FreeSpec

Implementing and Certifying Impure Computations in Coq

Thomas Letan[1] Yann Régis-Gianas[2,3]

[1] French Cybersecurity Agency (ANSSI)
[2] Université de Paris, IRIF/PPS
[3] πr^2, Inria Paris-Rocquencourt

February 17, 2020

someone> Could you write a certified compiler for me?
coq-der> My pleasure!

someone> Could you write a certified compiler for me?
coq-der> My pleasure!

someone> Could you write a hello world program for me?
coq-der> No.
-!-   coq-der [~xl@166.37.73.42] has left #coq [Crying]

```
someone> Could you write a certified compiler for me?
coq-der> My pleasure!

someone> Could you write a hello world program for me?
coq-der> No.
-!-      coq-der [~xl@166.37.73.42] has left #coq [Crying]

yurug  > sure!
```

```
Exec do
  let* name := scan in
  echo ("hello " ++ name ++ "!\n")
end.
```

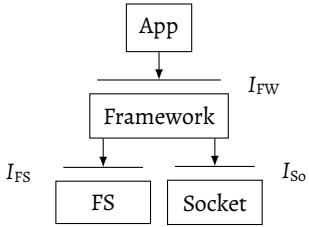
FreeSpec in a Nutshell

Past and Present

A general-purpose framework to **implement** (with an interfaces-indexed Free monad) and **certify** (with interface contracts) impure computations

In-the-large

Composition principles for interfaces, contracts and components



In-the-small

Proof-engineering and quality-life features for **(re)usability**

```
Exec do
  let* name := scan in
  echo ("hello " ++ name ++ "!\n")
end.
```

Our Long-Term Goal

Turning Coq into a Programming Language

Our Long-Term Goal

Turning Coq into a Programming Language

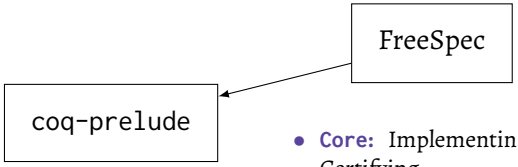
`coq-prelude`

An overlay for the Coq standard library

- `Equality` typeclass
- `Monad` typeclass
- `text` and `bytes`

Our Long-Term Goal

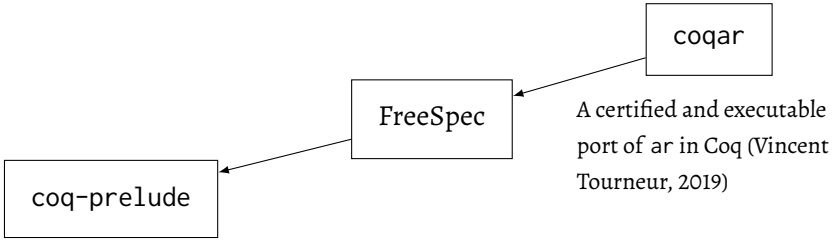
Turning Coq into a Programming Language



- **Core:** Implementing and Certifying
- **Exec:** Executing within Coq

Our Long-Term Goal

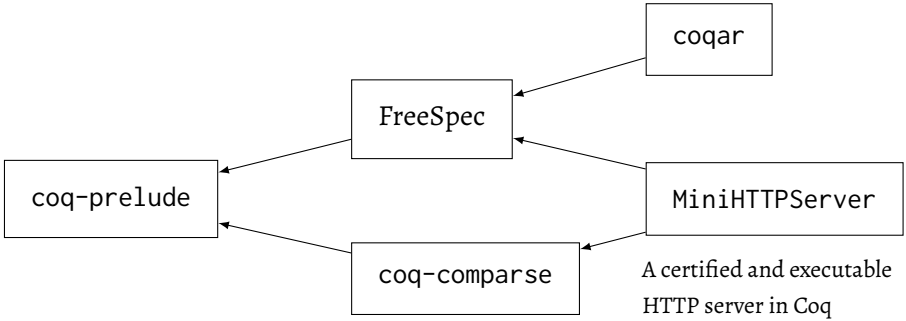
Turning Coq into a Programming Language



A certified and executable port of `ar` in Coq (Vincent Tourneur, 2019)

