

# Saturn: a library of verified concurrent data structures for OCaml 5

---

Clément Allain (INRIA Paris)

Vesa Karvonen (Tarides)

Carine Morel (Tarides)

September 5, 2024







## Formal verification

*Testing* a concurrent algorithm is hard due to the number of potential interleavings.

*Formal verification* is here to help us!



## From OCaml to Coq

```
let rec push t v =  
  let old = Atomic.get t in  
  let new_ = v :: old in  
  if not (Atomic.compare_and_set t old new_) then (  
    Domain.cpu_relax () ;  
    push t v  
  )
```

OCAML

```
Definition stack_push : val :=  
  rec: "stack_push" "t" "v" =>  
    let: "old" := !"t" in  
    let: "new" := `Cons( "v", "old" ) in  
    ifnot: CAS "t" "old" "new" then (  
      Yield ;;  
      "stack_push" "t" "v"  
    ) .
```

Coq

$$\text{linearizability} \quad \underset{\text{in } \overline{\text{SC}}}{\approx} \quad \frac{\frac{\frac{\{ \text{stack-inv } t \}}{\langle \text{vs. stack-model } t \text{ vs } \rangle}}{\text{stack\_push } t \text{ } v}}{\langle \text{stack-model } t (v :: \text{vs}) \rangle}}{\{ () . \text{True} \}}$$

*Theorems for free from separation logic specifications*

Birkedal, Dinsdale-Young, Guéneau, Jaber, Svendsen & Tzevelekos

## Relaxed memory model (future work)

$$\frac{\frac{\frac{\{ \text{stack-inv } t \}}{\langle v_0, \dots, v_n, \mathcal{V}_0, \dots, \mathcal{V}_n. \text{stack-model } t ((v_0, \mathcal{V}_0), \dots (v_n, \mathcal{V}_n)) \rangle}}{\text{stack\_push } t \, v, \mathcal{V}}}{\frac{\langle \text{stack-model } t ((v, \mathcal{V}), (v_0, \mathcal{V}_0), \dots (v_n, \mathcal{V}_n)) \rangle}{\{ () . \text{True} \}}}$$

*Cosmo: a concurrent separation logic for multicore OCAML*

Mével, Jourdan & Pottier



## Writing concurrent protocols in Iris

```

Definition stack_inv t  $\iota$  : iProp  $\Sigma$  :=
   $\exists$  l  $\gamma$ ,
   $\ulcorner t = \#l \urcorner * \text{meta } l \text{ nroot } \gamma * \\
  \text{inv } \iota ( \\
    \exists \text{ vs}, l \mapsto \text{lst\_to\_val } \text{vs} * \text{stack\_model}_2 \gamma \text{ vs} \\
  )$ .

```

```

Lemma stack_push_spec t  $\iota$  v :
  <<< stack_inv t  $\iota$ 
  |  $\forall$  vs, stack_model t vs >>>
    stack_push t v @  $\uparrow \iota$ 
  <<< stack_model t (v :: vs)
  | RET (); True >>>

```

### Proof.

• • •

Qed.

Thank you for your attention!