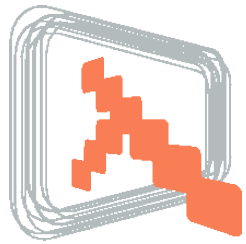
Testing What Should Work, Not What Shouldn't Fail

Samuel Rivas

LambdaStream S.L.
<samuel.rivas@lambdastream.com>

ProTest
property based testing

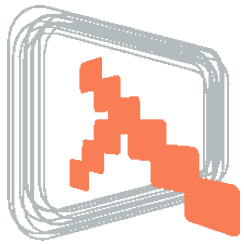
EUC'2010 Stockholm



Schedule

Today, we have two testing stories for you:

- **“Testing”** a classical story (with sad ending)
- **“Testing Reloaded”** A sci-fi remake (with property technology)
- **Behind the Scenes**

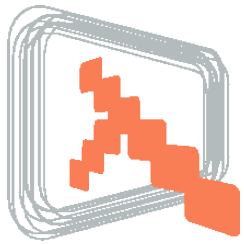



Testing
a classical story

Template Library

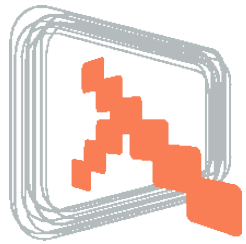
Specification

```
template:apply(
   "@lang@ rulez!", [{"lang", "Erlang"}]) ->

"Erlang rulez!"
```



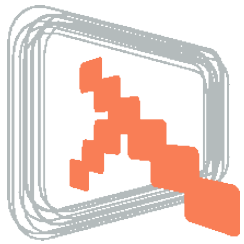
The Test Suite

```
?assertEqual(
    "Erlang rulez!", template:apply(
    "@lang@ rulez!",[{"lang","Erlang"}])

?assertEqual(
    "C sux!", template:apply(
    "@lang@ @what@!",[{"lang","C"},{"what","sux"}])

?assertEqual("Hello", template:apply("Hello",[]))

?assertEqual("", template:apply("",[]))
```



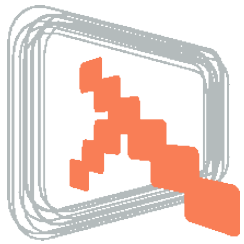
The Process

1. Implement the library
2. Test it
3. Fix it
4. Test it...



The Process

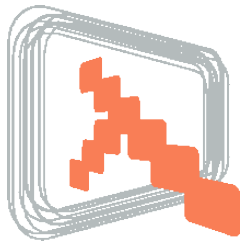
1. Implement the library
2. Test it
3. Fix it
4. Test it...
5. Fix it again
6. Test it
7. Ship it



The Process

1. Implement the library
2. Test it
3. Fix it
4. Test it...
5. Fix it again
6. Test it
7. Ship it

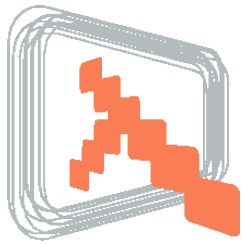
€€€ !!



Feature Request

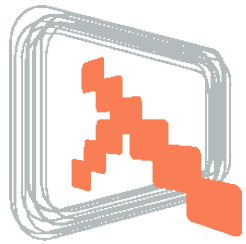
Specification

```
template:apply(
   "samuel.rivas@@lambdastream.com", []) ->

"samuel.rivas@lambdastream.com"
```



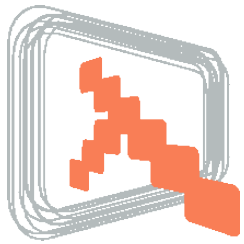
New Test Case

```
?assertEqual("@",template:apply("@@",[]))
```