Our Long-Term Goal

Turning Coq into a Programming Language

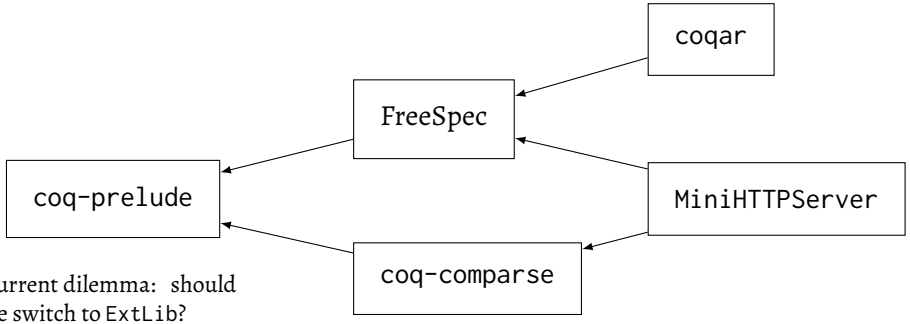


A certified and executable HTTP server in Coq (CPP'2020)

A general-purpose monadic parsing for Coq

Our Long-Term Goal

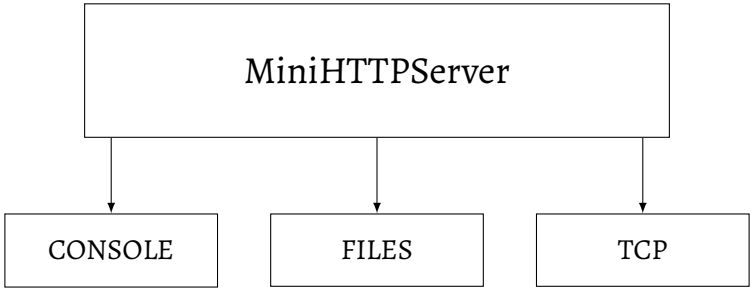
Turning Coq into a Programming Language



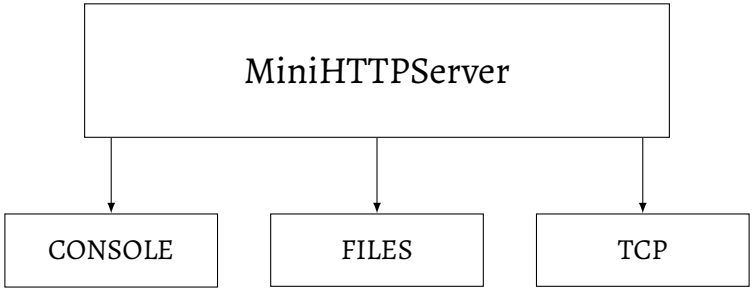
Current dilemma: should we switch to `ExtLib`?

`ANSSI-FR/FreeSpec#39`

Implementing and Certifying
The Example of MiniHTTPServer



Implementing and Certifying
The Example of MiniHTTPServer



FreeSpec shall allow for writing **idiomatic** monadic code

- Clear interface, opaque internals
- “Batteries included” standard library
- Straightforward execution

Implementing a HTTP Server

FreeSpec Cheat Sheet

```
interface := Type -> Type

impure : interface -> Type -> Type

Provide : interface -> interface -> Constraint

pure {ix a} : a -> impure ix a

request `{Provide ix i} {a} : i a -> impure ix a

do let* x := foo in
   bar x;
   foobar
end
```

Certifying a HTTP Server

Step 1. Specifying a Contract

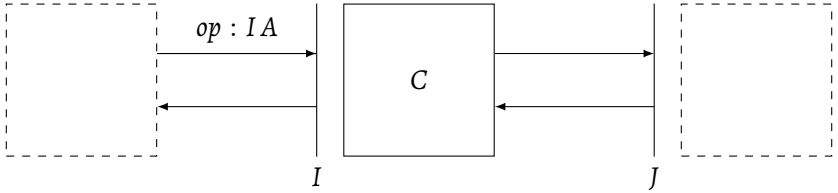
FreeSpec provides the following `contract` type

```
Record contract (i : interface) (W : Type) : Type := make_contract
  { witness_update (w : W) : forall (a : Type), i a -> a -> W
  ; caller_obligation (w : W) : forall (a : Type),  i a -> Prop
  ; callee_obligation (w : W) : forall (a : Type), i a -> a -> Prop
  }.
```

Defining a `contract` means specifying **how an interface has to be used** and **what to expect from it**.

