

Formal Specifications for Free!

Koen Claessen

John Hughes

Nick Smallbone



Algebraic Specifications

- Relate functions in an API to each other
 - Capture useful properties
 - Don't really say what they do

$$\begin{aligned}Xs ++ [] &== Xs \\ [] ++ Xs &== Xs \\ (Xs ++ Ys) ++ Zs &== Xs ++ (Ys ++ Zs)\end{aligned}$$

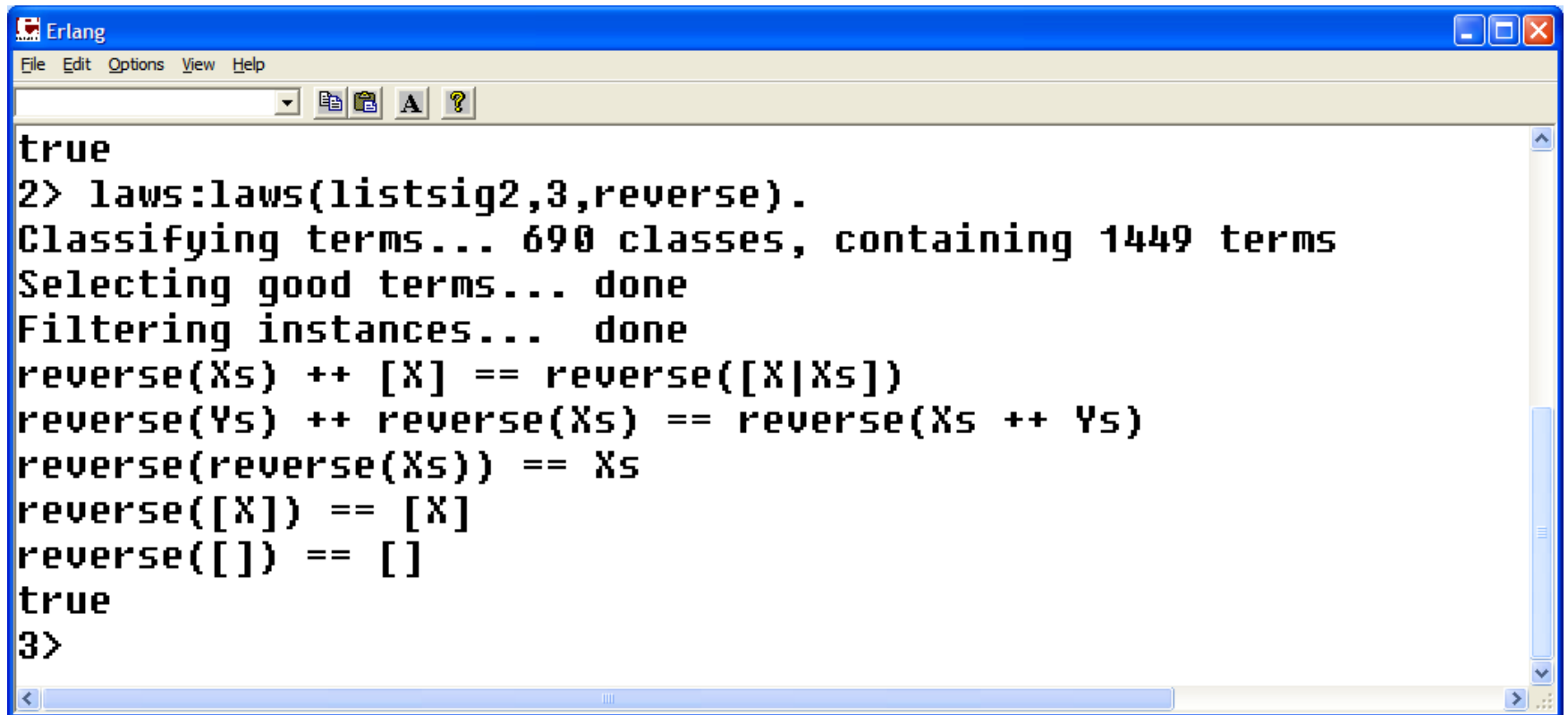
- Good for data-structures
- Only *pure functions* need apply!