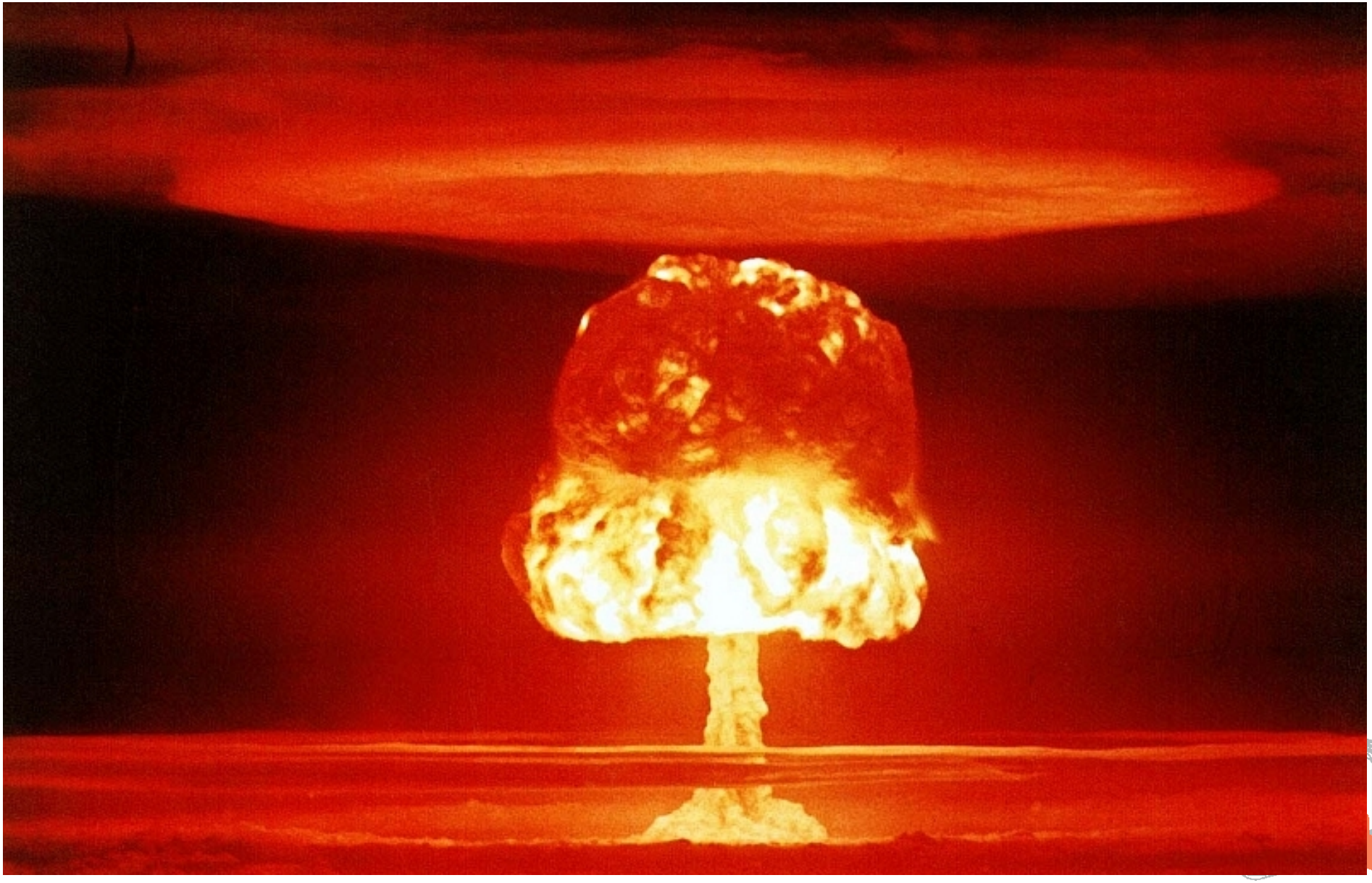


The Change Process

1. Modify the library
2. Test it
3. Fix it
4. Test it
5. Ship it



What the Customer Got



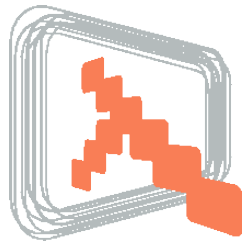
The Failure

Production Code

```
template:apply(

   "@name@@@@domain@",

   [{"name", "samuel.rivas" },
    {"domain", "lambdastream.com"}]).
```

Production Error

```
exception:  {variable_not_found,"name@@domain"}
```



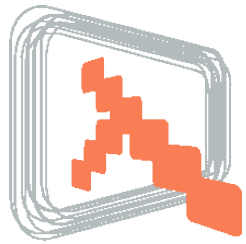
The Failure

Production Code

```
template:apply(

   "@name@@@domain@",

   [{"name", "samuel.rivas" },
    {"domain", "lambdastream.com"}]).
```