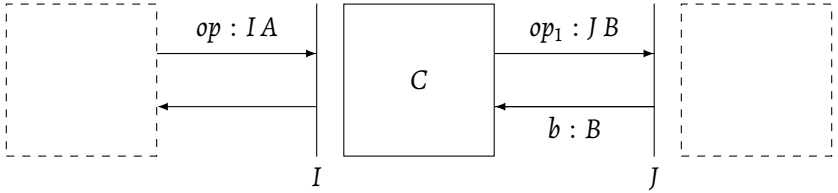
$\Rightarrow$ You can define as many contracts as you need.

Proving the Correct Use of an Interface



The component C receives a computational request op through I

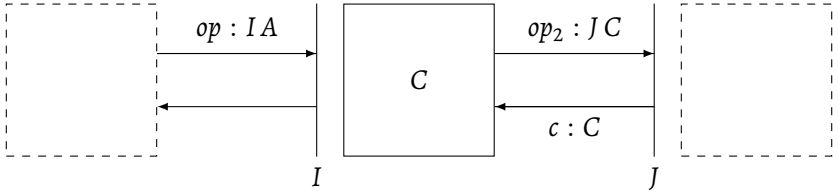
Proving the Correct Use of an Interface



The component C receives a computational request op through I

1. Sends a computational request op_1 to J
2. Waits for a result b

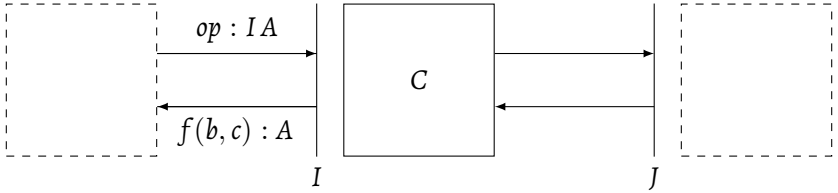
Proving the Correct Use of an Interface



The component C receives a computational request op through I

1. Sends a computational request op_1 to J
2. Waits for a result b
3. Sends a computational request op_2 to J
4. Waits for a result c

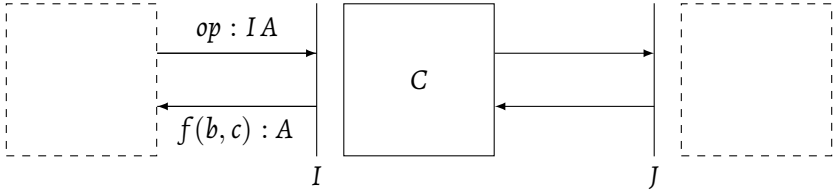
Proving the Correct Use of an Interface



The component C receives a computational request op through I

1. Sends a computational request op_1 to J
2. Waits for a result b
3. Sends a computational request op_2 to J
4. Waits for a result c
5. Computes the result of op

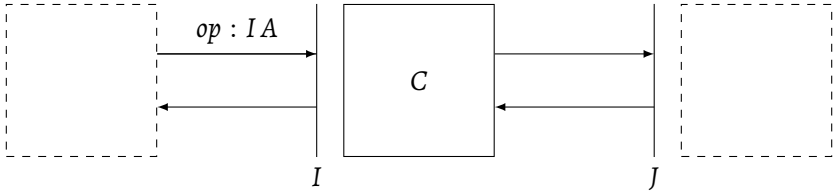
Proving the Correct Use of an Interface



The component C receives a computational request op through I

```
do let* b := request (op1) in
   let* c := request (op2) in
   pure (f b c)
end
```

