Property Based Testing

Thomas Arts

- Test driven development
- Writing tests in Erlang

plan

Taken from a C course:
http://www2.its.strath.ac.uk/courses/c/

```
#include <stdio.h>
#define KILOS_PER_POUND .45359

main() {
 int pounds;
 printf(" US lbs UK st. lbs INT Kg\n");
 for(pounds=10; pounds < 250; pounds+=10)
 { int stones = pounds / 14;
   int uklbs = pounds % 14;
   float kilos = pounds * KILOS_PER_POUND;
   printf(" %d %d %d %f\n",
      pounds, stones, uklbs, kilos);
 }
}
```

A simple C program

Automatic testing of this conversion program really difficult

How can we write a program that is testable?

Write the tests first:

"Test driven development"

Testing a simple program

Characteristics of a good language for tests

- Easy to write correct code
- Pattern matching
- Expresive
- Good libraries (such as ASN.1 compiler, sockets, http client/server, etc)

and probably much more

Language to specify tests in

Tests first

```
-module(convert_test).
-compile(export_all).

test(lbs_to_stones) ->
   Pounds = 10,
   Pounds = convert:lbs_to_stones(Pounds) * 14.
```

Test Driven Development

DEMO

Test Driven Development

Tests first

```
-module(convert_test).
-compile(export_all).

test(lbs_to_stones) ->
   Pounds = 10,
   Pounds = convert:lbs_to_stones(Pounds) * 14.
```

Test Driven Development

Tests first

```
-module(convert_test).
-compile(export_all).

test(lbs_to_stones, Pounds) ->
   Pounds = convert:lbs_to_stones(Pounds) * 14.
```

Test Driven Development

Tests first

```
-module(convert_test).
-compile(export_all).

test_lbs_to_stones(Pounds) ->
   Pounds = convert:lbs_to_stones(Pounds) * 14.

test(lbs_to_stones) ->
   test_lbs_to_stones(250).
```

Test Driven Development

Data type list

[] empty list

[Head|Tail] element Head and list Tail

[1,2,3] shorthand for [1 | [2 | [3]]]

Erlang's list data type

List are typically traversed in a recursive function

example

```
f([]) ->
  0;
f([Head | Tail] ) ->
  Head + f(Tail).
```

Lists and recursion

Tests first

```
-module(convert_test).
-compile(export_all).

test_lbs_to_stones([]) ->
   true;
test_lbs_to_stones([Pounds|Rest]) ->
   Pounds = convert:lbs_to_stones(Pounds) * 14,
   test_lbs_to_stones(Rest).

test(lbs_to_stones) ->
   test_lbs_to_stones([10,14,15,61,83,250]).
```

Test Driven Development

Writing test first:

- specify which interface to implement
  - lbs_to_stones, lbs_to_uklbs, lbs_to_kilos
- find errors immediately

TRY it yourself in exercise

- write tests for convert
- write convert in Erlang!

Test Driven Development

- The eqc_c module lets you call C functions as if they were Erlang functions

convert.c

```
int lbs_to_stones (int pounds) {
  return pounds / 14;
}
```

```
17> eqc_c:start(convert).
ok
18> convert:lbs_to_stones(14).
1
```

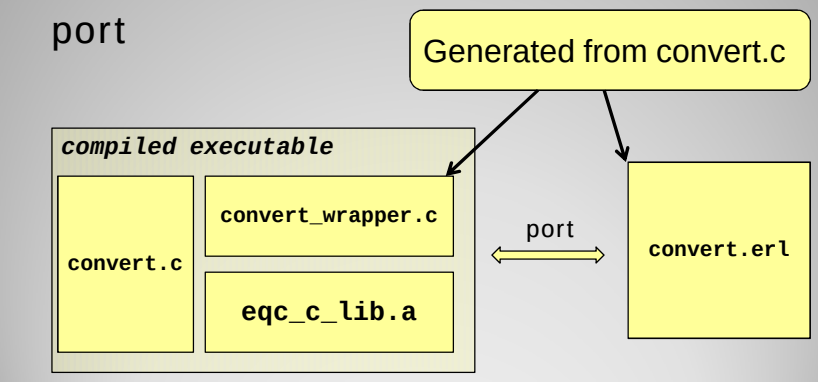
Generates a module convert from convert.c

Calling C code from Erlang

- Allow testing of C programs using Erlang
  - Safe interface: arguments are type checked before calling C functions
- Not built for using C libraries in Erlang programs
  - speed is not a primary concern

Purpose

- The C program is running in a separate OS process
- Communication over a port



How does it work?