Production Error

```
exception:  {variable_not_found,"name@@domain"}
```



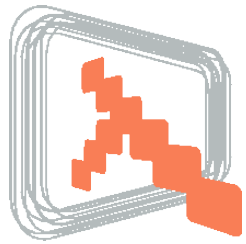
The Failure

Production Code

```
template:apply(

   "@name@@@@domain@",

   [{"name", "samuel.rivas" },
    {"domain", "lambdastream.com"}]).
```

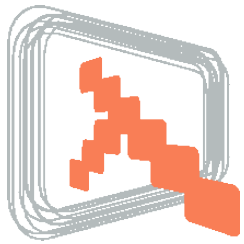
Production Error

```
exception:  {variable_not_found,"name@@domain"}
```



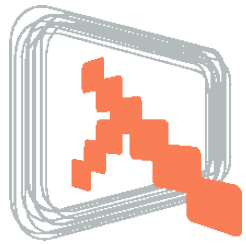
Testing What Should't Fail

And you though the computer would generalise ...



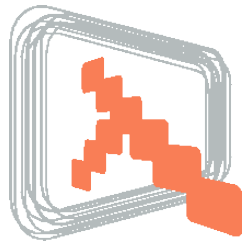
This Shouldn't Fail

```
?assertEqual(
    "Erlang rulez!", template:apply(
    "@lang@ rulez!",[{"lang","Erlang"}])

?assertEqual("@",template:apply("@@",[]))
```



Missing Case

```
?assertEqual(
  "foobar", template:apply(
  "@one@@two@",[{"one","foo"},{"two", "bar"}])
```

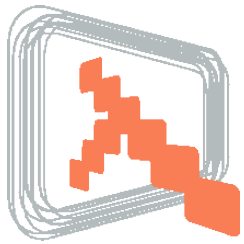



Testing Reloaded
A Property Based Testing Remake

Template Library

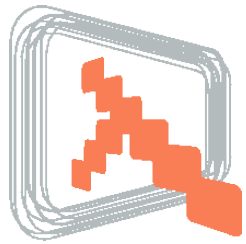
Specification

```
template:apply(
   "@lang@ rulez!", [{"lang", "Erlang"}]) ->

"Erlang rulez!"
```



The (Rough) Property

For all template T, list of variables X, and list of substitutions X', apply(T, X) must yield T with X values switched to X'

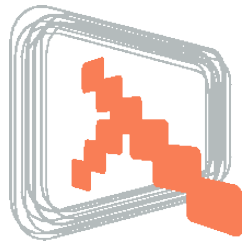


The (Rough) Property

For all template T, list of variables X, and list of substitutions X', apply(T, X) must yield T with X values switched to X'

Erlang Implementation

```
?FORALL(T, template(),
  template:apply(to_string(T), to_subs(T)) ==
  to_result(T)).
```



Change!

For all template T, list of variables X, and list of substitutions X', apply(T, X) must yield T with X values switched to X' and all escaped at symbols changed to at symbols

Erlang Implementation

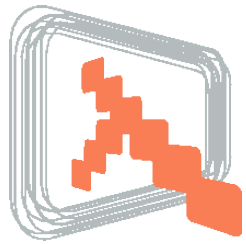
```
?FORALL(T,template(),

  template:apply(to_string(T), to_subs(T)) ==
  to_result(T)).
```

The Counterexample

The Change isn't Implemented Yet

```
prop_template: ..Failed! After 12 tests.
Shrinking.(1 times)
[escaped_at]
prop_template: Failed! After 1 tests.
[escaped_at]
```

```
Template: "@@"
Substs  : []
Expected: "@"
Got     : []
```

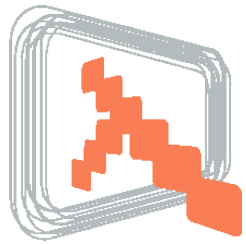


Another Counterexample

Hunting the Bug!

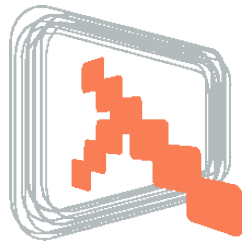
```
prop_template: ..Failed! After 16 tests.
Shrinking....(4 times)
[{var," ",[]},{var," ",[]}]
```

```
Template: "@ @@ @"
Substs  : [{" ", ""}]
Expected: ""
Got     : {error, {variable_not_found," @ "}}
```



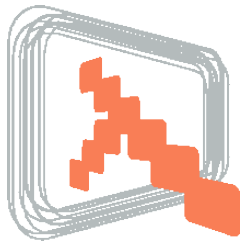
Testing What Should Always Work

For all template T, list of variables X, and list of substitutions X', apply(T, X) must yield T with X values switched to X' and all escaped at symbols changed to at symbols