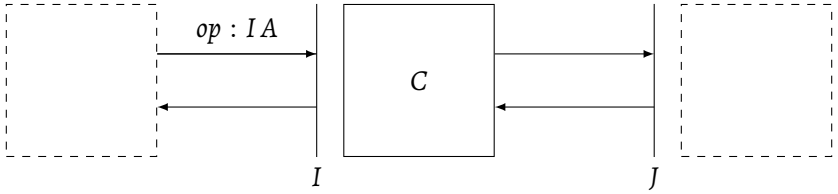
Proving the Correct Use of an Interface



Hypothesis: $\mathbb{P}_I(op)$
The component which uses I does it correctly

$$\frac{\mathbb{P}_I(op)}{\vdash}$$

Proving the Correct Use of an Interface

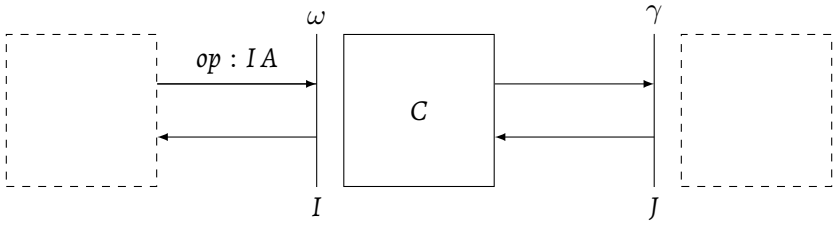


Hypothesis: $\mathbb{P}_I(op)$
The component which uses I does it correctly

Problem: Obligations may vary in time

$$\frac{\mathbb{P}_I(op, \omega)}{\vdash}$$

Proving the Correct Use of an Interface

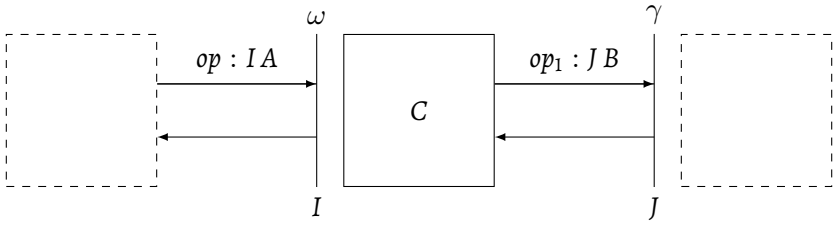


Solution: Parameterize obligations by an abstract state

Hypothesis: $\mathbb{P}_I(op, \omega)$

$$\frac{\mathbb{P}_I(op, \omega)}{\vdash}$$

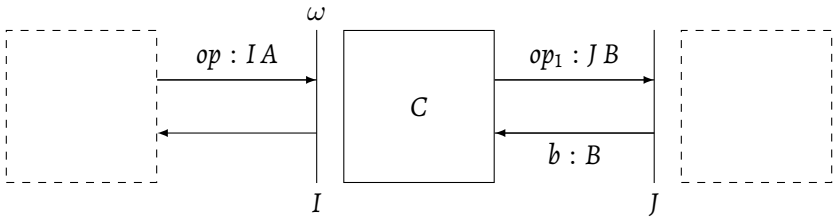
Proving the Correct Use of an Interface



Goal: $\mathbb{P}_J(op_1, \gamma)$
C shall correctly use J

$$\frac{\mathbb{P}_I(op, \omega)}{\vdash \mathbb{P}_J(op_1, \gamma)}$$

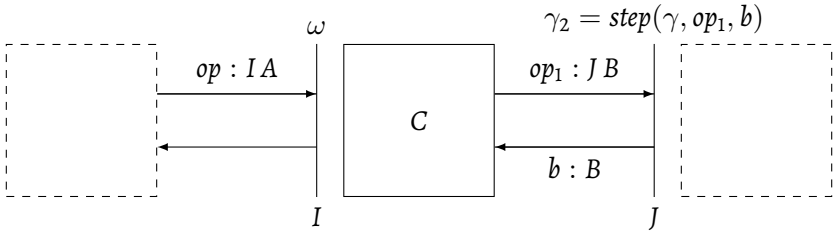
Proving the Correct Use of an Interface



Hypothesis: $\mathbb{P}_J(op_1, \gamma) \Rightarrow \mathbb{Q}_J(op_1, b, \gamma)$
The component which exposes J is correct

$$\frac{\mathbb{Q}_J(op_1, b, \gamma) \quad \mathbb{P}_J(op_1, \gamma) \quad \mathbb{P}_I(op, \omega)}{\vdash}$$