DEMO

```
Erlang
File Edit Options View Help
Erlang (BEAM) emulator version 5.6.4 [smp:2] [async-threads:0]
Eshell V5.6.4 (abort with ^G)
1> laws:laws(listsig1,3).
Classifying terms... 534 classes, containing 1129 terms
Selecting good terms... done
Filtering instances... done
Xs ++ [] == Xs
(Xs ++ Ys) ++ Zs == Xs ++ (Ys ++ Zs)
[X] ++ Xs == [X|Xs]
[] ++ Xs == Xs
[X|Xs ++ Ys] == [X|Xs] ++ Ys
true
2> █
```

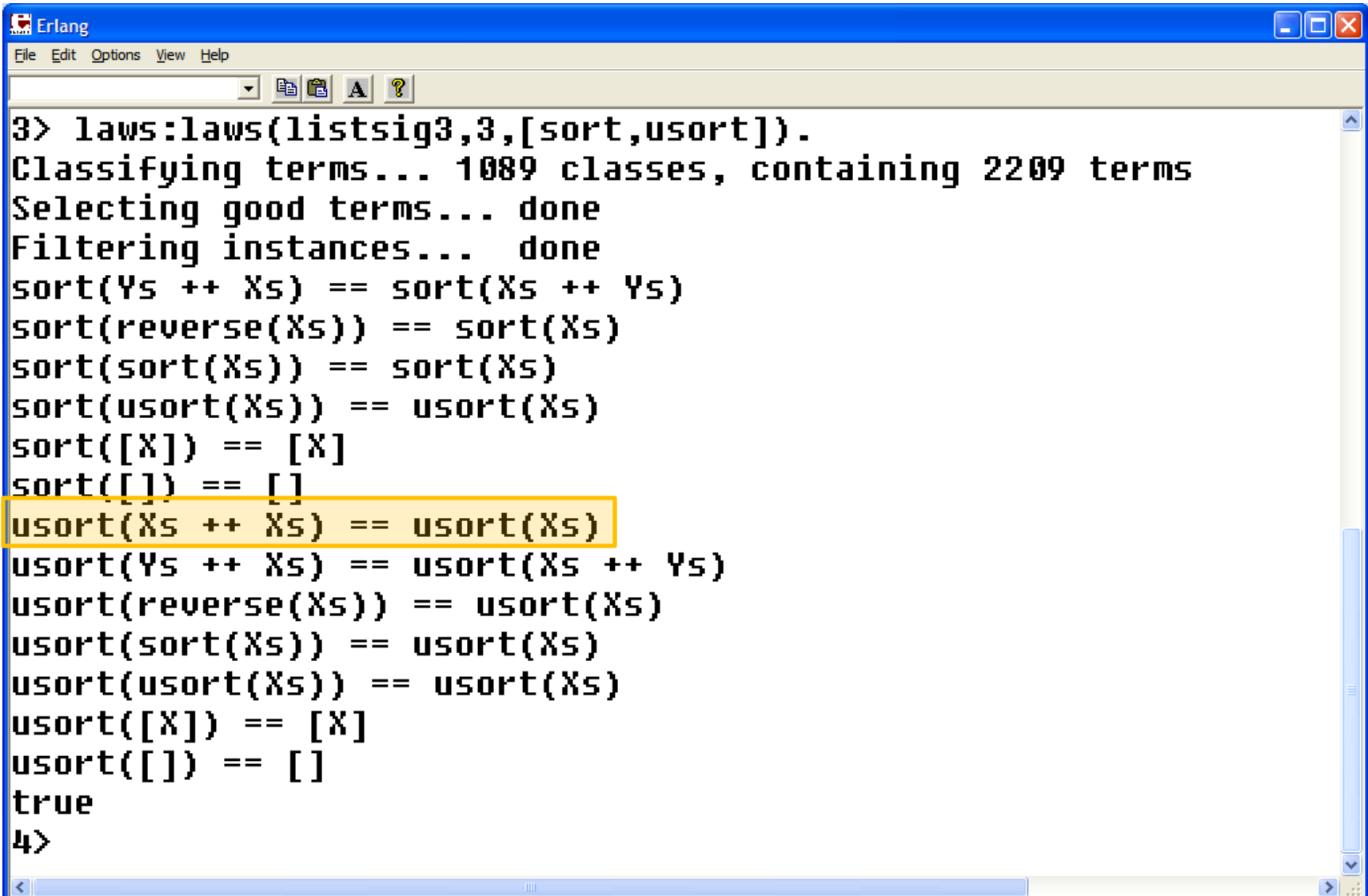
Complete algebraic specification of ++

Add reverse to the mix



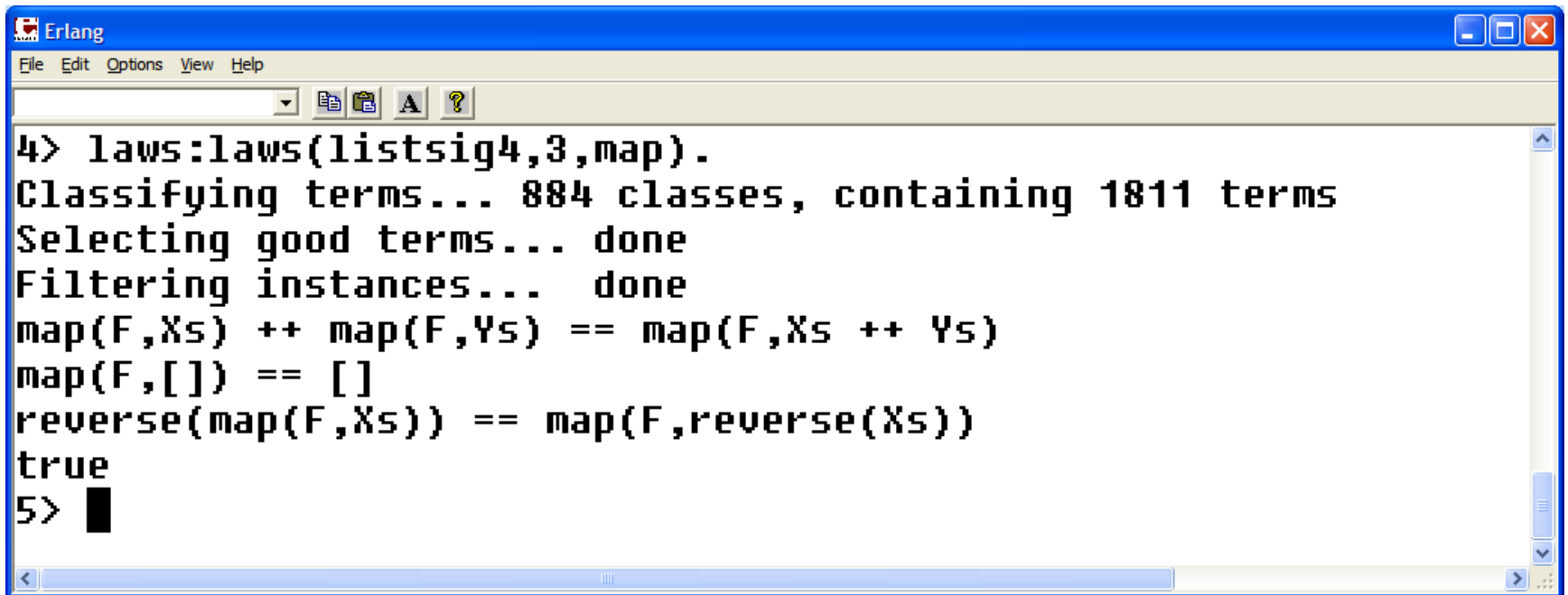
```
Erlang
File Edit Options View Help
true
2> laws:laws(listsig2,3,reverse).
Classifying terms... 690 classes, containing 1449 terms
Selecting good terms... done
Filtering instances... done
reverse(Xs) ++ [X] == reverse([X|Xs])
reverse(Ys) ++ reverse(Xs) == reverse(Xs ++ Ys)
reverse(reverse(Xs)) == Xs
reverse([X]) == [X]
reverse([]) == []
true
3>
```

What about sorting?



```
3> laws:laws(listsig3,3,[sort,usort]).
Classifying terms... 1089 classes, containing 2209 terms
Selecting good terms... done
Filtering instances... done
sort(Ys ++ Xs) == sort(Xs ++ Ys)
sort(reverse(Xs)) == sort(Xs)
sort(sort(Xs)) == sort(Xs)
sort(usort(Xs)) == usort(Xs)
sort([X]) == [X]
sort([]) == []
usort(Xs ++ Xs) == usort(Xs)
usort(Ys ++ Xs) == usort(Xs ++ Ys)
usort(reverse(Xs)) == usort(Xs)
usort(sort(Xs)) == usort(Xs)
usort(usort(Xs)) == usort(Xs)
usort([X]) == [X]
usort([]) == []
true
4>
```