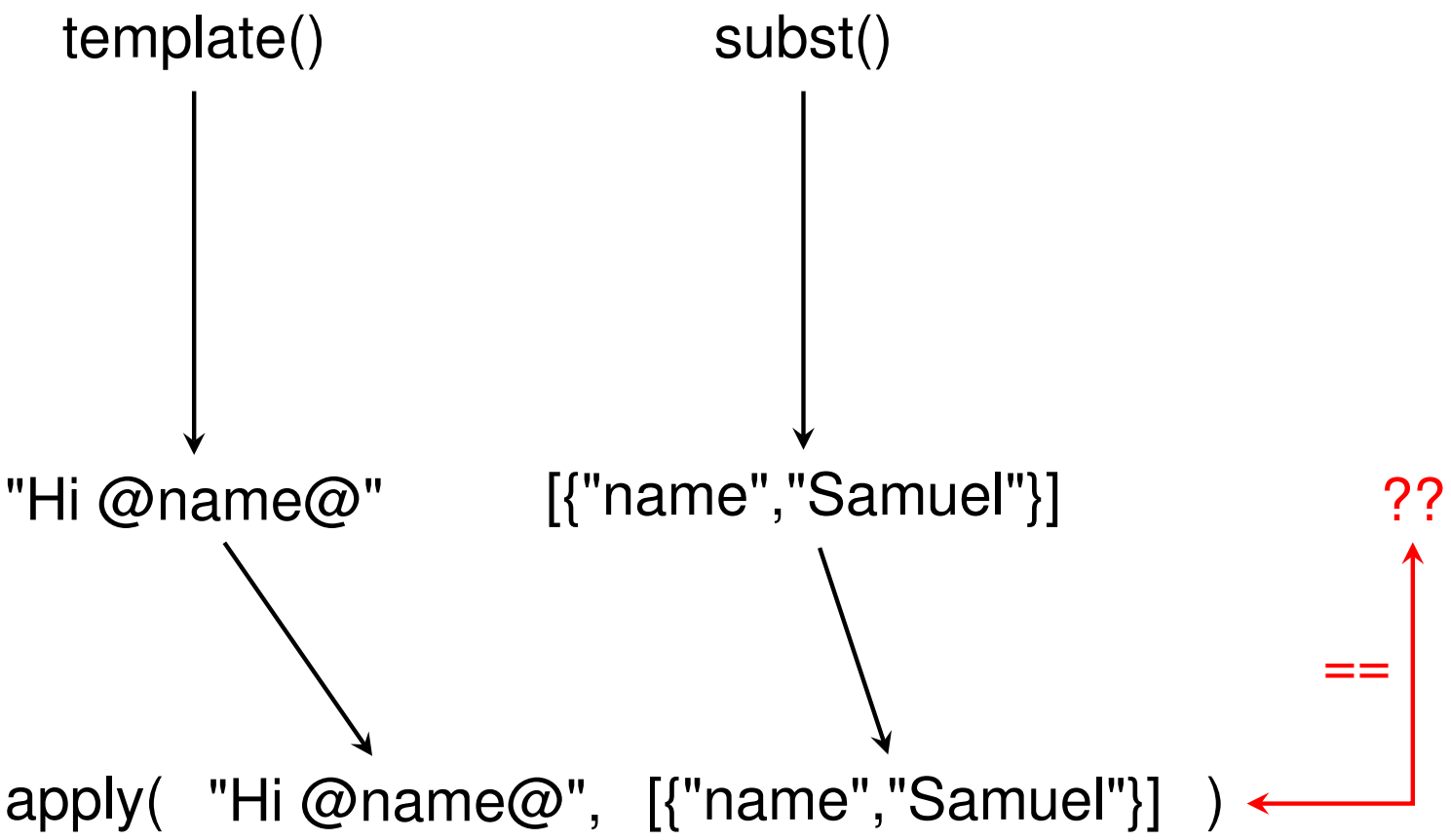


Behind the Scenes

