Zoo: A framework for the verification of concurrent OCaml 5 programs using separation logic

ANONYMOUS AUTHOR(S)

The release of OCaml 5, which introduced parallelism into the language, drove the need for safe and efficient concurrent data structures. New libraries like `Saturn` aim at addressing this need. From the perspective of formal verification, this is an opportunity to apply and further state-of-the-art techniques to provide stronger guarantees.

We present Zoo, a framework for verifying fine-grained concurrent OCaml 5 algorithms. Following a pragmatic approach, we defined a limited but sufficient fragment of the language to faithfully express these algorithms: ZooLang. We formalized its semantics carefully via a deep embedding in the Rocq proof assistant, uncovering subtle aspects of physical equality. We provide a tool to translate source OCaml programs into ZooLang syntax inside Rocq, where they can be specified and verified using the Iris concurrent separation logic. To illustrate the applicability of Zoo, we verified a subset of the standard library and a collection of fined-grained concurrent data structures from the `Saturn` and `Eio` libraries.

In the process, we also extended OCaml to more efficiently express certain concurrent programs.

1 Introduction

OCaml 5.0 was released on December 15th 2022, the first version of the OCaml programming language to support parallel execution of OCaml threads by merging the Multicore OCaml runtime [Sivaramakrishnan et al. 2020]. It provided basic support in the language runtime to start and stop coarse-grained threads ("domains" in OCaml parlance) and support for strongly sequential atomic references in the standard library. The third-party library `domainslib` offered a simple scheduler for a pool of tasks, used to benchmark the parallel runtime. A world of parallel and concurrent software was waiting to be invented.

Shared-memory concurrency is a difficult programming domain, and existing ecosystems (C++, Java, Haskell, Rust, Go...) took decades to evolve comprehensive libraries of concurrent abstractions and data structures. In the last couple years, a handful of contributors to the OCaml system have been implementing basic libraries for concurrent and parallel programs in OCaml, in particular `Saturn` [Karvonen and Morel 2024], a library of lock-free thread-safe data structures (stacks, queues, a work-stealing dequeue, a skip list, a bag and a hash table), `Eio` [Madhavapeddy and Leonard 2024], a library of asynchronous IO and structured concurrency, and `Kcas` [Karvonen 2024], a library offering a software-transactional-memory abstraction for users to build safe yet efficient thread-safe data structures.

Concurrent algorithms and data structures are extremely difficult to reason about. Their implementations tend to be fairly short, a few dozens of lines. There is only a handful of experts able to write such code, and many potential users. They are difficult to test comprehensively. These characteristics make them ideally suited for mechanized program verification.

We embarked on a mission to mechanize correctness proofs of OCaml concurrent algorithms and data structures as they are being written, in contact with their authors, rather than years later. In the process, we not only gained confidence in these complex new building blocks, but we also improved the OCaml language and its verification ecosystem.

Verification tools for concurrent programs.

OCaml language features. When studying the new codebases of concurrent and parallel data structures, we found a variety of unsafe idioms, working around expressivity or performance limitations with the OCaml language support for lock-free concurrent data structures. In particular, the support for *atomic references* in the OCaml library proved inadequate, as idiomatic concurrent

data-structures need the more expressive feature of *atomic record fields*. We designed an extension of OCaml with atomic record fields, implemented it as a an experimental compiler variant, and succeeded in getting it integrated in the upstream OCaml compiler: it should be available as part of OCaml 5.4, which is not yet released at the time of writing.

Specified OCaml semantics.

References

Vesa Karvonen. 2024. Kcas. https://github.com/ocaml-multicore/kcas

Vesa Karvonen and Carine Morel. 2024. Saturn. https://github.com/ocaml-multicore/saturn

Anil Madhavapeddy and Thomas Leonard. 2024. Eio. https://github.com/ocaml-multicore/eio

KC Sivaramakrishnan, Stephen Dolan, Leo White, Sadiq Jaffer, Tom Kelly, Anmol Sahoo, Sudha Parimala, Atul Dhiman, and Anil Madhavapeddy. 2020. Retrofitting parallelism onto OCaml. *Proc. ACM Program. Lang.* 4, ICFP, Article 113 (Aug. 2020), 30 pages. doi:10.1145/3408995