OCaml Companion Tools

OCaml Users & Developers 2012
Copenhagen - 14 september 2012
Xavier Clerc - forum@x9c.fr
Whatever your process is...

- Requirements
- Design
- Implementation
- Verification
- Maintenance

- Design
- Deployment
- Development
- Q/A
Anyway, you will need to...

- Browse APIs
- Write specifications and/or tests
- Understand program behaviour
- Ensure that test is thorough
- Ensure that code is readable
Argot

- Extended ocamldoc generator
- Adds tags and style
- Provides search facilities
  - name-based
  - type-based
  - fulltext-based
Kaputt

- Combinator-based test library
- Tests can be stored in .mlt files
- Provides support for various kinds of tests
  - assertion (xUnit-like)
  - enumeration (SmallCheck-like)
  - specification (QuickCheck-like)
Kaputt

OCaml version 4.00.0

# #directory "+kaputt";;
# #load "kaputt.cma";;
# open Kaputt.Abbreviations;;
# let gen_list = Gen.list (Gen.make_int 0 10) Gen.int;;
val gen_list : int list Kaputt.Generator.t = (<fun>, <fun>)
# let rec is_sorted = function
   | fst :: snd :: tl -> (fst <= snd) && is_sorted (snd :: tl)
   | _ -> true;;
val is_sorted : 'a list -> bool = <fun>
# check gen_list (List.sort compare) [Spec.always ==> is_sorted];;
Test 'untitled no 1' ... 100/100 cases passed
- : unit = ()
# let () =

Test.add_simple_test

(fun () ->
  let eq_int_list = Assert.make_equal_list (=) string_of_int in
  let f = Mock.from_function succ in
  let i = [0; 1; 2; 0] in
  let o = List.map (Mock.func f) i in
  let o' = [1; 2; 3; 1] in
  eq_int_list o' o;
  eq_int_list i (Mock.calls f);
  Assert.equal_int 4 (Mock.total f));;
Bolt

- both camlp4- and API-based log library
- Allows to configure how/where to log at runtime
- Layouts for various formats
  - from bare text to XML
  - Pajé (for multi-threads)
  - Daikon (for invariant computation)
let cat filename =
   LOG "printing file %s" filename LEVEL TRACE;
let channel = open_in filename in
try
   while true do
      let line = input_line channel in
      print_endline line
   done
with _ -> ()

let () =
   LOG "application start" LEVEL TRACE;
let len = Array.length Sys.argv in
for i = 1 to pred len do
   Aux.cat Sys.argv.(i)
done;
LOG "application end" LEVEL TRACE

logger "" {
   level = trace;
   filter = all;
   layout = simple;
   output = file;
   name = "log";
}

logger "" {
   level = fatal;
   filter = all;
   layout = html;
   output = file;
   name = "log-important";
}

logger "Main" {}
   level = trace;
   filter = all;
   layout = default;
   output = file;
   name = "log-main";

logger "Aux" {}
   level = trace;
   filter = all;
   layout = default;
   output = file;
   name = "log-aux";
Bisect

- camlp4-based code instrumenter
- Allows to tell whether an expression has been evaluated
- Reports in various formats
  - bare text (statistics)
  - HTML (code replica)
  - XML (EMMA-compatible)
Statistics:

<table>
<thead>
<tr>
<th>kind</th>
<th>coverage</th>
<th>kind</th>
<th>coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>binding</td>
<td>2 / 2 (100%)</td>
<td>class expression</td>
<td>0 / 0 (%)</td>
</tr>
<tr>
<td>sequence</td>
<td>2 / 2 (100%)</td>
<td>class initializer</td>
<td>0 / 0 (%)</td>
</tr>
<tr>
<td>for</td>
<td>0 / 0 (%)</td>
<td>class method</td>
<td>0 / 0 (%)</td>
</tr>
<tr>
<td>if/then</td>
<td>0 / 0 (%)</td>
<td>class value</td>
<td>0 / 0 (%)</td>
</tr>
<tr>
<td>try</td>
<td>1 / 1 (100%)</td>
<td>toplevel expression</td>
<td>0 / 0 (%)</td>
</tr>
<tr>
<td>while</td>
<td>1 / 1 (100%)</td>
<td>lazy operator</td>
<td>0 / 0 (%)</td>
</tr>
<tr>
<td>match/function</td>
<td>1 / 1 (100%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source:

```
let cat filename =

(*[1]*) try
  (*[1]*) let channel = open_in filename in
  (*[1]*)
  (*[1]*) while true do
  (*[1]*)   (*[1]*) let line = input_line channel in
  (*[1]*)   (*[1]*) print_endline line
  (*[1]*)
  (*[1]*) done
(*[1]*) with _ -> (*[1]*)()
```

