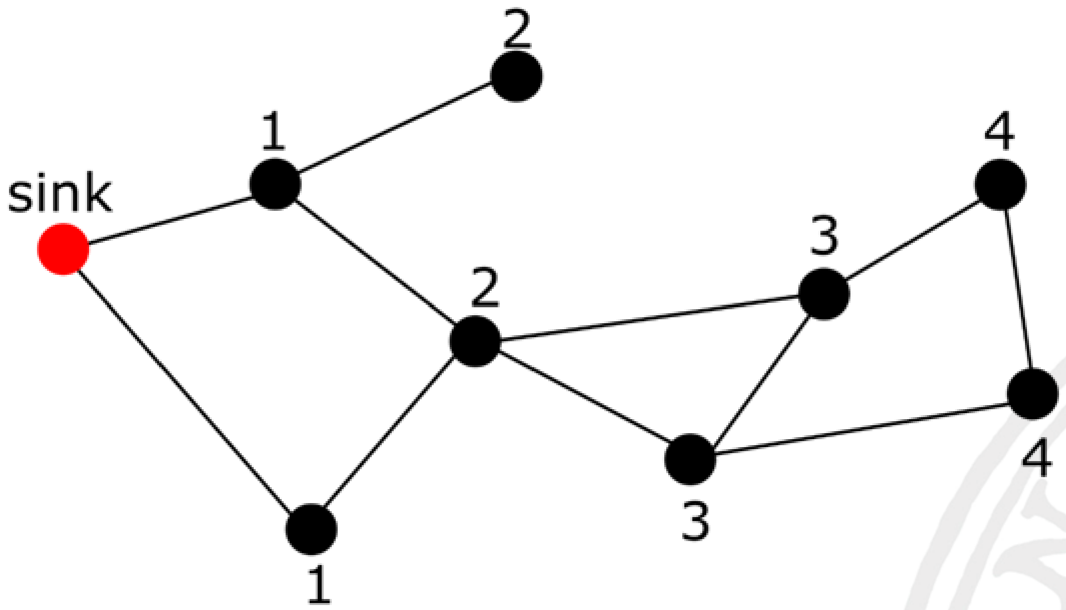
```
1> proper:quickcheck(example:prop_list_reverse(), 1000).
.................... 1000 dots ......................
OK: Passed 1000 test(s).
```

Graph Generator

```
graph(N) ->
  Vs = lists:seq(1, N),
  ?LET(Es, proper_types:list(edge(Vs)),
       {Vs, lists:usort(Es)}).

edge(Vs) ->
  ?SUCHTHAT({V1, V2}, {oneof(Vs), oneof(Vs)},
            V1 < V2).
```

Distance From Sink



On this graph, the maximum distance to sink is 4.

Is there a network with 42 nodes where the maximum distance to the sink > 21?

Distance From Sink

```
prop_length() ->
  ?FORALL(G, graph(42),
      begin
        L = lists:max(distance_from_sink(G)),
        L < 21
      end).
```

Distance From Sink

```
1> proper:quickcheck(example:prop_length(), 100000).
...................... 100000 dots .........................
OK: Passed 100000 test(s).
```

Same result for 1000 repetitions.

But we know that the property does not hold for some graphs.

Possible Solutions

- Write more involved generators?
- Guide input generation?

Possible Solutions

- ~~Write more involved generators?~~

- **Guide input generation!**
  - Using a search strategy.

Targeted Property-Based Testing

```
prop_length() ->
  ?FORALL(G, graph(42),
      begin
        L = lists:max(distance_from_sink(G))
        L < 21
      end).
```

- Use a search strategy to find a G that falsifies the property.
- Observe the relationship between G and L.

Targeted Property-Based Testing

- Combine Search Techniques with Property-Based Testing.
- Guide input generation towards input with high probability of failing.

- Gather information during test execution in form of **utility values** (**UVs**).
- UVs capture how close input came to falsifying a property.

Targeted Property-Based Testing

```
prop_length_hc() ->
  ?FORALL(G, graph(42),
      begin
        L = lists:max(distance_from_sink(G)),
        L < 21
      end).
```

Targeted Property-Based Testing

```
prop_length_hc() ->
  ?FORALL(G, graph(42),
      begin
        UV = lists:max(distance_from_sink(G)),
        UV < 21
      end).
```

Utility Values

Targeted Property-Based Testing

```
prop_length_hc() ->
  ?FORALL(G, graph(42),
      begin
        UV = lists:max(distance_from_sink(G)),
        ?MAXIMIZE(UV),
        UV < 21
      end).
```

Utility Values

Search Target

Targeted Property-Based Testing

```
prop_length_hc() ->
  ?FORALL(G, ?TARGET(graph(42)),
      begin
        UV = lists:max(distance_from_sink(G)),
        ?MAXIMIZE(UV),
        UV < 21
      end).
```

Generator the strategy controls

Utility Values

Search Target

UPPSALA UNIVERSITET

Targeted Property-Based Testing

```
prop_length_hc() ->
  ?TARGET_STRATEGY(hill_climbing,
  ?FORALL(G, ?TARGET(graph(42))),
      begin
        UV = lists:max(distance_from_sink(G)),
        ?MAXIMIZE(UV),
        UV < 21
      end)).
```

Search Strategy

Generator the strategy controls

Utility Values

Search Target

Targeted Property-Based Testing

```
prop_length_hc() ->
  ?TARGET_STRATEGY(hill_climbing,
  ?FORALL(G, ?TARGET(graph_hc(42))),
      begin
        UV = lists:max(distance_from_sink(G)),
        ?MAXIMIZE(UV),
        UV < 21
      end)).
```

Search Strategy

Generator the strategy controls

Utility Values

Search Target

Now **prop_length_hc** fails after 17,666 tests (on average).

