

Radama

QuickCheck

czyli deklaratywne testowanie oprogramowania

Miëtek Bak

mietek.bak@erlang-solutions.com

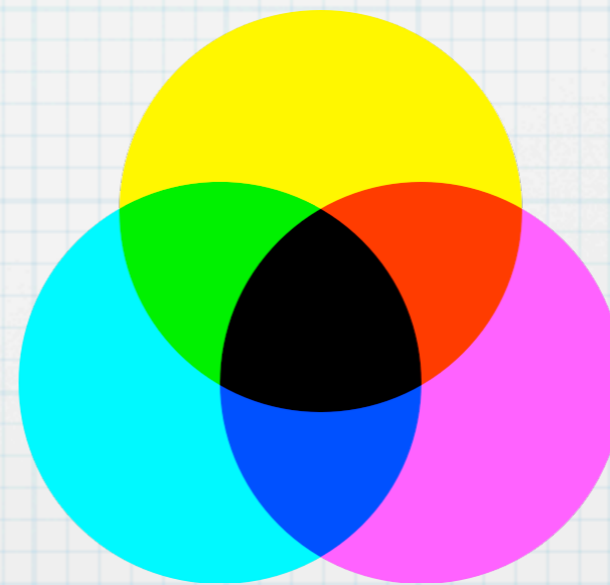
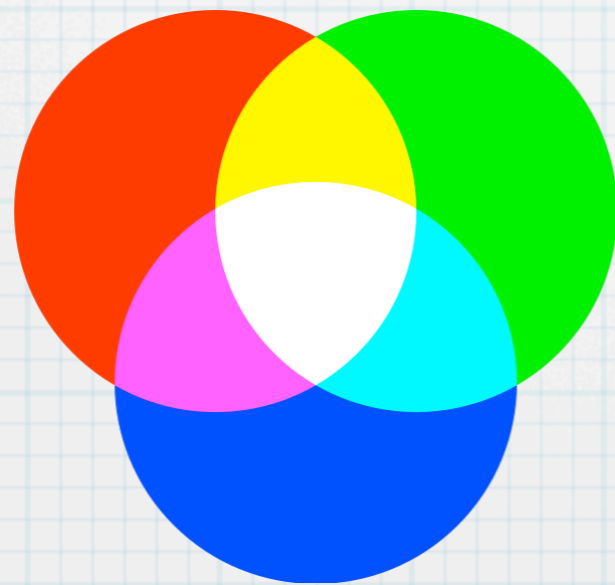
Systemy typów

	silne	słabe
statyczne	Java, Haskell, ML	C, C++
dynamiczne	Erlang, Python	JavaScript, PHP

Systemy barw

addytywne

subtraktywne



Haskell

h1.hs:

```
data Additive      = Red | Green | Blue
data Subtractive  = Cyan | Magenta | Yellow

fromAdditive :: Additive ->
              (Subtractive, Subtractive)
fromAdditive Red   = (Magenta, Yellow)
fromAdditive Green = (Cyan, Yellow)
fromAdditive Blue  = (Cyan, Magenta)
```

h2.hs:

```
data Additive      = Red | Green | Blue
data Subtractive  = Cyan | Magenta | Yellow
```

```
fromAdditive :: Additive ->
              (Subtractive, Subtractive)
fromAdditive Red    = (Magenta, Yellow)
fromAdditive Green = (Cyan, Yellow)
-- fromAdditive Blue = (Cyan, Magenta)
```

```
$ ghc -Wall h2.hs
```

```
Warning: Pattern match(es) are non-exhaustive
         In the definition of `fromAdditive':
           Patterns not matched: Blue in h2.hs
```

h3.hs:

```
data Additive      = Red | Green | Blue
data Subtractive  = Cyan | Magenta | Yellow
```

```
fromAdditive :: Additive ->
              (Subtractive, Subtractive)
```

```
fromAdditive Red   = (Magenta, Yellow)
```

```
fromAdditive Green = (Cyan, Yellow)
```

```
fromAdditive Blue  = 42
```

\$ ghc h3.hs

```
h3.hs: No instance for Num (Subtractive, Subtractive)
    arising from the literal `42'
```

h4.hs:

```
data Additive      = Red | Green | Blue
data Subtractive  = Cyan | Magenta | Yellow
```

```
fromAdditive :: Additive ->
              (Subtractive, Subtractive)
fromAdditive Red      = (Magenta, Yellow)
fromAdditive Green    = (Cyan, Yellow)
fromAdditive Bleu     = (Cyan, Magenta)
```

