QuickerChick

3–7× faster property-based random testing in Rocq

Ivan Mladenov, Alperen Keles, Leonidas Lampropoulos

ICFP SRC 2026

A tale of extraction

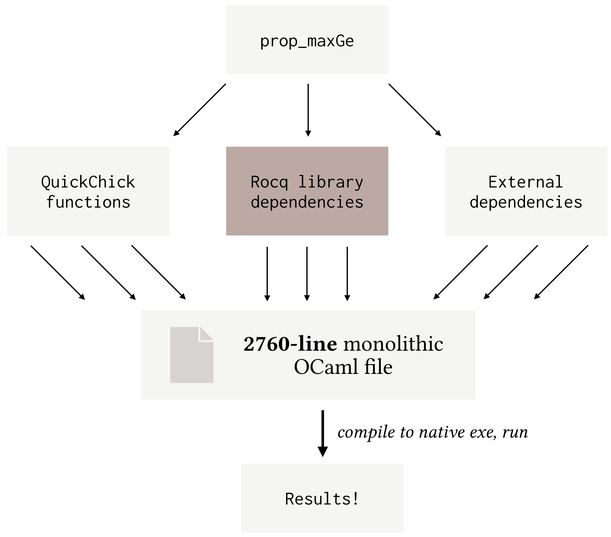
Imagine you're in Proof General or Coqtail and want to test the following property:

```
Definition max (x : nat) (y : nat) : nat :=
  if x <=? y then y else x.
Conjecture prop_maxGe:
  forall (x y : nat), max x y >= y /\ max x y >= x.
QuickChick prop_maxGe.
```

When you run it, you'd see:

```
> QuickChecking prop_maxGe
> +++ Passed 10000 tests (0 discards)
```

What you don't see: QuickChick uses *program extraction* to translate Rocq to OCaml, which allows for the side effects (randomness/IO) that PBT needs. For extraction to work, it must also extract dependencies:



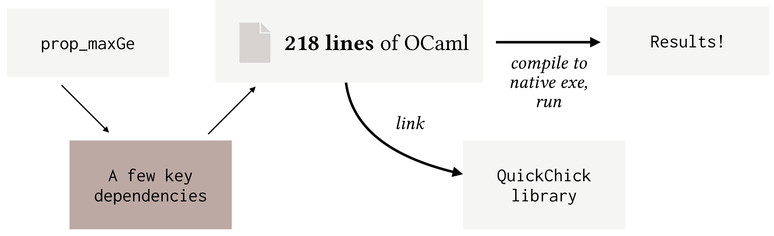
Every time you step through the **`QuickChick`** command, this process is repeated even though core QuickChick doesn't change across runs. This redundancy has significant performance implications.

QuickChick library à la OCaml

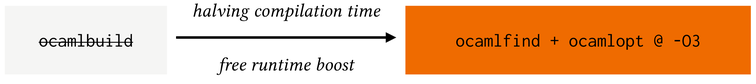
Core QuickChick already has manual extraction directives, so we can extend them to map QuickChick dependencies and Rocq definitions to efficient OCaml equivalents. Here's an example of manual extraction:

```
Extract Inductive State.Map.bst => "Qc__Coq_Map.bst"
[ "Qc__Coq_Map.Leaf" "Qc__Coq_Map.Node" ].
Extract Constant State.Map.t "'elt" => "'elt Qc__Coq_Map.t".
Extract Inlined Constant State.Map.empty => "Qc__Coq_Map.empty".
```

We extended the library to include common type definitions and functions from QuickChick (test generation, randomness) and Rocq libraries (strings, natural numbers). Now, the example from before:



That's over a **90% reduction** in generated OCaml. But now we can also leverage the benefits of the OCaml ecosystem to tune the library. By leveraging better build commands and optimization levels:



We rewrote poorly extracted OCaml and sped up test size generation. For example, we rewrote this bottleneck to be tail-recursive:

```
let rec combine l l' =
  match l with
  | [] -> []
  | x :: tl -> ( match l'
with [] -> [] | y :: tl' ->
(x, y) :: combine tl tl' )
```

```
let combine l l' =
  let rec aux l l' acc =
    match (l, l') with
    | [], _ | _, [] -> List.rev acc
    | x :: tl, y :: tl' -> aux tl
tl' ((x, y) :: acc)
  in
  aux l l' []
```

Measuring the speedup

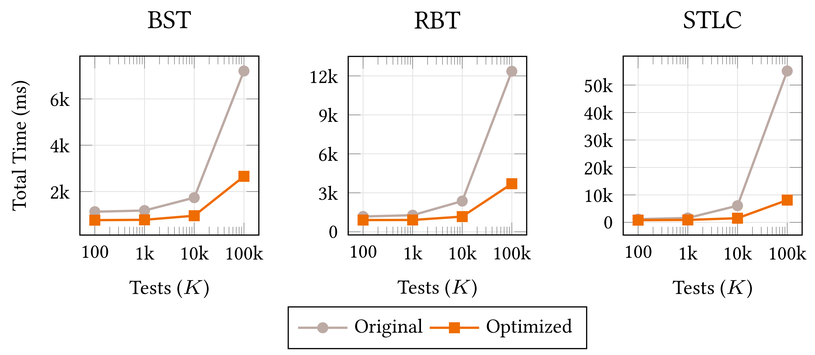
We evaluated our progress in *ETNA*, a benchmarking platform for PBT. In a nutshell, we saw:

1.5–3× speedup in extraction

>2× speedup in compilation

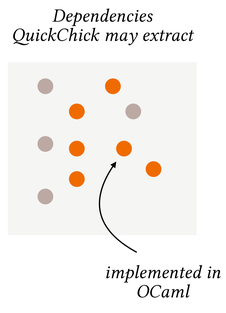
3–7× speedup in testing

On smaller test runs, faster extraction and compilation makes testing feel much more interactive. In your favorite Rocq IDE, lag is noticeably reduced and it feels snappy. For larger workloads, the improvements are impossible to overlook:



PBT Workloads from ETNA

The STLC benchmark alone shows how we can take testing from a minute down to under 10 seconds. The upshot: our changes do not affect how users interact with QuickChick, yet everyone can reap the benefits.

Conclusion and future work

Ideally, we'd have a 1–1 equivalent of all QuickChick library functions hand-optimized and compiled in OCaml. Because this library is extensible, we can keep translating Rocq functions to approach this goal. Ultimately, this extracted library lays the groundwork for a better-performing QuickChick, where the gap between testing verified programs and fast execution is narrowing.