Targeted Property-Based Testing

- Hill Climbing requires a neighborhood function
  - which, currently, needs to be supplied by the programmer
  - remove and add some random edges from/to the graph

Depends on the search strategy

- Hill Climbing can get stuck in local optima

→ Simulated Annealing is a better strategy

Targeted Property-Based Testing

```
prop_Target() ->
  ?TARGET_STRATEGY(SearchStrategy,
  ?FORALL(Input, ?TARGET(Params)),
          begin
            UV = SUT:run(Input),
            ?MAXIMIZE(UV),
            UV < Threshold
          end)).
```

Search Strategy

Generator the strategy controlls

Utility Values

Search Target

Case Study 1

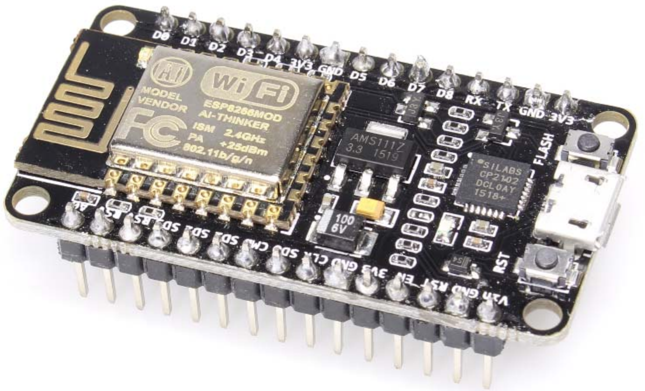
Setup:

- Sensor network
- Random distribution of UDB server and client nodes
- Client node periodically sends messages to server node

Test:

- Has X-MAC for any network a duty-cycle > 25%?

(duty-cycle ::= % time the radio is on)



Case Study 1

Random PBT

- Average amount of tests: **1188**
- Average time per tests: 23.5s
- **Mean Time to Failure: 7h46m**

Targeted PBT

- Average amount of tests: **200**
- Average time per tests: 40.6s
- **Mean Time to Failure : 2h12m**

Case Study 3

$$\frac{i(pc) = \mathit{Noop}}{\boxed{pc \mid s \mid m} \Rightarrow \boxed{pc{+}1 \mid s \mid m}} \quad (\textsc{Noop})$$

$$\frac{i(pc) = \mathit{Push}\ v}{\boxed{pc \mid s \mid m} \Rightarrow \boxed{pc{+}1 \mid v : s \mid m}} \quad (\textsc{Push})$$

$$\frac{i(pc) = \mathit{Pop}}{\boxed{pc \mid v : s \mid m} \Rightarrow \boxed{pc{+}1 \mid s \mid m}} \quad (\textsc{Pop})$$

- Definitions for an abstract machine.
- Test: Do these definitions fulfill a certain security criteria?

  (**Noninterference**)

Cătălin Hrițcu et al. "Testing noninterference, quickly." *Journal of Functional Programming,* 26 (2016).

Case Study 3

Random PBT

- **Naive**: generate random programs
- **ByExec**: generate program step-by-step one instruction a time; new instruction should not crash program

	Random PBT	
	Naive	ByExec
ADD	2234,08ms	312,97ms
LOAD	324028,34ms	987,91ms
STORE A	*timeout*	4668,04ms

Case Study 3

Targeted PBT

- **List**: programs are a list of instructions; using the built-in list generator for Simulated Annealing
- **ByExec**: neighbor of a program is a program with one more instruction

	Random PBT		Targeted PBT	
	Naive	ByExec	List	ByExec
ADD	2234,08	312,97	319,86	68,49
LOAD	324028,34	987,91	287,23	135,52
STORE A	–	4668,04	1388,09	263,94

Case Study 3

hand written; ca. 30 lines of additional code

	PBT		Target	
	Naive	ByExec	List	ByExec
ADD	2234,08	312,97	319,86	68,49
LOAD	324028,34	987,91	287,23	135,52
STORE A	–	4668,04	1388,09	263,94

1 line of code!

UPPSALA UNIVERSITET

Concluding Remarks

- Framework for Targeted Property-Based Testing.
- High-level expressive language for specifying properties.

- Compatible with random PBT.
- Two built-in strategies: hill climbing + simulated annealing.
- Infrastucture for additional search strategies.
- Fully integrated into PropEr.

PropEr

A QuickCheck-Inspired Property-Based Testing Tool for Erlang