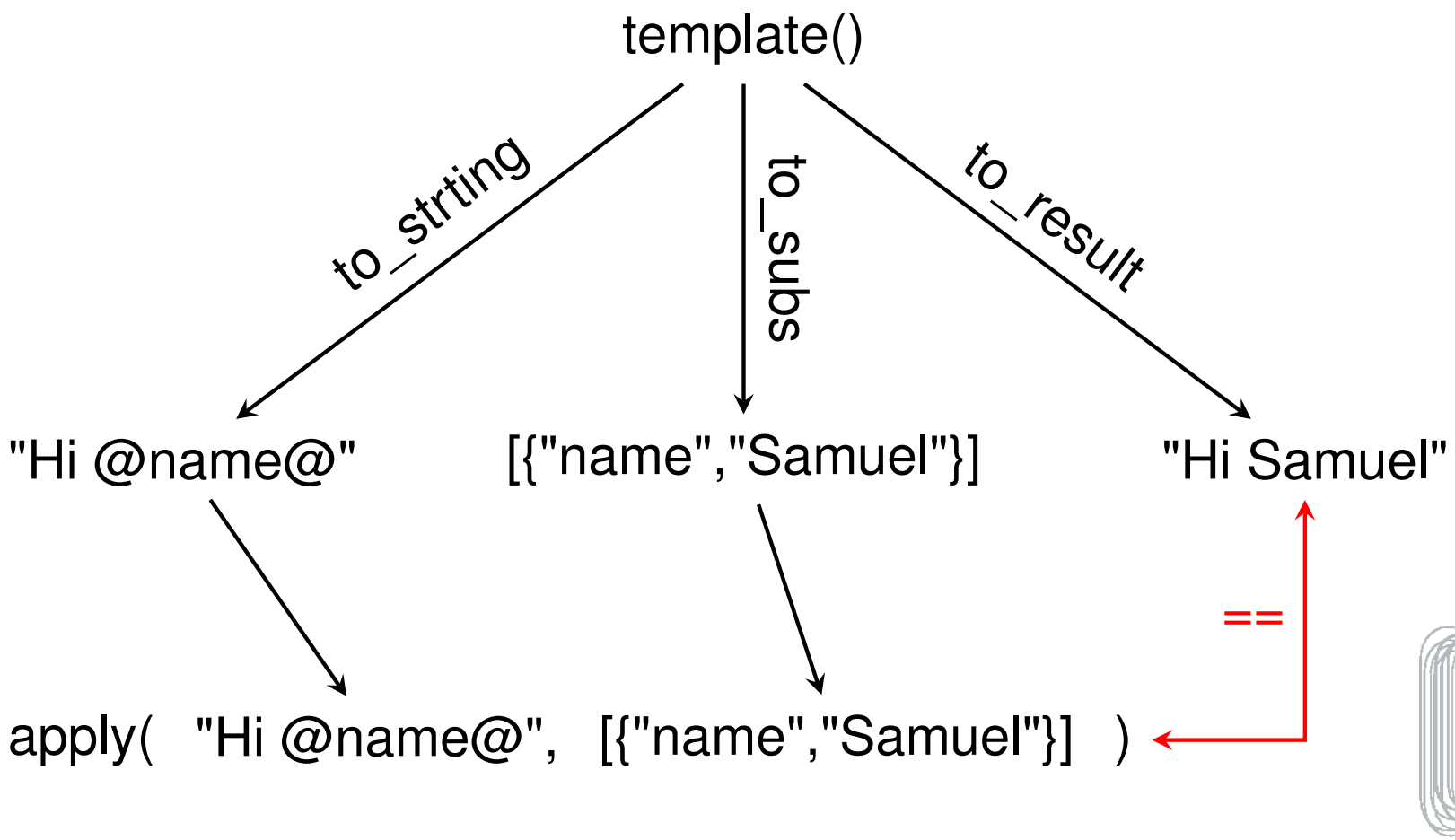
Looking under the hood



Generating Templates?