What about map?



```
Erlang
File Edit Options View Help
4> laws:laws(listsig4,3,map).
Classifying terms... 884 classes, containing 1811 terms
Selecting good terms... done
Filtering instances... done
map(F,Xs) ++ map(F,Ys) == map(F,Xs ++ Ys)
map(F,[]) == []
reverse(map(F,Xs)) == map(F,reverse(Xs))
true
5> █
```

The new array library

- Flexible arrays, indexed from 0
- Purely functional updates

`new() -> array()`

Create a new, extendible array with initial size zero

`get(l::integer(), Array::array()) -> term()`

Get the value of entry l.

`set(l::integer(), Value::term(), Array::array()) -> array()`

Set entry l of the array to Value.

Get and Set

```
Erlang
File Edit Options View Help
12> laws:laws(arraysig1,3).
Classifying terms... 1460 classes, containing 2666 terms
Selecting good terms... done
Filtering instances... done
get(I,new()) == default_element()
get(I,set(I,X,A)) == X
get(I,set(J,default_element(),new())) == default_element()
get(J,set(I,X,new())) == get(I,set(J,X,new()))
set(I,X,set(I,Y,A)) == set(I,X,A)
set(J,X,set(I,X,A)) == set(I,X,set(J,X,A))
true
13> █
```

What happened to
`set(I,X,set(J,Y,A)) == set(J,Y,set(I,X,A))`?
It needs $I \neq J$!

