From Sets to Bits in Coq

Arthur Blot* Pierre-Évariste Dagand† Julia Lawall

Whisper – Inria Paris/CNRS

Motivation

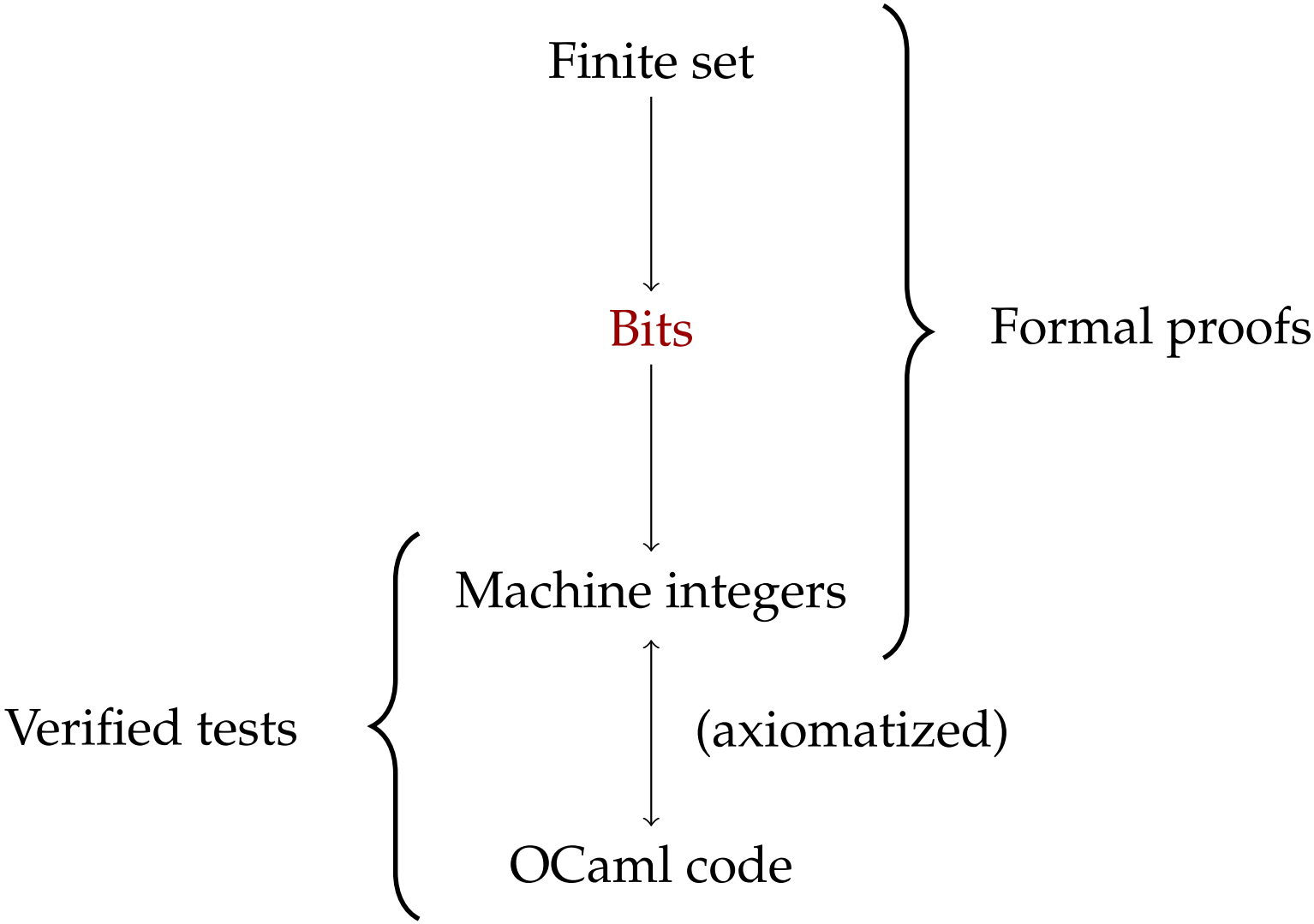
Bitsets in System Software:

- `bitmap` in Linux (*eg.* `cpumask`)
- Packet processing (*eg.* `flags` in IP header)
- Device drivers (*eg.* configuration bitsets)

Also, out of sheer curiosity:

How much of "Hacker's Delight" could we formalize in Coq?
What would the specifications look like?