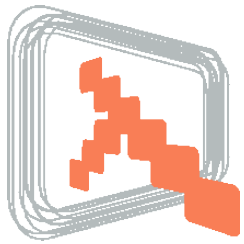


Symbolic Templates

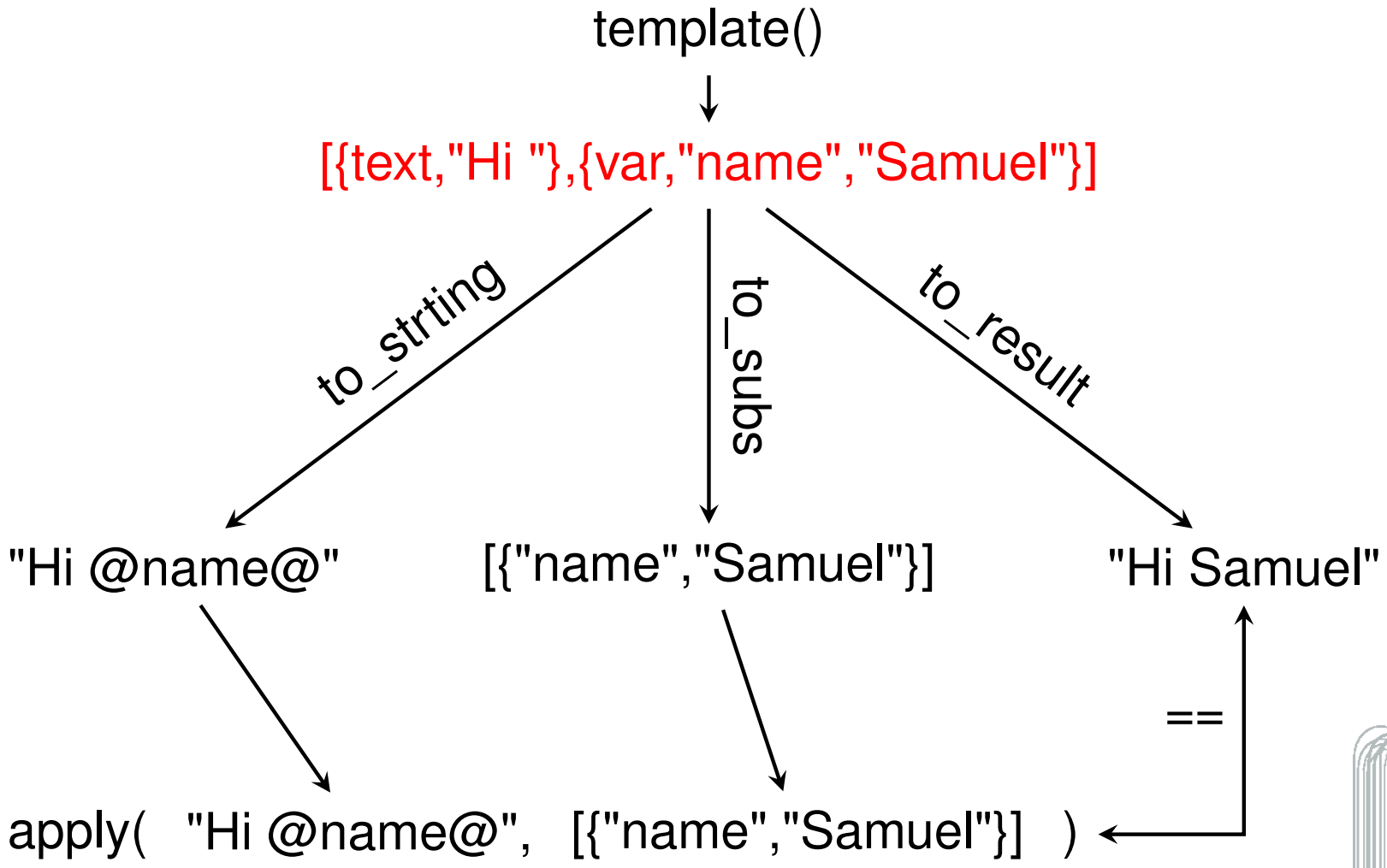


An Outline of the Symbolic Template Generator

