Property-Based Testing in Ada

The missing 10%

Fabien Chouteau

Embedded Software Engineer at AdaCore

Mastodon : @DesChips@mamot.fr
GitHub : Fabien-Chouteau
Hackaday.io: Fabien.C

Property-Based Testing

Demystifying monads in Rust through property-based testing

sunshowers.io/posts/monads-through-pbt/

Property-Based Testing

PBT starts with a developer deciding on a formal specification that they want their code to statisfy and encoding that specification as an executable property.

An automated test harness checks the property against their code using hundreds or thousands of random inputs, produced by a generator.

Property-Based Testing

The easy part:

> *PBT starts with a developer deciding on a formal specification that they want their code to statisfy and encoding that specification as an executable property.*

Property-Based Testing

```
subtype Some_Type is Float range 0.0 .. 1.0;

function Some_Function (A : Some_Type; B : Boolean)
                        return Positive
  with Pre  => (if B then A > 0.5 else A < 0.2),
       Post => Some_Function'Result > Positive (A) * 2;
```

Property-Based Testing

The missing part:

> *An automated test harness checks the property against their code using hundreds or thousands of random inputs, produced by a generator.*

GNATfuzz

- Repeatedly executes tests and generates new test cases at a very high frequency.
- Test cases that find a new path of execution are retained and will undergo further mutations.
- Explore deeper into the code base than other forms of random injection testing.

GNATtest Test Cases Generation

```
procedure Bubble_Sort (List : in out My_Array) is
begin
   for I in reverse List'First + 1 .. List'Last loop
      for J in List'First .. I - 1 loop
         if List (J) > List (J + 1) then
            declare
               Tmp : constant Integer := List (J);
            begin
               List (J) := List (J + 1);
               List (J + 1) := Tmp;
            end;
         end if;
      end loop;
   end loop;
end Bubble_Sort;
```

GNATtest Test Cases Generation

```
$ gnattest -Pprj.gpr  --gen-test-vectors --minimize
```

```
"test_vectors":[{
   "origin":"TGen",
   "Param_values":[{
      "name":"List","type_name":"sort.my_array","value":{
         "dimensions":[{"First":"6","Last":"12"}],
         "sizes":[" 7"],
         "array":["1602710117","-1730071287",...]
      }
}]
```

Strategy

Strategy

```
$ alr with strategy
```

Strategy

```
type My_Array is array (Natural range <>) of Unsigned_8;

package My_Array_VT
is new Indefinite_Value_Tree (My_Array);

package My_Array_Strat
is new Strategy.Arrays.Indefinite_Array_Strat
  (Unsigned_8, Natural, My_Array,
   My_Array_VT,
   Unsigned_8_Strat.Strat);
```

Strategy

```
procedure Sort
is new Ada.Containers.Generic_Array_Sort
  (Natural, Unsigned_8, My_Array);

function Check_Sorted (Arr : My_Array) return Boolean;
```