- C programs are all about pointers and arrays
- How does this work on the Erlang side?
  - eqc_c provides functions for manipulating pointers and arrays
- Very basic interface
  - Used for creating test inputs and inspecting outputs

C programming in Erlang

Pointer operations: alloc/2, deref/1, store/2

```
eqc_c:alloc(Type, Val)  Allocate memory for a Type
                        and initialize to Val
eqc_c:deref(Ptr)        *Ptr
eqc_c:store(Ptr, Val)   *Ptr = Val;
```

```
13> P = eqc_c:alloc(int, 5).
{ptr,int,2098000}
14> eqc_c:deref(P).
5
15> eqc_c:store(P, 6).
ok
16> eqc_c:deref(P).
6
```

Pointers are tagged with their types

Working with pointers

Tuples are fixed size sequences of values

{T1,T2,T3} is a tuple of 3 elements, with T1, T2, and T3 arbitrary Erlang terms

Typically first element of a tuple is used as a tag. (Reminds about structs)

Erlang's tuple data structure

More pointer functions

	C Code
• free(Ptr)	**free(Ptr)**
◦ Calls the C free function to deallocate memory	
• cast_ptr(Type, Ptr)	**(Type *)Ptr**
◦ Change the type of a pointer	
◦ Use with care!	
• add_to_ptr(Ptr, N)	**(Ptr + N)**
◦ Advance a pointer N steps	
◦ Depends on the size of the type pointed to	

Working with arrays

- C arrays are represented by a pointer to the first element
- Functions: create_array/2, read_array/2, store_array/3

```
20> P = eqc_c:create_array(int, [1,2,3,4,5]).
{ptr,int,2098512}
21> eqc_c:read_array(P, 4).
[1,2,3,4]
22> eqc_c:store_array(P, [7,8,9]).
ok
23> eqc_c:read_array(P, 5).
[7,8,9,4,5]
```

You need to give the length of the array when reading it

Indexing arrays

- array_index(Ptr, N) **Ptr[N]**
- array_index(Ptr, N, Val) **Ptr[N] = Val;**
- Zero based indexing!

```
20> P = eqc_c:create_array(int, [0,1,2,3,4]).
{ptr,int,2098512}
21> eqc_c:array_index(P, 3).
3
22> eqc_c:array_index(P, 4, 10).
ok
23> eqc_c:read_array(P, 5).
[0,1,2,3,10]
```


Strings

- C strings are null terminated arrays of characters
- create_string/1, read_string/1

```
24> P = eqc_c:create_string("Hello World!").
{ptr,char,2099536}
25> eqc_c:read_string(P).
"Hello World!"
26> eqc_c:read_array(P, 13).
[72,101,108,108,111,32,87,111,114,108,100,33,0]
```

Erlang has no string data type, instead strings are lists of integers

Erlang and strings

- It's easy to crash a C program
- What happens when we do?

C exceptions are turned into Erlang exceptions

```
27> Null = {ptr, int, 0}.
{ptr,int,0}
28> eqc_c:deref(Null).
** exception error: segmentation_fault
```

- The C program is automatically restarted
  - but any internal state of the program is lost

When things go wrong

- T ::= int | char | unsigned_char | short | float | double | {ptr, P} | {struct, Atom} | D
- P ::= T | void | {func, T, [T]}
  - Types that can be pointed to by pointers
- D = string()
  - types defined by typedefs
- Not yet implemented
  - long int/double
  - unsigned short/int

The name of the struct

Supported C types

- Type definitions are expanded automatically

```
def.c

typedef int MyInt;
```

```
41> eqc_c:alloc("MyInt", 3).
{ptr,int,2097648}
```