Picture of the Battlefield



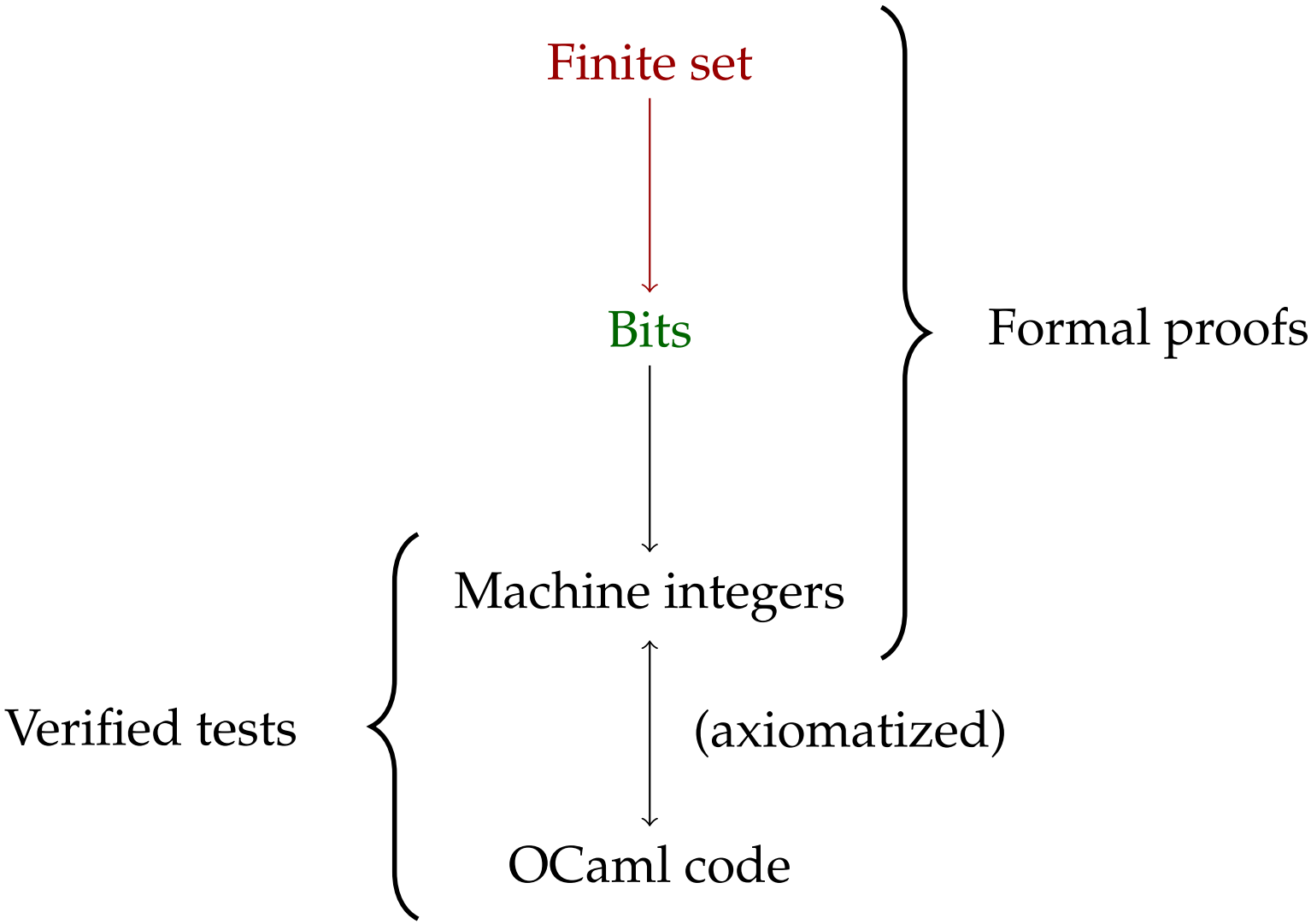
Bits in Coq

State of the art(?)