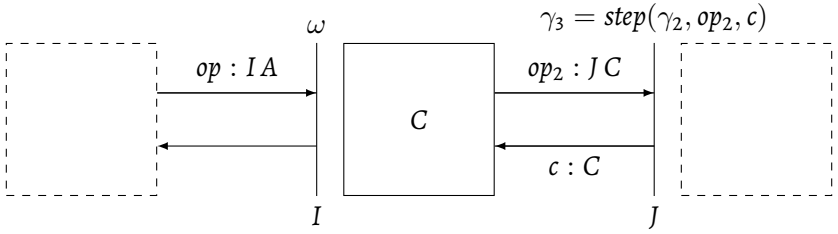
Proving the Correct Use of an Interface



Updating the abstract state attached to J updates the related pre and postcondition

$$\frac{\begin{array}{l}\gamma_2 = step(\gamma, op_1, b)\\ \mathbb{Q}_J(op_1, b, \gamma)\\ \mathbb{P}_J(op_1, \gamma)\\ \mathbb{P}_I(op, \omega)\end{array}}{\vdash}$$

Proving the Correct Use of an Interface

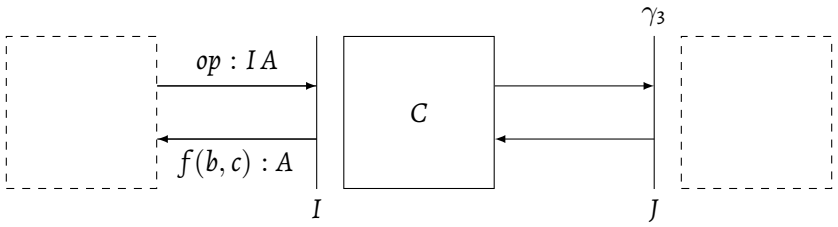


Goal: $\mathbb{P}_J(op_2, \gamma_2)$
C shall correctly use J

Hypothesis: $\mathbb{P}_J(op_2, \gamma_2) \Rightarrow \mathbb{Q}_J(op_2, c, \gamma_2)$
The component which exposes J is correct

$$\gamma_3 = step(\gamma_2, op_2, c)$$
$$\mathbb{Q}_J(op_2, c, \gamma_2)$$
$$\mathbb{P}_J(op_2, \gamma_2)$$
$$\gamma_2 = step(\gamma, op_1, b)$$
$$\mathbb{Q}_J(op_1, b, \gamma)$$
$$\mathbb{P}_J(op_1, \gamma)$$
$$\mathbb{P}_I(op, \omega)$$
$$\vdash$$

Proving the Correct Use of an Interface



Goal: $\mathbb{Q}_I(op, f(b,c), \omega)$
C shall correctly implement I

$$\frac{\begin{array}{l}\gamma_3 = step(\gamma_2, op_2, c)\\ \mathbb{Q}_J(op_2, c, \gamma_2)\\ \mathbb{P}_J(op_2, \gamma_2)\\ \gamma_2 = step(\gamma, op_1, b)\\ \mathbb{Q}_J(op_1, b, \gamma)\\ \mathbb{P}_J(op_1, \gamma)\\ \mathbb{P}_I(op, \omega)\end{array}}{\vdash \mathbb{Q}_I(op, f(b,c), \omega)}$$

Certifying a HTTP Server

Step 2. Verification Process

A **correct** impure computation

1. Uses its interfaces "correctly" ⇒ **Contract compliance**
2. Produces "correct" results ⇒ **Executions outcomes**

To **reason about impure computations**, FreeSpec's users prove:

1. `respectful_impure c w p`
2. `forall w' x, respectful_run c p w w' x -> pred w w' x`
   where `pred` is the desired properties of the final abstract state `w'` and the result `x` of `p`.

someone> Is it me, or you forgot to make a conclusion slide?
coq-der> Hold my coffee :)