Type definitions

- An Erlang record is generated for each C struct

struct.c

```
struct point {
  int x, y;
};
```

```
41> eqc_c:start(struct).
ok
```

generates

struct.hrl

```
-record(point, {x, y}).
```

Structs

Records are tuples with the record name as its first element

`-record(point,{x,y}).`
defines a record named point with fields x and y

```
P = #point{x=1,y=0} create record
P#point.x           return value of x field
P#point{y=1}        change value of y
```

internal representation
`{point,1,1}`

Erlang's record data type

struct.c

```
struct point {
  int x, y;
};

struct point pt(int x, int y) {
  struct point p = {x, y};
  return p;
}
```

```
42> struct:pt(1, 2).
{point, 1, 2}
```

Structs

- eqc_c:start(ErlModule, CFileName, OptionList)
  - **definitions_only**
    - only generate code for defined functions
    - useful to avoid generating code for functions declared in included header files
  - **{include_functions, Funs}**
    - only generate code for the given functions
  - **{exclude_functions, Funs}**
    - don't generate code for the given functions
  - plus options to control the C compiler

Startup options

TRY it yourself

write convert in C and use the Erlang tests written before to test it.

Implement testable fibonacci in C with tests in Erlang (see next slide)

Test C with Erlang

Taken from a C course:
http://www2.its.strath.ac.uk/courses/c/

```
#include <stdio.h>
main()
{
  int fib[24];
  int i;
  fib[0] = 0;
  fib[1] = 1;
  for(i = 2; i < 24; i++)
      fib[i] = fib[i-1] + fib[i-2];
  for (i = 0; i < 24; i++)
      printf("%3d %6d\n", i, fib[i]);
}
```

Implement tests first and then implementation that is testable

Fibonacci in C

Recall: We wrote inversion functions for our tests!

These may be very useful in the actual product.

Move them from test to product. Makes testing simpler

```
test_kilos([]) ->
   true;
test_kilos([Pounds|Rest]) ->
   Pounds = convert:kilos_to_lbs(
                convert:lbs_to_kilos(Pounds)).
   test_kilos(Rest).
```

Improve product to simplify testing

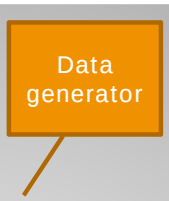
Instead of writing a number of tests, write a property that should hold for the program

Say what should hold for the program, let the computer generate tests automatically

Property Based Testing

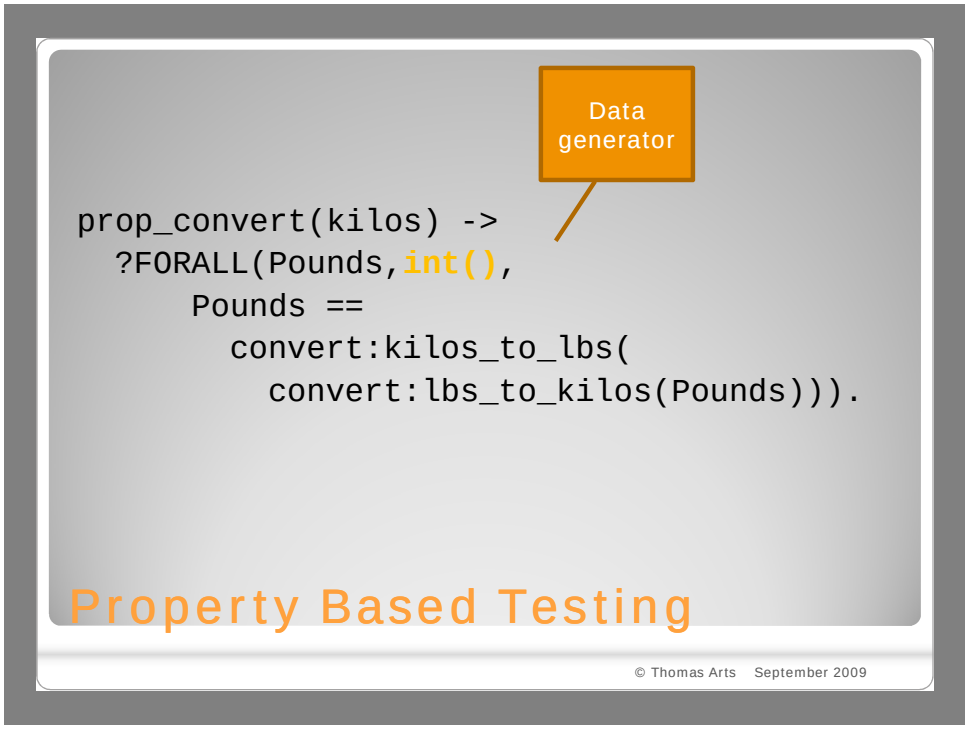
```
prop_convert(kilos) ->
  ?FORALL(Pounds,choose(10,250),
      Pounds ==
        convert:kilos_to_lbs(
          convert:lbs_to_kilos(Pounds))).
```