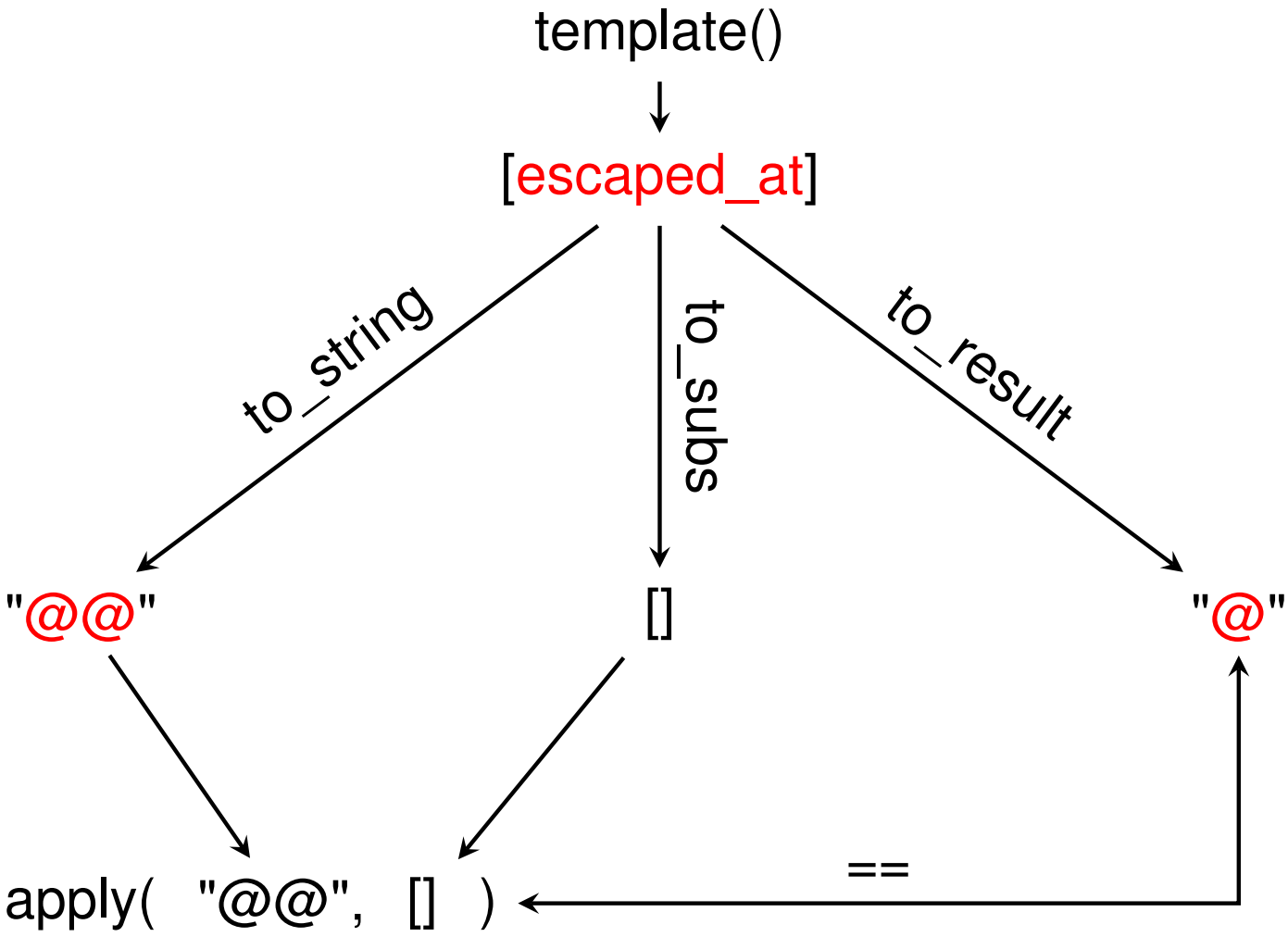
```
template() -> list(elements([text(),var()])).
     text() -> {text,string()}.
      var() -> {var,string(),string()}.
```