\$ ghc h4.hs

```
h4.hs: Not in scope: data constructor `Bleu'
```

Erlang

e1.erl:

```
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow};  
from_additive(blue)  -> {cyan, magenta}.
```

e2.erl:

```
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow}.  
% from_additive(blue) -> {cyan, magenta}.
```

```
$ erlc -Wall e2.erl
```

e3.erl:

```
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow};  
from_additive(blue)  -> 42.
```

```
$ erlc -Wall e3.erl
```

e4.erl:

```
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow};  
from_additive(bleu)  -> {cyan, magenta}.
```

```
$ erlc -Wall e4.erl
```

Erlang + Dialyzer

ed1.erl:

```
-type additive()      :: red | green | blue.  
-type subtractive()  :: cyan | magenta | yellow.  
  
-spec from_additive(additive()) ->  
      {subtractive(), subtractive()}.  
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow};  
from_additive(blue)  -> {cyan, magenta}.  
  
test() -> from_additive(blue).
```

ed2.erl:

```
-type additive()      :: red | green | blue.  
-type subtractive()  :: cyan | magenta | yellow.  
  
-spec from_additive(additive()) ->  
      {subtractive(), subtractive()}.  
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow}.  
% from_additive(blue) -> {cyan, magenta}.  
  
test() -> from_additive(blue).
```

```
$ dialyzer -n --src -r .
```

```
ed2.erl: Function test/0 has no local return  
ed2.erl: The call ed2:from_additive('blue') will  
never return since it differs in argument position 1  
from the success typing arguments: ('green' | 'red')
```

ed3.erl:

```
-type additive()      :: red | green | blue.
-type subtractive()  :: cyan | magenta | yellow.

-spec from_additive(additive()) ->
      {subtractive(), subtractive()}.
from_additive(red)    -> {magenta, yellow};
from_additive(green) -> {cyan, yellow};
from_additive(blue)  -> 42.

test() -> from_additive(blue).
```

```
$ dialyzer -Wspecdiffs -n --src -r .
```

```
ed3.erl: Type specification ed3:from_additive
(additive()) -> {subtractive(),subtractive()} is not
equal to the success typing: ed3:from_additive('blue'
| 'green' | 'red') -> 42 | {'cyan','yellow'} |
{'magenta','yellow'}
```

ed4.erl:

```
-type additive()      :: red | green | blue.  
-type subtractive()  :: cyan | magenta | yellow.  
  
-spec from_additive(additive()) ->  
      {subtractive(), subtractive()}.  
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow};  
from_additive(bleu)   -> {cyan, magenta}.  
  
test() -> from_additive(blue).
```

```
$ dialyzer -n --src -r .
```

```
ed4.erl: Function test/0 has no local return  
ed4.erl: The call ed4:from_additive('blue') will  
never return since it differs in argument position 1  
from the success typing arguments: ('bleu' | 'green'  
| 'red')
```

ed5.erl:

```
-type additive()      :: red | green | blue.
-type subtractive()  :: cyan | magenta | yellow.

-spec from_additive(additive()) ->
    {subtractive(), subtractive()}.
from_additive(red)    -> {magenta, yellow};
from_additive(green) -> {cyan, yellow};
from_additive(blue)  -> {cyan, magneta}.

test() -> from_additive(blue).
```

```
$ dialyzer -Wspecdiffs -n --src -r .
```

```
ed5.erl: Type specification ed5:from_additive
(additive()) -> {subtractive(),subtractive()} is not
equal to the success typing: ed5:from_additive('blue'
| 'green' | 'red') -> {'cyan', 'magneta' | 'yellow'} |
{'magenta', 'yellow'}
```

Erlang + QuickCheck

qc1.erl:

```
-define(ADDITIVE, [red, green, blue]).  
-define(SUBTRACTIVE, [cyan, magenta, yellow]).
```

```
additive() -> oneof(?ADDITIVE).
```

```
prop_from_additive(Module) ->  
  ?FORALL(A, additive(), begin  
    {S1, S2} = Module:from_additive(A),  
    lists:member(S1, ?SUBTRACTIVE) andalso  
    lists:member(S2, ?SUBTRACTIVE)  
  end).
```

e1.erl:

```
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow};  
from_additive(blue)  -> {cyan, magenta}.
```

```
> eqc:quickcheck(qc1:prop_from_additive(e1)).
```

```
.....
```

```
.....
```

```
OK, passed 100 tests
```

```
true
```

e2.erl:

```
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow}.  
% from_additive(blue) -> {cyan, magenta}.
```

```
> eqc:quickcheck(qc1:prop_from_additive(e2)).
```

Failed! Reason:

```
{'EXIT', {function_clause,  
          [{e2, from_additive, [blue]},  
           {qc1, '-prop_from_additive/1-fun-0-', 2},  
           ...]}}
```

After 1 tests.