- Coq's `Coq.Bool.Bvector`
- CompCert's `lib/Integers.v`
- Why3's `bv`
- x86proved's `coq-bits`

Interface

```
                n : nat ⊢ BITS n : Type
 bs : BITS n, k : nat ⊢ getBit bs k : bool
      xs, ys : BITS n ⊢ {andB, orB, xorB} xs ys : BITS n
          xs : BITS n ⊢ invB xs : BITS n
  xs : BITS n, k: nat ⊢ {shrBn, shlBn} xs k : BITS n
```


Finite set
Bits
Machine integers
Formal proofs
Verified tests
(axiomatized)
OCaml code

From Bits to Finite Sets

Refinement (Cohen *et al.*)

Let E be a *finite* set of elements of a *finite* type.

$$\mathrm{repr}(bs, E) \triangleq E \cong \{e \in E \mid bs.(e) = \texttt{true}\}$$

Benefits

- Parametricity results & reasoning
- Compositionality
- Uniformity

From Bits to Finite Sets

Example: intersection

Operation

```
Definition inter {n} (bs: BITS n) (bs': BITS n): BITS n
  := andB bs bs'.
```

Specification

```
Lemma inter_repr n (bs: BITS n) (bs': BITS n) E E':
  repr bs E → repr bs' E' → repr (inter bs bs') (E :&: E').
```

From Bits to Finite Sets

Example: cardinality

Operation

```
int pop(unsigned x){
  static char table[256] = {0, 1, 1, 2, 1, ...};
  return table[x & 0xFF] + table[x >> 8 & 0xFF]
    + table[x >> 16 & 0xFF] + table[x >> 24 & 0xFF];

}
```

Specification

```
Lemma cardinal_repr n k (bs: BITS n) E:
  k %| n → k > 0 → repr bs E → cardinal k bs = #(#|E|).
```

"A less interesting algorithm", A. Warren

From Bits to Finite Sets

Example: number of trailing zeros

Operation

```
Definition ntz {n}(k: nat)(bs: BITS n): BITS n
  := subB #n (cardinal k (orB bs (negB bs))).
```

Specification

```
Lemma ntz_repr n (bs: BITS n) k x E:
  k %| n → k > 0 → repr bs E → x ∈ E →
    ntz k bs = #[arg min_(k < x in E) k].
```

