Generating tests from formal specification

The good, the bad and the ugly

Nicolas Osborne

ANR grant ANR-22-CE48-0013

Introduction

What is the talk about?

- Dynamic formal verification, or specification-driven testing
- Meta-programming (programs that write programs, a lot of fun)

Contents

Quick presentation of QCheck-STM

What is QCheck-STM?

QCheck-STM [1], [2] is a model-based test framework in the OCaml ecosystem.

Generates and runs programs, then compares the collected results with running the same program on a model.

Works very well with mutable data structures.

Can test a library for parallel use (data race free).

Different kinds of testing

Example-based testing

$$\text{sort}\,[1;3;2] = [1;2;3]$$

The user provides **facts.**

Different kinds of testing

Property-based testing

$$\forall\ \text{xs} : \text{int list}.\ \text{is\_permutation xs (sort xs)} \wedge \text{is\_sorted (sort xs)}$$

The user provides **generators** and **properties.**

QCheck-STM testing

$$\forall\ \text{pg}.\ \text{behaviour pg} \equiv \text{behaviour (model pg)}$$

The user provides (amongst other things):

- the type of the data structure to test (SUT) and the type of its model (state)
- a function to initialize the SUT and the corresponding initial state
- a state machine transition function for the model
- a command generator
- pre and postconditions (relating calls to model)

How does Ortac/QCheck-STM work?

Ortac in the Gospel ecosystem

Ortac [3] tool is part of the Gospel ecosystem.

- Gospel: General OCaml SPEcification Language, a tool agnostic, contract and model-based behavioural specification language [4], [5]
- Ortac/Core: Translate the computable subset of Gospel terms into OCaml expressions
- Ortac/QCheck-STM: Glue the OCaml translations into QCheck-STM tests
- Other plugins for Ortac are developped [6], [7]

Model for type declaration

Interface file

```
type 'a t
(*@ mutable model contents: 'a sequence *)

val create : unit -> 'a t
(*@ t = create ()
    ensures t.contents = Sequence.empty *)
```

Model for type declaration

Configuration file

```
type sut = int t
let init_sut = create ()
```

Model for type declaration

Generated code

```
type sut = int t
let init_sut () = create ()
type state = { contents: int Ortac_runtime.Gospelstdlib.sequence }
let init_state = { contents = Ortac_runtime.Gospelstdlib.Sequence.empty }
```

Contract for a function

Interface file

```
val push: 'a -> 'a t -> unit
(*@ push x t
    modifies t.contents
    ensures t.contents = Sequence.cons x (old t.contents) *)
```

Contract for a function

Generated code

```
let next_state cmd state =
  match cmd with
   | Push a -> { contents = Ortac_runtime.Gospelstdlib.Sequence.cons a
state.contents }
  ...
```

Contract for a function with postconditions

Interface file

```
val pop_opt: 'a t -> 'a option
(*@ o = pop_opt t
    modifies t.contents
    ensures t.contents = if (old t.contents) = Sequence.empty
        then Sequence.empty
        else Sequence.tl (old t.contents)
    ensures o = if (old t.contents) = Sequence.empty
        then None
        else Some (Sequence.hd (old t.contents))
*)
```

Contract for a function with postconditions

Generated code

```
let postcond cmd state res =
  match (cmd, res) with
  | Pop_opt, Res ((Option Char, _), o) ->
    o = if state.contents = Ortac_runtime.Gospelstdlib.Sequence.empty
        then None
        else Some (Ortac_runtime.Gospelstdlib.Sequence.hd state.contents)
  ...
```

Command generator

Generated code

```
type cmd =
  | Create of unit
  | Push of int
  | Pop_opt

let gen_cmd _ =
  let open QCheck in
  oneof_weighted
    [(1, ((pure (fun () -> Create ())) <*> unit));
     (1, ((pure (fun v -> Push v)) <*> int));
     (1, (pure Pop_opt))]
```

Niceties, limitations and workaround

Some niceties

- Add a bug report based on the information from the Gospel annotations and a runnable scenario to reproduce the failure
- Handle natively test involving multiple SUTs [8]
- Test with Domains soon to be released

Bug discovery

- A missing check of index out of bound in the `varray` library
- An undefined behaviour still in the `varray` library
- Some integer overflows in the `bitv` library
- Some inconsistent behavior with zero-length vectors in the `bitv` library
- A peculiar behaviour in `Hashtbl.create` from the standard library

Limitations and workaround

- the tool consumes only specifications written in a certain style
- assumptions on higher-order functions are not taken into account
- default command generator may not be fit, configuration file allows to:
  1. add frequencies to fine tune the command generator
  2. override default argument generator or provide one for custom types
  3. completely provide the command generator to take advantage of it being parameterized over the state

References

[1] Jan Midtgaard, Olivier Nicole, and Nicolas Osborne, “Multicoretests - Parallel Testing Libraries for OCaml 5.0,” OCaml Users and Developers Workshop (2022). [Online]. Available: https://janmidtgaard.dk/papers/Midtgaard-Nicole-Osborne:OCaml22.pdf

[2] Jan Midtgaard, “Property-Based Testing of OCaml 5’s Runtime System, Fun and Segfaults with Interpreters and State Transition Functions,” Proceedings of the Workshop Dedicated to Olivier Danvy on the Occasion of His 64th Birthday (OLIVIERFEST ’25). [Online]. Available: https://janmidtgaard.dk/papers/Midtgaard%3aOLIVIERFEST25.pdf

[3] Ortac repository. [Online]. Available: https://github.com/ocaml-gospel/ortac

[4] Gospel repository. [Online]. Available: https://github.com/ocaml-gospel/gospel

[5] Arthur Charguéraud, Jean-Christophe Filliâtre, Cláudio Lourenço, and Mário Pereira, “GOSPEL - Providing OCaml with a Formal Specification Language,” FM 2019 - 23rd International Symposium on Formal Methods (2019). [Online]. Available: https://doi.org/10.1007/978-3-030-30942-8_29

[6] Clément Pascutto, “Runtime verification of OCaml programs.” [Online]. Available: https://theses.hal.science/tel-04696708

[7] Nicolas Osborne and Clément Pascutto, “Leveraging Formal Specifications to Generate Fuzzing Suites,” OCaml Users and Developers Workshop (2021). [Online]. Available: https://inria.hal.science/hal-03328646

[8] Nikolaus Huber, Naomi Spargo, Nicolas Osborne, Samuel Hym, and Jan Midtgaard, “Dynamic Verification of OCaml Software with Gospel and Ortac/QCheck-STM,” 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2025). [Online]. Available: https://hal.science/hal-05073121

Thank you!