

Collaboration between untrusted imperative code and Coq code

David Monniaux & Sylvain Boulmé

Verimag

January 29, 2025

Where we came from

Coq code with proofs + unverified oracles

VPL Verified polyhedra library, compute over **convex polyhedra in constraint form**, check results with a verified checker.

<https://github.com/VERIMAG-Polyhedra/VPL>

SatAns-Cert Verifying **unSAT** certificates

<https://github.com/boulme/satans-cert>

Chamois **CompCert + extra optimisations**, targets, security features...

<https://gricad-gitlab.univ-grenoble-alpes.fr/certicompil/Chamois-CompCert>



CompCert

extraction.v OCaml extraction of axioms

Retrieving command-line options

Parameter `optim_for_size: unit → bool`

Extract Constant `optim_for_size ⇒`
`"fun _ → !Clflags.option_0size"`

Oracles

Parameter `if_conversion_heuristic:`

`ident → Cminor.expr → Cminor.expr → Cminor.expr →`
`AST.typ → CminorSel.stmt → bool`

Extract Constant `if_conversion_heuristic ⇒`
`"Selectionaux.if_conversion_heuristic"`

Parameter `regalloc: RTL.function → res LTL.function`

Extract Constant `regalloc ⇒ "Regalloc.regalloc"`



Obvious problems

```
Axiom toto : { n : nat | n < 3 }  
Definition tata := proj1_sig toto
```

```
Lemma tata_lt_3: tata < 3
```

```
Program Definition tutu :=  
  match tata with  
    | 0  $\Rightarrow$  false  
    | 1 | 2  $\Rightarrow$  true  
    | S (S (S _))  $\Rightarrow$  _  
  end
```

Evaluating tutu leads to assert false in extracted code

Fatal error: exception Assert_failure("bad_dependent_pair.ml", 19, 2)

Obvious answer

“Values returned from external calls should have a simple type carrying no proof term.”

More generally: “Values returned from external calls should have a simple, non-dependent type.”

An assumption of determinism

Coq/Gallina is a pure functional language. OCaml is not.
To Coq, external calls are “uninterpreted functions”: f is unknown but it is known that if $x = y$ then $f(x) = f(y)$ across all calls.

```
Parameter nombre: unit → nat
```

```
Extract Constant nombre ⇒  
  ”let compteur = ref 0 in fun () →  
    compteur := S !compteur; !compteur”
```

```
Definition diff (_ : unit) :=  
  let x := nombre tt in let y := nombre tt in y - x
```

```
Lemma diff_0 : (diff tt) = 0
```

Run it, diff always returns 1.

Segmentation fault

Axiom flipper : unit $\rightarrow$ bool

Extract Constant flipper $\Rightarrow$
"(let s = ref true in fun () $\rightarrow$ s := not !s; !s)"

Definition disj_type (b : bool) : **Type** :=
if b then (bool * bool) else bool

Definition prepare (_:unit) : disj_type (flipper tt) :
if flipper tt as x return disj_type x
then (false, true) else true

Definition extract (x:bool) : disj_type x $\rightarrow$ bool :=
if x return disj_type x $\rightarrow$ bool
then snd
else fun x $\Rightarrow$ x

Definition extract_prepare (_ : unit) :=
extract (flipper tt) (prepare tt)

Pragmatism

Never do two calls to an external oracle with same input then rely on returns being identical.

External oracle = expensive, why call it twice for nothing?

It should be ok

“Non adversarial view”: Coq+OCaml code not meant to exploit loopholes.

Coq used to block honest mistakes in semantics etc.

A nondeterministic monad

??X is the type of nondeterministic values of type X

Monad bind extracted to banal OCaml function composition

All external calls should return in some ??X

All problems due to incoherent returns disappear

Practical issue: **everything now lives in a monad**, proofs are tedious

We are programmed to receive

You can check out any time you like, but you can never leave

(Glenn Frey, Don Henley, *Hotel California*)

Pointer equality

Parameter phys_eq: $\forall \{T : \mathbf{Type}\}, T \rightarrow T \rightarrow \mathbf{bool}$

Definition a := 1

Definition b := 1

Definition eq1 := phys_eq a a

Definition eq2 := phys_eq a b

Lemma eq_same : eq1=eq2

- ▶ OCaml interpreter: eq1,eq2 = true,false
- ▶ OCaml native compiler: eq1,eq2 = true,true

Solutions

Monadic

Parameter `phys_eq`: $\forall \{T : \mathbf{Type}\}, T \rightarrow T \rightarrow ??\text{bool}$

Jourdan's “local short-circuit”

“Fast path equal to slow path under the condition that two things are identical.”

Parameter `jourdan_physEq` $\{A\ B\}$ $(x\ y : A)$
 $(\text{fast_path} : x=y \rightarrow B)$
 $(\text{slow_path} : \text{unit} \rightarrow B)$
 $(\text{Hdet} : \forall (\text{Heq} : x=y), \text{fast_path Heq} = \text{slow_path}())$
 $: \{r : B \mid r = \text{slow_path}()\}$

Exiting the monad

Axiom `det_coerce`: $\forall \{A\} (k: ?? A) (v: unit \rightarrow A)$
 $(DET: \forall r, k \rightsquigarrow r \rightarrow r = v \text{ tt}), A$

Axiom `det_coerce_correct`: $\forall A (k: ?? A) v$
 $(DET: \forall r, k \rightsquigarrow r \rightarrow r = v \text{ tt}), (det_coerce\ k\ v\ DET) = (v\ \text{tt})$

“If the result of a nondeterministic computation is always equal to a deterministic result, then treat it as deterministic.”

Can implement JH Jourdan’s local short-circuit

Soundness by type safety

“A hash table takes values of universally quantified type α and produces values of type α ” also for dependent Coq types.

If a OCaml polymorphic hash table takes elements

$\{(a, b) \mid P(a, b)\}$

then it can only provide elements $\{(a, b) \mid P(a, b)\}$

(not just arbitrary (a, b))

But it is modeled as nondeterministic (cannot prove that it does not forget elements etc.)

Extends to more general settings, à la “theorems for free”:
polymorphic OCaml code can only act on α through functions
provided from Coq

Case study: a BDD package

Reduced ordered binary decision diagrams

Used in model-checking etc

Uses hash-consing, memoizing, hash tables all over

Algorithms for deciding formulas

Unique definition results over formulas

Everything in a monad

Final result: exit the monad

Logical soundness

All axioms we use have (trivial) models in Coq

- ▶ nondeterministic exit just calls the slow deterministic function
- ▶ hash tables that forget everything

Does not introduce compatibility issues

Trivial models would be inefficient
(BDDs would be fully unfolded)

Gratuitous advertisements

- ▶ professorship in formal methods & cybersecurity VERIMAG / UGA
- ▶ trying to find collaborations / a consortium for a project on Rust semantics, towards verified compilation (see Mattermost)



Chamois



Arsene

<https://gricad-gitlab.univ-grenoble-alpes.fr/certicompil/Chamois-CompCert>

<https://gricad-gitlab.univ-grenoble-alpes.fr/certicompil/Chamois-Arsene>