Legend:
- `one code` - line containing no point
- `one code` - line containing only visited points
- `one code` - line containing only unvisited points
- `one code` - line containing both visited and unvisited points
Mascot

- Customizable style-checker (configuration, plugins)
- Complementary to compiler warnings
- Wide range of checks
  - typography, documentation
  - code smells, interface smells
  - metrics, etc.
```plaintext
category typography {
  line_length = { maximum = 50; };
  tab_character = true;
  trailing_white_space = true;
}

category code {
  empty_for = true;
  ignore_unit = true;
  no_effect_assignment = true;
}

let iter : ('a -> unit) -> 'a array 
  for i = 0 to Array.length a - 1 do 
  done

let print : ('a -> string) -> 'a array 
  let i = ref 0 in 
  while !i < Array.length a do 
    print_endline (m.(i));
    i := i + 1
  done
```

Summary: 0 warning, 6 errors, and 0 info

<table>
<thead>
<tr>
<th>src/style.ml</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>typography</td>
<td>line_length</td>
</tr>
<tr>
<td>code</td>
<td>empty_for</td>
</tr>
<tr>
<td>typography</td>
<td>tab_character</td>
</tr>
<tr>
<td>typography</td>
<td>line_length</td>
</tr>
<tr>
<td>typography</td>
<td>trailing_white_space</td>
</tr>
<tr>
<td>code</td>
<td>no_effect_assignment</td>
</tr>
</tbody>
</table>

- line 1, column 56, error: line is too long (56 instead of 50)
- line 2, column 2, error: empty 'for' loop
- line 3, column 0, error: tab character
- line 6, column 59, error: line is too long (59 instead of 50)
- line 9, column 29, error: trailing white space
- line 10, column 4, error: assignment with no effect
Using ocamlbuild

open Ocamlbuild_plugin
open Ocamlbuild_pack

let rec copy_mlt_files path =
    let elements = Pathname.readdir path in
    Array.iter (fun p ->
        if Pathname.is_directory (path / p) then
            copy_mlt_files (path / p)
        else if Pathname.check_extension p "mlt" then
            let src = path / p in
            let dst = !Options.build_dir / path / p in
            Shell.mkdir_p (!Options.build_dir / path);
            Pathname.copy src dst
        else
            ()
    ) elements

let getenv name default =
    String.uppercase
    (try
        Sys.getenv name
    with _ -> default)

let lib_dir = "/path/to/lib/ocaml"

let test_code = getenv "TEST" "OFF"
let log_level = getenv "LOG_LEVEL" "NONE"
let coverage = getenv "COVERAGE" "OFF"

let () =
    dispatch begin function
        | After_rules ->
            copy_mlt_files "src";
            flag ["ocaml_tools"; "pp"]
                (S [A"kaputt_pp.byte";"
                    (match test_code with "ON" -> A"on" | _ -> A"off")
                    A"camlp4o"; A"str.cma";
                    (match coverage with
                        | "ON" -> A(lib_dir ^ "/bisect/bisect_pp.cmo")
                        | _ -> N);
                        A(lib_dir ^ "/bolt/bolt_pp.cmo";
                        A"-level"; (A log_level))]);
            flag ["ocaml_tools"; "compile"]
                (S [A"-I"; A(lib_dir ^ "/bolt");
                    A"-I"; A(lib_dir ^ "/bisect");
                    A"-I"; A(lib_dir ^ "/kaputt")]);
            flag ["ocaml_tools"; "link"; "byte"]
                (S [A"-I"; A(lib_dir ^ "/bolt");
                    A"-I"; A(lib_dir ^ "/bisect");
                    A"-I"; A(lib_dir ^ "/kaputt")])
        | _ -> ()
    end
Using `ocamlbuild`

- Tag files with «`ocaml_tools`»
- Use environment variables to select features
  - `TEST=on` `LOG_LEVEL=trace` `COVERAGE=on` `ocamlbuild (...)`
  - `BOLT_CONFIG=config` `BISECT_FILE=coverage prog`
Using Jenkins

- Bisect has EMMA-compatible output
- Kaputt has jUnit-compatible output
- Mascot has CheckStyle-compatible output
- Easy to plug into Jenkins
- Jenkins will take care of project history
Checkstyle Trend

Tendance des résultats des tests

Code Coverage Trend

Emma Coverage Report

Overall Coverage Summary

Coverage Breakdown by Package

Résultats des tests

Tous les tests qui ont échoué

Nom du test

>>> Kaputt Report.random test

Tous les tests

Package
(root)
Thanks!
Questions?