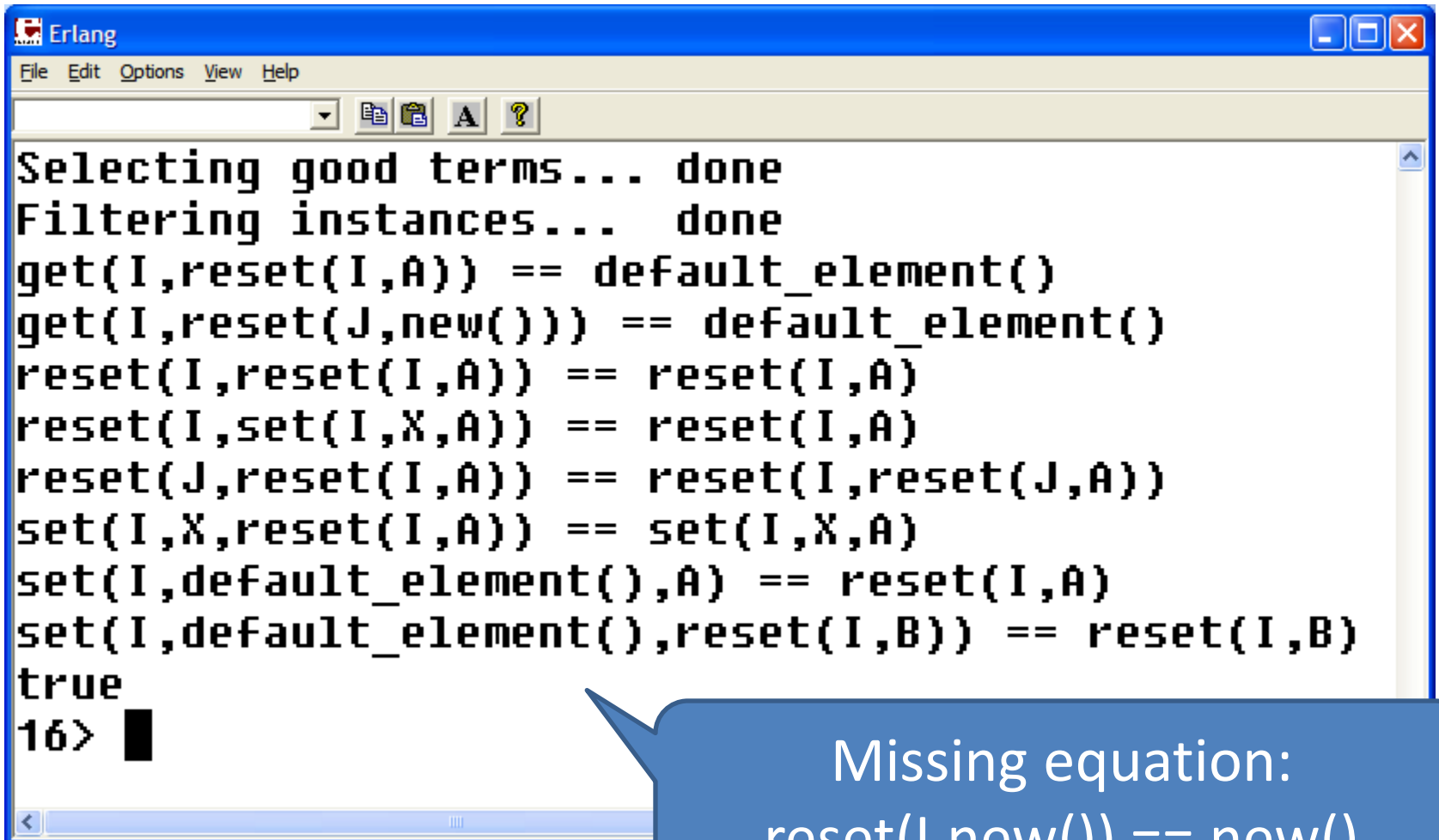
Adding Reset

- Elements can be reset to the default value

```
reset(I::integer(), Array::array()) -> array()
```

Reset entry I to the default value for the array.

Specifying reset



```
Erlang
File Edit Options View Help
Selecting good terms... done
Filtering instances... done
get(I,reset(I,A)) == default_element()
get(I,reset(J,new())) == default_element()
reset(I,reset(I,A)) == reset(I,A)
reset(I,set(I,X,A)) == reset(I,A)
reset(J,reset(I,A)) == reset(I,reset(J,A))
set(I,X,reset(I,A)) == set(I,X,A)
set(I,default_element(),A) == reset(I,A)
set(I,default_element(),reset(I,B)) == reset(I,B)
true
16> 
```

Missing equation:
reset(I,new()) == new()

Fix and Resize

- It is possible to *fix* the size of an array...