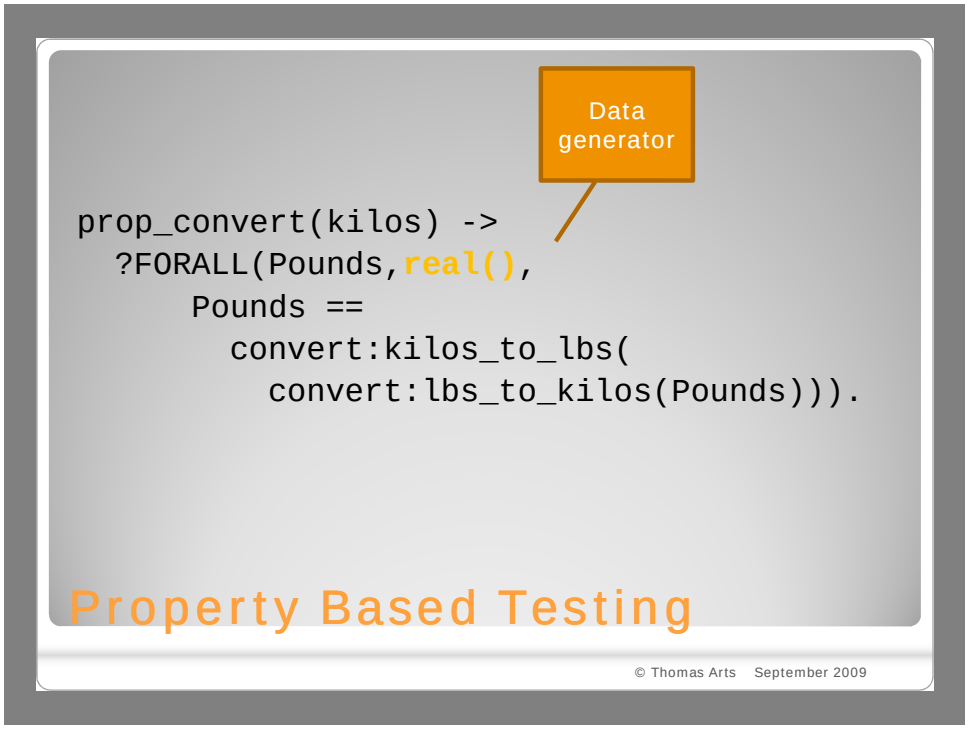
Property Based Testing




```
prop_convert(kilos) ->
  ?FORALL(Pounds,choose(10,250),
      Pounds ==
        convert:kilos_to_lbs(
          convert:lbs_to_kilos(Pounds))).
```

Property Based Testing


Data generator
prop_convert(kilos) ->
?FORALL(Pounds,int(),
Pounds ==
convert:kilos_to_lbs(
convert:lbs_to_kilos(Pounds))).
Property Based Testing
© Thomas Arts September 2009


Data generator
prop_convert(kilos) ->
?FORALL(Pounds,real(),
Pounds ==
convert:kilos_to_lbs(
convert:lbs_to_kilos(Pounds))).
Property Based Testing
© Thomas Arts September 2009


More tomorrow
© Thomas Arts September 2009