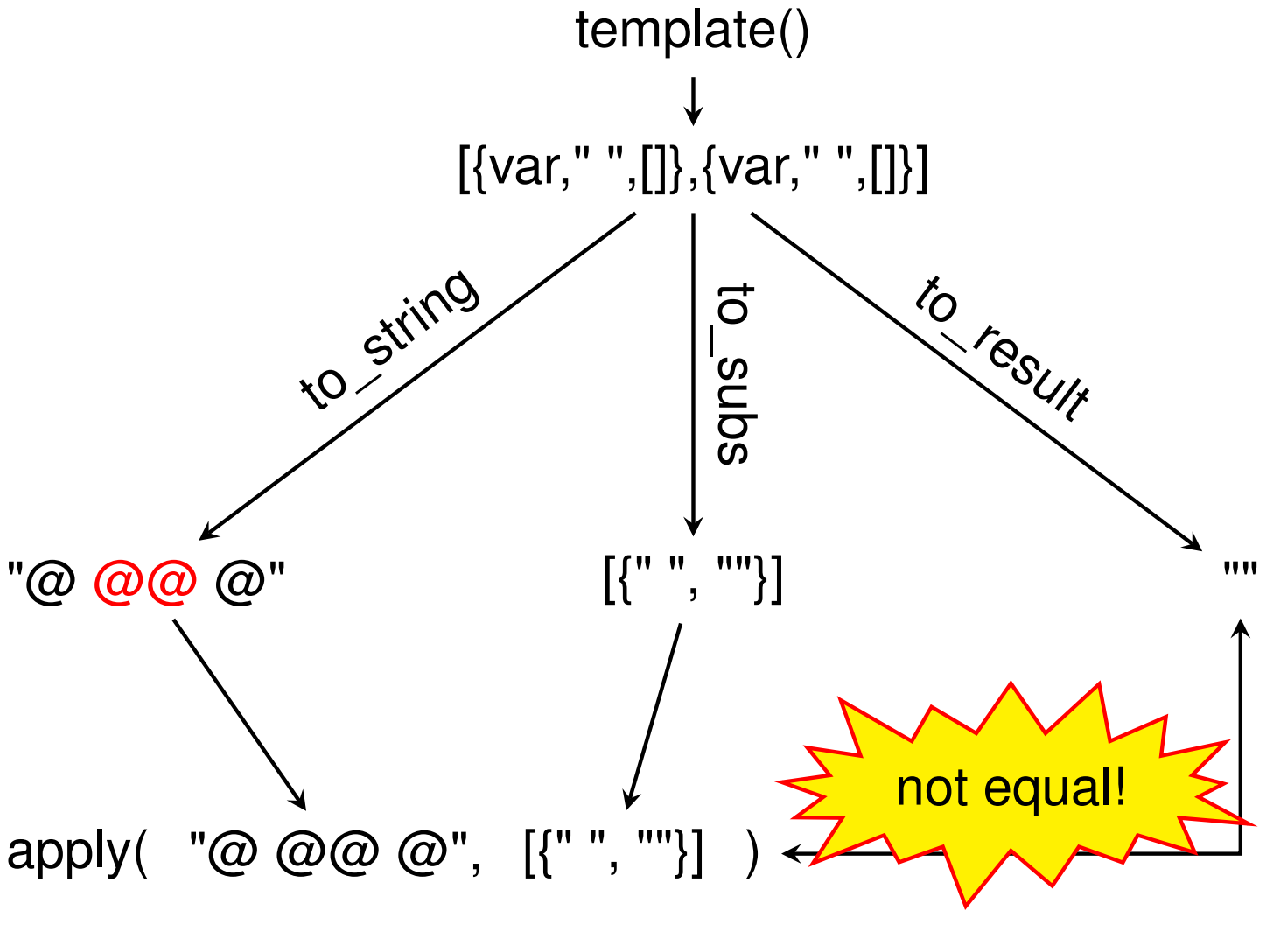
Symbolic Templates



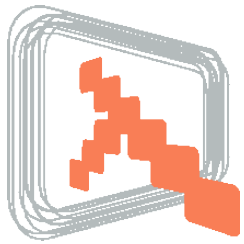
Templates 2.0 (With Escaped Ats)



The Counterexample Hero



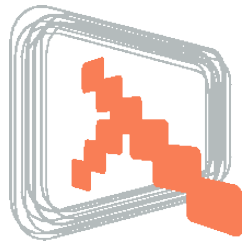
Conclusions



Conclusions

Test cases are **concrete behaviour examples**

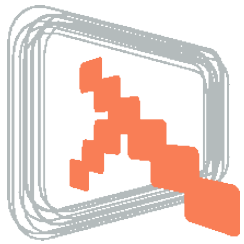
- They test *what shouldn't fail*
- Humans can abstract them in generic behaviour
- **computers can't generalise test cases to test generic behaviour**



Conclusions

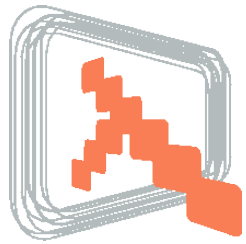
Properties test generic behaviour

- They test *what should always work*



Conclusions

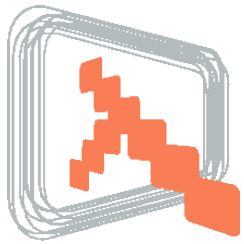
Properties resist change better than test cases



Thanks For Your Attention!

Questions?

Code available in Github:
http://github.com/lambdastream/tdd_template



Testing What Should Work, Not What Shouldn't Fail

Samuel Rivas

LambdaStream S.L.
<samuel.rivas@lambdastream.com>

ProTest
property based testing

EUC'2010 Stockholm