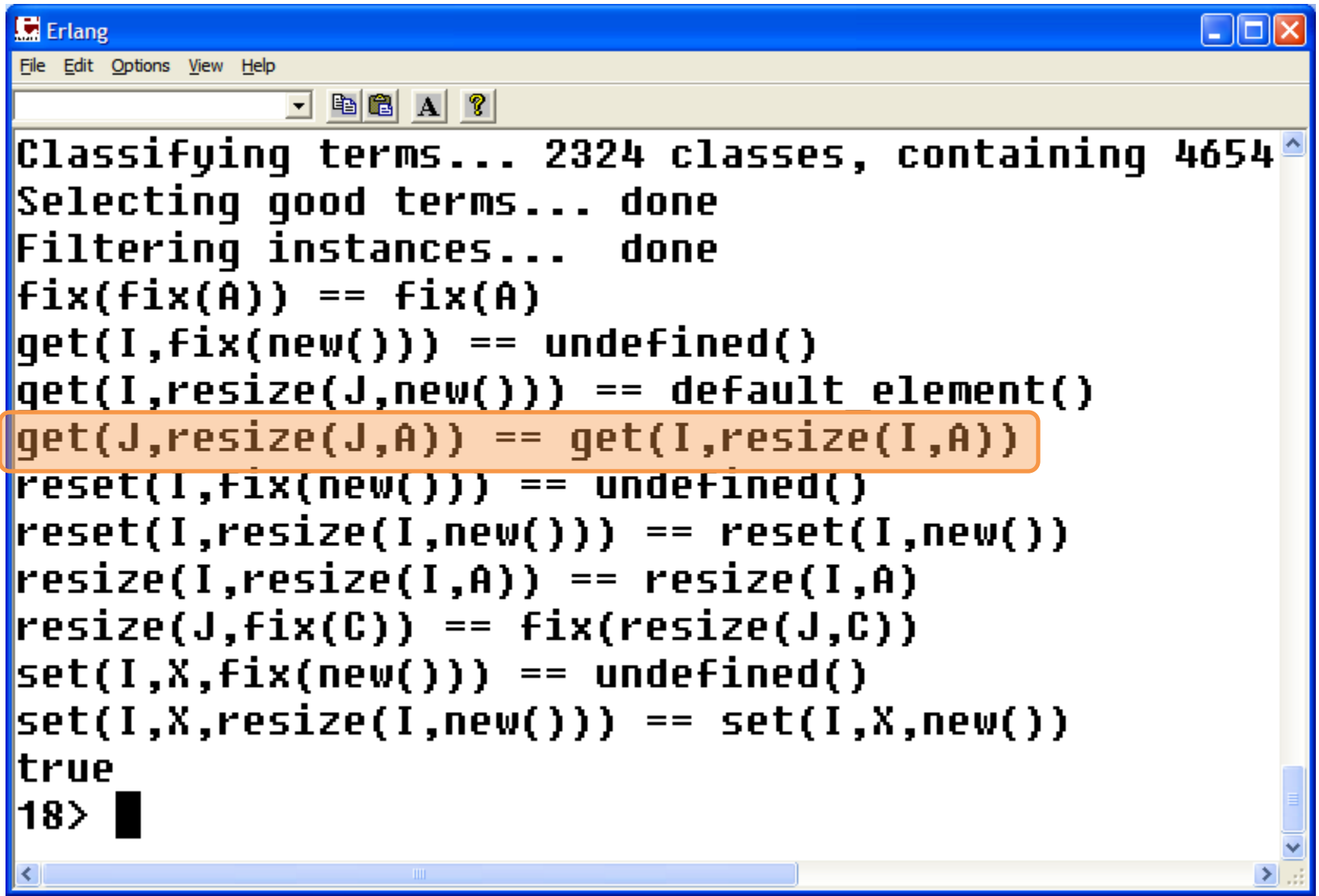
```
fix(Array::array()) -> array()
```

Fix the size of the array.

```
resize(Size::integer(), Array::array()) -> array()
```

Change the size of the array.

