SATURN: a library of verified lockfree data structures for OCAML 5

Clément Allain (INRIA)
Vesa Karvonen (Tarides)
Carine Morel (Tarides)

May 22, 2024

1 Presentation

We present SATURN, a new OCAML 5 library available on `opam`. SATURN offers a collection of concurrent lockfree data structures: stack, queue, skiplist, hash table, work-stealing deque, etc. In particular, the implementations of these data structures do not use any locking mechanism and guaranty system-wide progress. They are also faster than blocking implementations. The `Domainslib` library already uses SATURN's work-stealing deque.

Designing such data structures can be very tricky. Therefore, we aim at providing OCAML 5 programmers with a sound and efficient library saving them the trouble. SATURN is well tested, benchmarked and in part verified.

2 Library design

SATURN has been designed to meet the needs of programmers looking for lockfree implementations of standard data structures—typically to replace blocking implementations. From this perspective, it can be considered as a standard library for lockfree data structures. More precisely, SATURN currently features: TODO.

As any standard library, though, SATURN is not tailored to more demanding needs: specialized data structures fitting some specific contexts. However, experienced programmers looking for exotic variants may still find SATURN's standard algorithms useful as a basis to implement their own.

Regarding the performance of SATURN, there is often a tension between improving efficiency and remaining in the safe fragment of OCAML 5. In particular, due to the relatively poor compiler support for atomic accesses, it is tempting to bypass current limitations by using unsafe features of the language. Although these optimizations are performed with care and should not break anything for regular uses, memory safety is not guaranteed for illegal uses and the interaction with FLAMBDA is not clear. As a consequence, we propose two versions of SATURN: a safe version and an unsafe version. While most users should find the safe version efficient enough for their needs, daring users may prefer the unsafe version provided they encapsulate it correctly and verify their code somehow.

3 Verification

Lockfree algorithms are notoriously difficult to get right. To provide stronger guarantees, we have verified part of SATURN's data structures and hope to verify the entire library in the future. Moreover, one important benefit is that we get formal specifications for verified data structures.

This verification effort has been conducted using Iris [1], a state-of-the-art mechanized *concurrent separation logic*. In particular, all proofs are formalized in Coq.

A common criterion to specify concurrent operations on shared data structures is *linearizability*. The equivalent Iris notion is *logical atomicity*: there exists a point in time when a concurrent operation atomically takes effect (the linearization point). This statement takes the form of an *atomic specification*:

$$\begin{array}{c} \{\, \text{queue-inv}\ t \,\} \\ \hline \langle\, \forall vs.\, \text{queue-model}\ t\ vs \,\rangle \\ \hline \texttt{queue\_push}\ t\ v \\ \hline \langle\, \text{queue-model}\ t\ (vs +\!\!+ [v]) \,\rangle \\ \hline \{\, ().\, \text{True} \,\} \end{array}$$

In this example, we specify the `queue_push` operation from an implementation of a concurrent queue. Similarly to Hoare triples, the two assertions inside curly brackets express the precondition and postcondition. Here, the queue-inv t precondition represents the queue invariant. As it is persistent, we do not need to give it back in the postcondition. The other two assertions inside angle brackets express the *atomic precondition* and *atomic postcondition*. Basically, they specify the linearization point of the operation: during the execution of `queue_push`, the logical model of the queue is atomically updated from vs to $vs +\!\!+ [v]$, in other words v is atomically pushed at the back of the queue.

As a final note, we emphasize that our verification assumes a sequentially consistent memory model. Nevertheless, OCaml 5's relaxed memory model has been formalized [2] in Iris. It should be possible and is future work to adapt our specifications and proofs to support it.

4 Talk proposal

In our talk, we will introduce Saturn, including the main data structures and design guidelines. We will also discuss ongoing work on verifying the library using the Iris concurrent separation logic.

References

[1] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. "Iris from the ground up: A modular foundation for higher-order concurrent separation logic". In: *J. Funct. Program.* 28 (2018), e20. URL: `https://doi.org/10.1017/S0956796818000151`.

[2] Glen Mével, Jacques-Henri Jourdan, and François Pottier. "Cosmo: a concurrent separation logic for multicore OCaml". In: *Proc. ACM Program. Lang.* 4.ICFP (2020), 96:1–96:29. URL: `https://doi.org/10.1145/3408978`.