blue

false

e3.er1:

```
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow};  
from_additive(blue)  -> 42.
```

```
> eqc:quickcheck(qc1:prop_from_additive(e3)).  
..Failed! Reason:  
{'EXIT', {{badmatch, 42},  
          [{qc1, '-prop_from_additive/1-fun-0-', 2},  
           ...]}}
```

After 3 tests.

blue

false

e4.erl:

```
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow};  
from_additive(bleu)  -> {cyan, magenta}.
```

```
> eqc:quickcheck(qc1:prop_from_additive(e4)).
```

```
.Failed! Reason:
```

```
{'EXIT', {function_clause,  
          [{e4, from_additive, [blue]},  
           {qc1, '-prop_from_additive/1-fun-0-', 2},  
           ...]}}
```

```
After 2 tests.
```

```
blue
```

```
false
```

e5.erl:

```
from_additive(red)    -> {magenta, yellow};  
from_additive(green) -> {cyan, yellow};  
from_additive(blue)  -> {cyan, magenta}.
```

```
> eqc:quickcheck(qc1:prop_from_additive(e5)).
```

```
.Failed! After 2 tests.
```

```
blue
```

```
false
```

Erlang + QuickCheck (2)

```
qc2.erl:
```

```
prop_lists_delete() ->  
  ?FORALL(I, int(),  
    ?FORALL(L, list(int()),  
      not lists:member(I, lists:delete(I, L)))).
```

```
> eqc:quickcheck(qc2:prop_lists_delete()).
```

```
.....
```

```
.....
```

```
OK, passed 100 tests
```

```
true
```

qc2.erl:

```
prop_lists_delete() ->
    ?FORALL(I, int(),
    ?FORALL(L, list(int())),
    not lists:member(I, lists:delete(I, L))).
```

```
> eqc:quickcheck(qc2:prop_lists_delete()).
```

```
.....
.....
```

```
OK, passed 100 tests
```

```
true
```

```
> eqc:quickcheck(qc2:prop_lists_delete()).
```

```
.....
.....Failed!
```

```
After 97 tests.
```

```
[-6, [-2, 7, -25, -23, -12, 3, -6, -6, -14, 26, -7]]
```

```
Shrinking... (3 times)
```

```
[-6, [-6, -6]]
```

```
false
```

```
qc3.erl:
```

```
prop_lists_delete() ->  
  ?FORALL(I, int(),  
    ?FORALL(L, list(int()),  
      collect({I, L}),  
      not lists:member(I, lists:delete(I, L)))).
```

```
> eqc:quickcheck(qc3:prop_lists_delete()).
```

```
.....  
.....
```

```
OK, passed 100 tests
```

```
3% {1, []}
```

```
3% {0, []}
```

```
2% {24, []}
```

```
2% {1, [1]}
```

```
2% {1, [-1]}
```

```
1% {29, [-26, 11]}
```

```
1% {26, [22, -3, -1, -16, 26, -31]}
```

```
1% {25, [6, 12, 10, -7, -17, 23]}
```

```
...
```

```
qc4.erl:
```

```
prop_lists_delete() ->  
  ?FORALL(I, int(),  
    ?FORALL(L, list(int()),  
      collect(lists:member(I, L),  
        not lists:member(I, lists:delete(I, L))))).
```

```
> eqc:quickcheck(qc4:prop_lists_delete()).
```

```
.....
```

```
.....
```

```
OK, passed 100 tests
```

```
95% false
```

```
5% true
```

```
true
```

```
qc5.erl:
```

```
prop_lists_delete() ->  
  ?FORALL(I, int(),  
    ?FORALL(L, list(int()),  
      ?IMPLIES(lists:member(I, L),  
        not lists:member(I, lists:delete(I, L))))).
```

```
> eqc:quickcheck(qc5:prop_lists_delete()).
```

```
XXXXXXXXXXXXXXXXX..X..XXX.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.XXXXXXXXXXX.XXXXXX.XXXXXX  
XXXXXXXXXXXXX.XX..XXXXXXX.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.XX.XXXX  
XXXXXXXXXXXXX.XXXX.X.X.XXXXXXXXXXXXXXXXXX.XXXFailed! After 20  
tests.
```

```
-15
```

```
[-4, -15, -15, 13, -17, 10, -21]
```

```
Shrinking...(3 times)
```

```
-15
```

```
[-15, -15]
```

```
false
```

```
qc6.erl:
```

```
prop_lists_delete() ->  
  ?FORALL(I, int(),  
  ?FORALL(L, list(int()),  
  ?IMPLIES(lists:member(I, L),  
  fails(  
    not lists:member(I, lists:delete(I, L)))))).
```

```
> eqc:quickcheck(qc6:prop_lists_delete()).  
XXXXXXXXXXXXXXXXXX.XXXXXXXXXXX.XXXXXXXXXX.XX.XXXXXXXXXXXXXXXXXX  
XXXXXXXXXXXXXXXXXX.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.XXXXXXXXXXXXXXXXXX  
.XXXXXXXXXXXXXXXXXX.XXXXXXXXXXXXXXXXXXX.XXXXXXXXXXXXXXXXXX..XXXXXX  
XX.XX.X.XXXXXXXXXXXXXXXXXX.XXXX..X.XX.XXXXXXXXXXXOK, failed as  
expected. After 21 tests.  
true
```