Specifying Fix and Resize

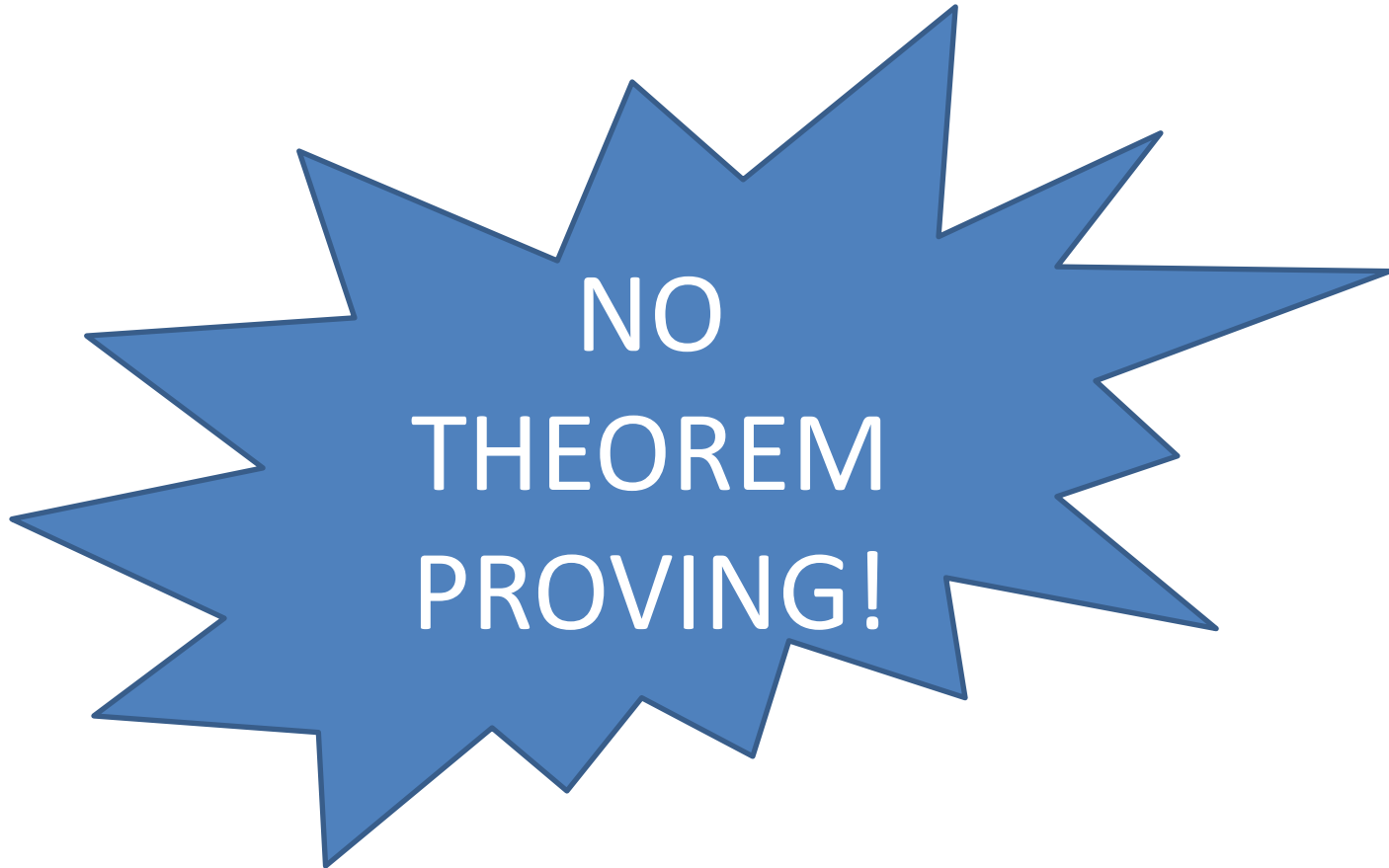


```
Erlang
File Edit Options View Help
Classifying terms... 2324 classes, containing 4654
Selecting good terms... done
Filtering instances... done
fix(fix(A)) == fix(A)
get(I,fix(new())) == undefined()
get(I,resize(J,new())) == default element()
get(J,resize(J,A)) == get(I,resize(I,A))
reset(I,fix(new())) == undefined()
reset(I,resize(I,new())) == reset(I,new())
resize(I,resize(I,A)) == resize(I,A)
resize(J,fix(C)) == fix(resize(J,C))
set(I,X,fix(new())) == undefined()
set(I,X,resize(I,new())) == set(I,X,new())
true
18> █
```

What use is this?

- Fun!
- Understanding
 - We learn things about the code by studying equations
- Design
 - *Missing equations* are a clue to possible improvements
- Testing
 - Easy to generate a QuickCheck test suite for regression testing

How does it work?



How does it work?

Print out the equations discovered

Xs
 $Xs++Ys$
 $reverse(Xs)$
 $reverse(reverse(Xs))$

Pick random values again:

$Xs==[3,4], Ys==[]$

given

Specifying which functions...

```
fun_types() ->  
  [{array,new,[],array},  
   {array,get,[index,array],elem},  
   {array,set,[index,elem,array],array},  
   {array,reset,[index,array],array},  
   {?MODULE,default_element,[],elem},  
   {array,fix,[array],array},  
   {array,resize,[index,array],array}  
  ].
```

Specifying which variables...

```
var_types() ->  
  {[x,y,z],elem},  
  {[a,b,c],array},  
  {[i,j,k],index}.
```