Strategy

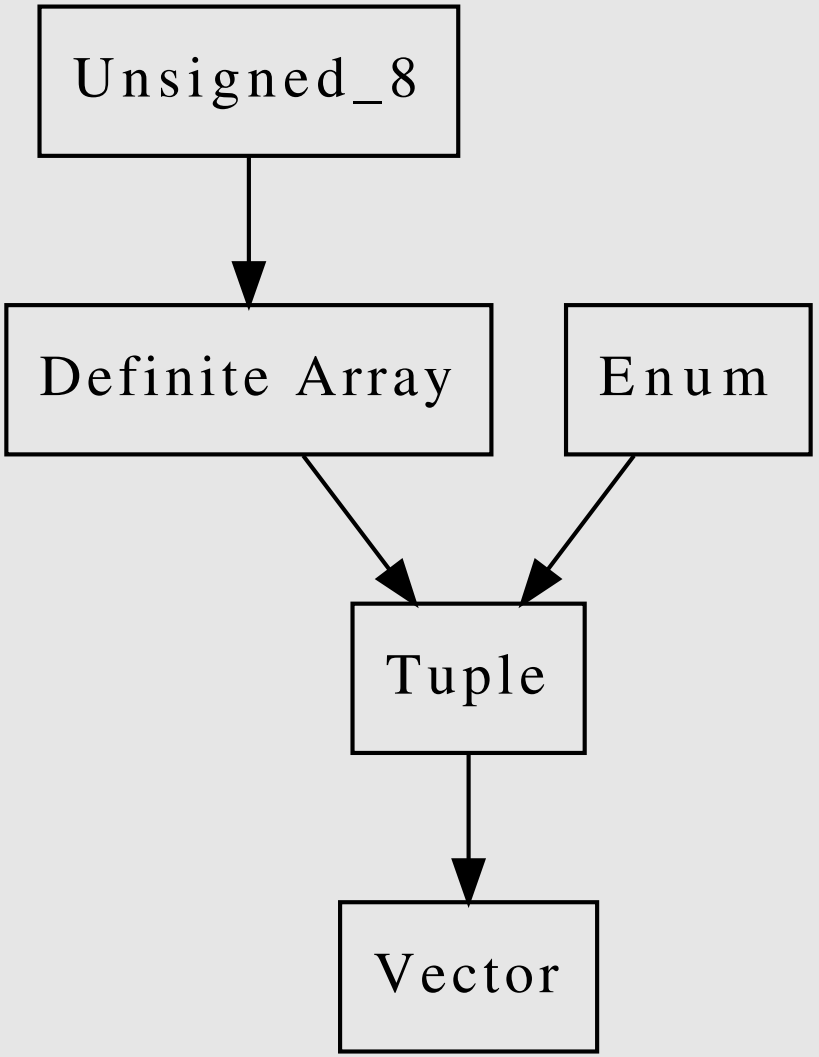
```
   package Sort_Runner
   is new Strategy.Runners.Indefinite_Runner
     (My_Array_Strat.Strat, Test_Sort);

   Result : Sort_Runner.Run_Result := Sort_Runner.Run;

begin
   if Result.Outcome = Fail then
      Put_Line (Result.Input.all'Img);
      Sort_Runner.Free (Result.Input);
   end if;
```

```
[0, 0, 0, 0, 0, 0, 23, 0, 0]
```

Infinite Possibilities



Strategy

```
type Node is interface;

function Simplify (This : in out Node) return Boolean
is abstract;

function Complicate (This : in out Node) return Boolean
is abstract;
```

Generics!

```
generic
   type Value is limited private;
package Definite_Value_Tree is

   type Value_Node is interface and Node;

   function Current (This : in out Value_Node) return Value
   is abstract;
end Definite_Value_Tree;
```

Generics!!

```
generic
   type Value is limited private;

   with package Value_Tree is new Definite_Value_Tree (Value);

   type Node (<>) is new Value_Tree.Value_Node with private;

   with function Create (Ctx : in out Runner_Context'Class)
                         return Node;

package Definite_Strategy is
   --  "Traits" like generic package that defines a value
   --  generation strategy for a definite type.
end Definite_Strategy;
```

Let's Create a Strategy

Generics!!!

```
generic
   type Value is mod <>;
   with package Value_Tree is new Definite_Value_Tree (Value);
package Modular_Strat is
   package Impl is
      type Node (<>) is new Value_Tree.Value_Node with private;
      function Create (Ctx : in out Runner_Context'Class)
                       return Node;
   private
      function Current (This : in out Node) return Value;
      function Simplify (This : in out Node) return Boolean;
      function Complicate (This : in out Node) return Boolean;
   end Impl;
   package Strat is new Definite_Strategy
     (Value, Value_Tree, Impl.Node, Impl.Create);
end Modular_Strat;
```

Generics!!!!

```
package Unsigned_8_Value_Tree
is new Definite_Value_Tree (Unsigned_8);

package Unsigned_8_Strat
is new Modular_Strat (Unsigned_8, Unsigned_8_Value_Tree);

package T2
is new Tuples.Tuple_2 (Unsigned_8_Strat.Strat,
                       Unsigned_8_Strat.Strat);
```

```
generic
   with package A_Strat is new Definite_Strategy (<>);
   with package B_Strat is new Definite_Strategy (<>);
package Tuple_2 is
...
```

Generics!!!!!

```
generic
   with package Input_Strat is new Definite_Strategy (<>);

   with function Test (Input : Input_Strat.Value) return Boolean;

package Definite_Runner is
   type Run_Result (Outcome : Result_Kind := Pass) is record ...

   function Run_One (Seed : Integer) return Run_Result;
   function Run (Runs : Positive := 1_000) return Run_Result;
end Definite_Runner;
```

Generics!!!!!!

```
   function Check_My_Function (Input : T2.Value) return Boolean;

   package Runner
   is new Strategy.Runners.Definite_Runner
     (T2.Strat, Check_My_Function);

   Result : Runner.Run_Result := Runner.Run;
begin
   if Result.Outcome = Runner.Fail then
      Put_Line (Result.Input'Img);
   end if;
```

Thanks