Erlang + QuickCheck (3)

```
qc7.erl:
```

```
prop_queue_last_cons() ->  
    ?FORALL(I, int(),  
            I == queue:last(queue:cons(I, queue:new()))).
```

```
> eqc:quickcheck(qc7:prop_queue_last_cons()).
```

```
.....  
.....
```

```
OK, passed 100 tests
```

```
true
```

qc8.erl:

```
queue() -> ?SIZED(N, queue(N)).
```

```
queue(0) -> queue:new();
```

```
queue(N) ->
```

```
    oneof([queue(0),  
          ?LET({I, Q}, {int(), queue(N - 1)},  
              queue:cons(I, Q))]).
```

```
> eqc_gen:sample(qc8:queue()).
```

```
{[-9], [-7, 0]}
```

```
{[-10], [-3, -11, 3]}
```

```
{[5], [5]}
```

```
{[], []}
```

```
{[], [15]}
```

```
{[], "\t"}
```

```
{[], [-16]}
```

```
{[-3], [17, -12, 8, -20, -16]}
```

```
...
```

qc8.er1:

```
prop_queue_last_cons() ->  
  ?FORALL({I, Q}, {int(), queue()},  
    queue:last(queue:cons(I, Q)) == I).
```

```
> eqc:quickcheck(qc8:prop_queue_last_cons()).  
.....Failed! After 10 tests.  
{-2, {[], [1]}}  
Shrinking.. (2 times)  
{0, {[], [1]}}  
false
```

qc9.erl:

```
queue() -> ?SIZED(N, queue(N)).
```

```
queue(0) -> {call, queue, new, []};
```

```
queue(N) ->
```

```
    ?LAZY(oneof([queue(0),  
                {call, queue, cons,  
                  [int(), queue(N - 1)]}]])).
```

```
> eqc_gen:sample(qc9:queue()).
```

```
{call, queue, new, []}
```

```
{call, queue, cons, [11, {call, queue, new, []}]}
```

```
{call, queue, cons, [-12, {call, queue, new, []}]}
```

```
{call, queue, cons, [2, {call, queue, cons, [17,  
    {call, queue, new, []}]}]}
```

```
{call, queue, new, []}
```

```
{call, queue, cons, [-8, {call, queue, new, []}]}
```

```
{call, queue, cons, [-10, {call, queue, cons, [-3,  
    {call, queue, new, []}]}]}
```

```
...
```

qc9.erl:

```
prop_queue_last_cons() ->
  ?FORALL({I, Q}, {int(), queue()},
    queue:last(queue:cons(I, eval(Q))) == I).
```

```
> eqc:quickcheck(qc9:prop_queue_last_cons()).
.....Failed! After 13 tests.
{-2, {call, queue, cons, [2, {call, queue, new, []}]}}
Shrinking..(2 times)
{0, {call, queue, cons, [1, {call, queue, new, []}]}}
false
```

```
qc10.erl:
```

```
model(Q) -> queue:to_list(Q).
```

```
prop_queue_last_cons() ->  
  ?FORALL({I, Q}, {int(), queue()},  
    model(queue:cons(I, eval(Q))) ==  
    model(eval(Q)) ++ [I]).
```

```
> eqc:quickcheck(qc10:prop_queue_last_cons()).  
...Failed! After 4 tests.  
{-1, {call, queue, cons, [0, {call, queue, new, []}]}}  
Shrinking.(1 times)  
{1, {call, queue, cons, [0, {call, queue, new, []}]}}  
false
```

```
qc11.erl:
```

```
model(Q) -> queue:to_list(Q).
```

```
prop_queue_last_cons() ->  
    ?FORALL({I, Q}, {int(), queue()},  
            model(queue:cons(I, eval(Q))) ==  
            [I | model(eval(Q))]).
```

```
> eqc:quickcheck(qc11:prop_queue_last_cons()).
```

```
.....  
.....
```

```
OK, passed 100 tests  
true
```

QuickCheck

- * Co jeszcze można testować?
 - * Systemy ze stanem
 - * Protokoły
 - * Automaty skończone
 - * Programy w obcych językach
- * Więcej informacji:
 - * <http://www.quvix.com/>
 - * <http://www.erlang-solutions.com/>

رمان