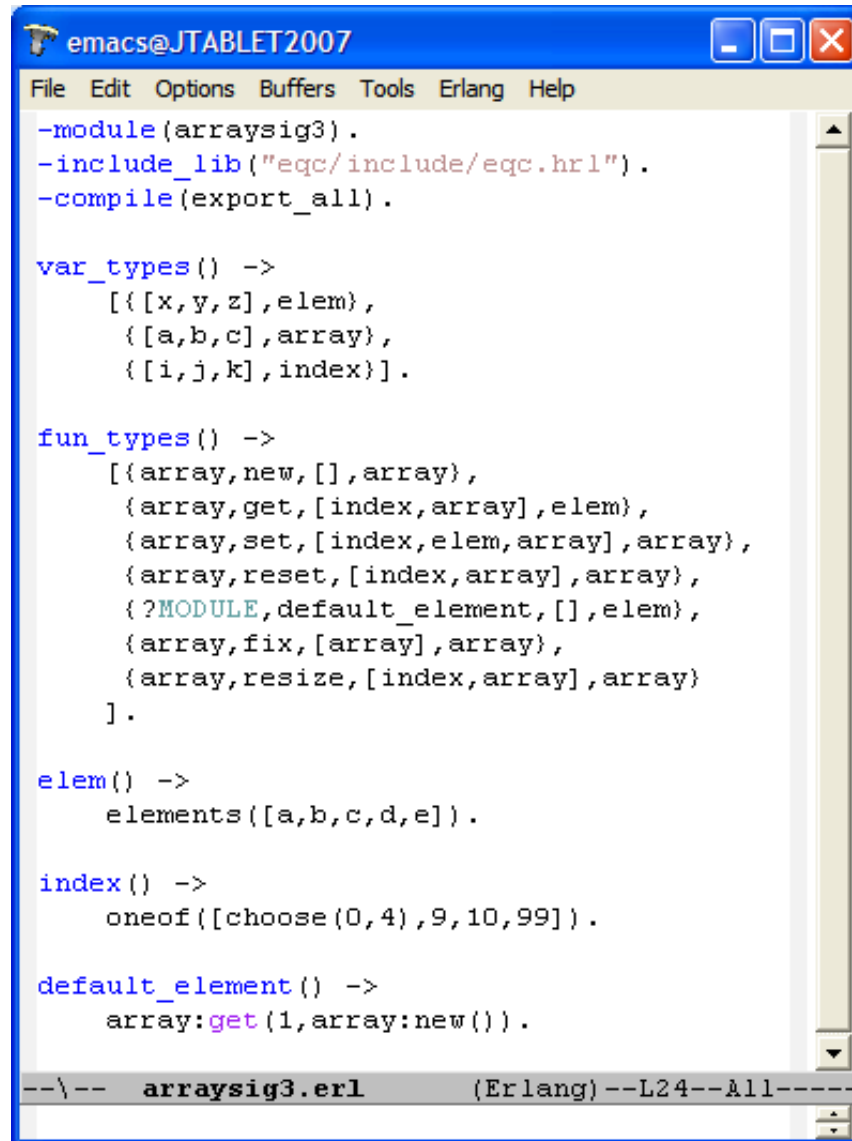
A few QuickCheck generators...

```
elem() ->  
  elements([a,b,c,d,e]).
```

```
index() ->  
  oneof([choose(0,4),9,10,99]).
```

- A generator for arrays is constructed automatically

That's It!



The image shows a screenshot of an Emacs window titled "emacs@JTABLET2007". The window contains Erlang code for a module named "arraysig3". The code defines the module, includes the "eqc" library, and sets compilation options. It then defines several types: "var_types" for variables, "fun_types" for functions, "elem" for elements, and "index" for indices. Finally, it defines a "default_element" function. The status bar at the bottom indicates the file name "arraysig3.erl" and the Erlang shell environment.

```
emac@JTABLET2007
File Edit Options Buffers Tools Erlang Help
-module(arraysig3).
-include_lib("eqc/include/eqc.hrl").
-compile(export_all).

var_types() ->
  [{[x,y,z],elem},
   {[a,b,c],array},
   {[i,j,k],index}].

fun_types() ->
  [{array,new,[],array},
   {array,get,[index,array],elem},
   {array,set,[index,elem,array],array},
   {array,reset,[index,array],array},
   {?MODULE,default_element,[],elem},
   {array,fix,[array],array},
   {array,resize,[index,array],array}
  ].

elem() ->
  elements([a,b,c,d,e]).

index() ->
  oneof([choose(0,4),9,10,99]).

default_element() ->
  array:get(1,array:new()).

--\-- arraysig3.erl (Erlang)--L24--All-----
```


Extensions

- Better filtering of equations
- Preconditions:
 - $I \neq J \implies \text{set}(I, X, \text{set}(J, Y, A)) == \text{set}(J, Y, \text{set}(I, X, A))$
- Abstractions:
 - For queues, $\text{tail}(\text{in}(X, Q)) \neq Q$, but *abstractly* they are the same
- Code with side-effects?

FREE

Formal
Specifications!

Get
yours
now!!!

quviq.com/euc2008.htm