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

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

September 5, 2024

Tarides

Plan

Intro

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

```ocaml
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

Coq

```
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"
    ).
```

Proving linearizability

linearizability $\underset{\text{in SC}}{\simeq}$

$$\frac{\frac{\frac{\frac{\{\,\text{stack-inv}\ t\,\}}{\langle\, vs.\,\text{stack-model}\ t\ vs\,\rangle}}{\texttt{stack\_push}\ t\ v}}{\langle\,\text{stack-model}\ t\ (v :: vs)\,\rangle}}{\{\ ().\,\text{True}\,\}}$$

Theorems for free from separation logic specifications

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

Relaxed memory model (future work)

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

Cosmo: a concurrent separation logic for multicore OCaml

Mével, Jourdan & Pottier

Writing concurrent protocols in Iris

```
Definition stack_inv t ι : iProp Σ :=
  ∃ l γ,
  ⌜t = #l⌝ ∗ meta l nroot γ ∗
  inv ι (
    ∃ vs, l ↦ lst_to_val vs ∗ stack_model₂ γ vs
  ).

Lemma stack_push_spec t ι v :
  <<< stack_inv t ι
  |   ∀∀ vs, stack_model t vs >>>
        stack_push t v @ ↑ι
  <<< stack_model t (v :: vs)
  |   RET (); True             >>>.
Proof.
  ...
Qed.
```

Thank you for your attention!