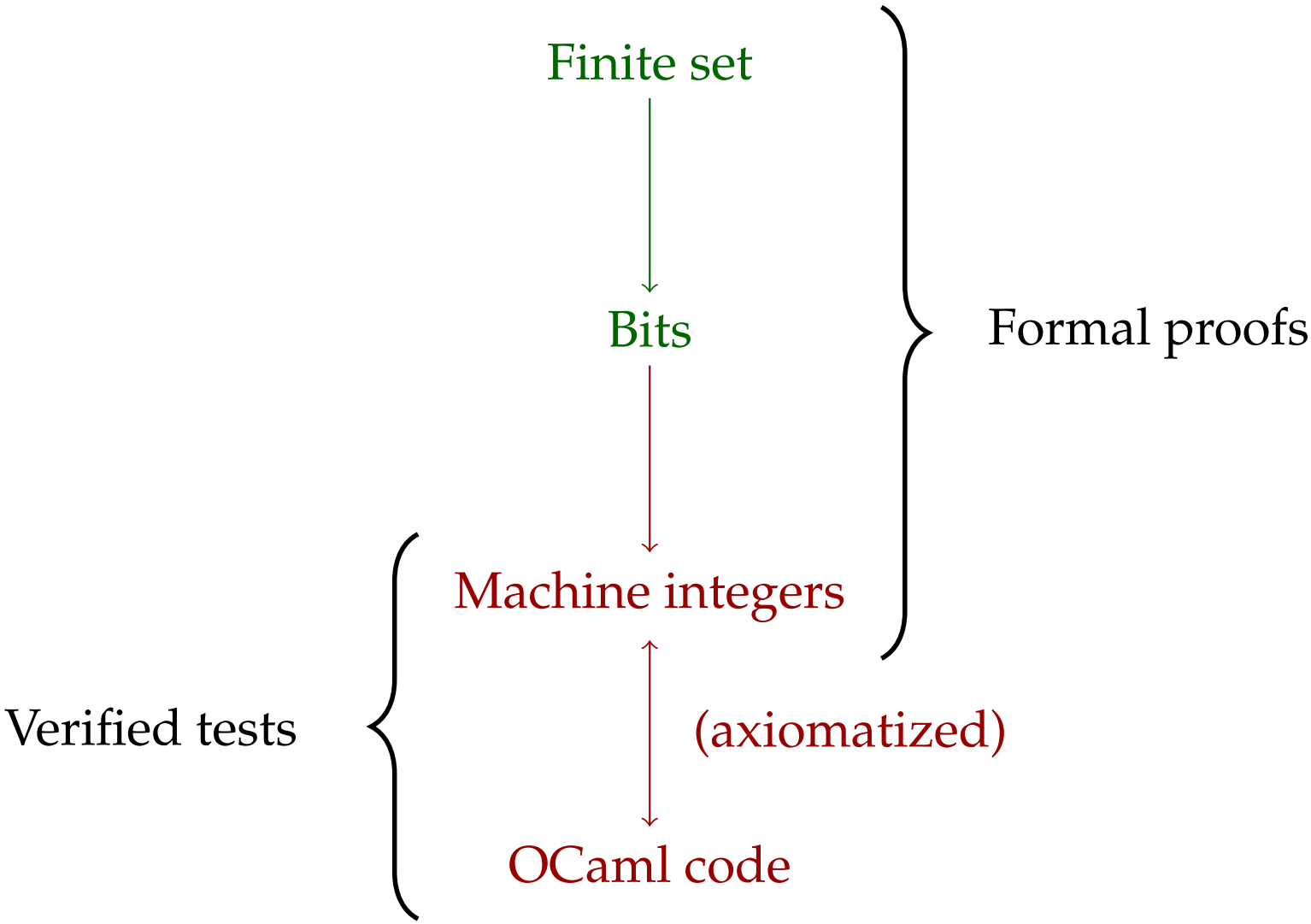
From Bits to Finite Sets

$$\texttt{\{set T\}} \quad \cong \quad \texttt{BITS \#T}$$

(morally, at least...)

Operations

- Empty set, full set
- Set membership, insertion and removal
- Set complement, union, intersection and sym. diff.
- Cardinality, minimal element
- ...


Finite set
Bits
Machine integers
Formal proofs
Verified tests
(axiomatized)
OCaml code

Machine Integers

Axioms

Axiomatic integers

```
Axiom Int8: Type.
Extract Inlined Constant Int8 => "int".
```

Operations

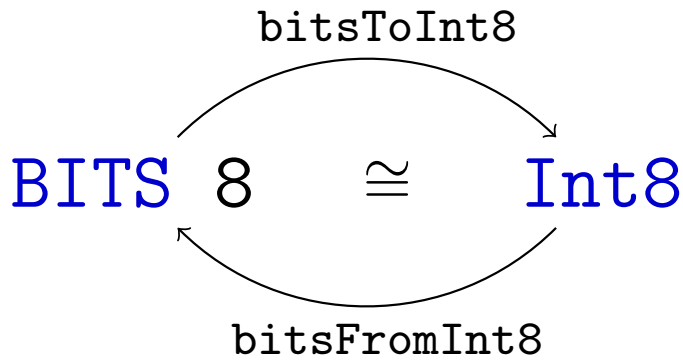
```
Axiom land: Int8 → Int8 → Int8.
Extract Inlined Constant land => "(land)".

Axiom lsl: Int8 → Int8 → Int8.
Extract Inlined Constant lsl => "(fun x y → (x lsl y))".
```

Could my `Int8` compute within Coq?

Machine Integers

Specification



(experimentally, perhaps...)

Methodology

- Write your program over `BITS` *n*
- Prove its correctness with finite sets
- Refine it to `Int`*n*
- Extract.

Coq for the Physicist

Step 1: Design an experiment

```
Definition bitsToInt8K_test: bool :=
 [forall bs , bitsFromInt8 (bitsToInt8 bs) == bs ].
```

Step 2: Observe

```
Axiom bitsToInt8K_valid: bitsToInt8K_test.
```

Step 3: Deduce

```
Lemma bitsToInt8K: cancel bitsToInt8 bitsFromInt8.
Proof.
  move=> bs; apply/eqP; move: bs.
  by apply/forallP: bitsToInt8K_valid.
Qed.
```

Coq for the Physicist

Experimental toolbox

Equality

```
Axiom eq: Int8 → Int8 → bool.
Extract Inlined Constant eq => "(=)".

Axiom eqInt8P : Equality.axiom eq.
```

Quantification / Enumeration

```
Axiom forallInt8 : (Int8 → bool) → bool.
Extract Inlined Constant forallInt8 => "Forall.forall_int".

Axiom forallInt8P P PP :
  viewP P PP → reflect (forall x, PP x)
                       (forallInt8 P).
```

Coq for the Physicist

Example: logical and

```
(* 1. Experiment: *)
Definition land_test: bool
  := forallInt8 (fun i =>
       forallInt8 (fun i' =>
         native_repr (land i i')
                     (andB (bitsFromInt8 i) (bitsFromInt8 i')))).

(* 2. Observation. *)
Axiom land_valid: land_test.

(* 3. Specification. *)
Lemma land_repr: forall i i' bs bs',
    native_repr i bs → native_repr i' bs' →
    native_repr (land i i') (andB bs bs').
Proof. (..) Qed.
```

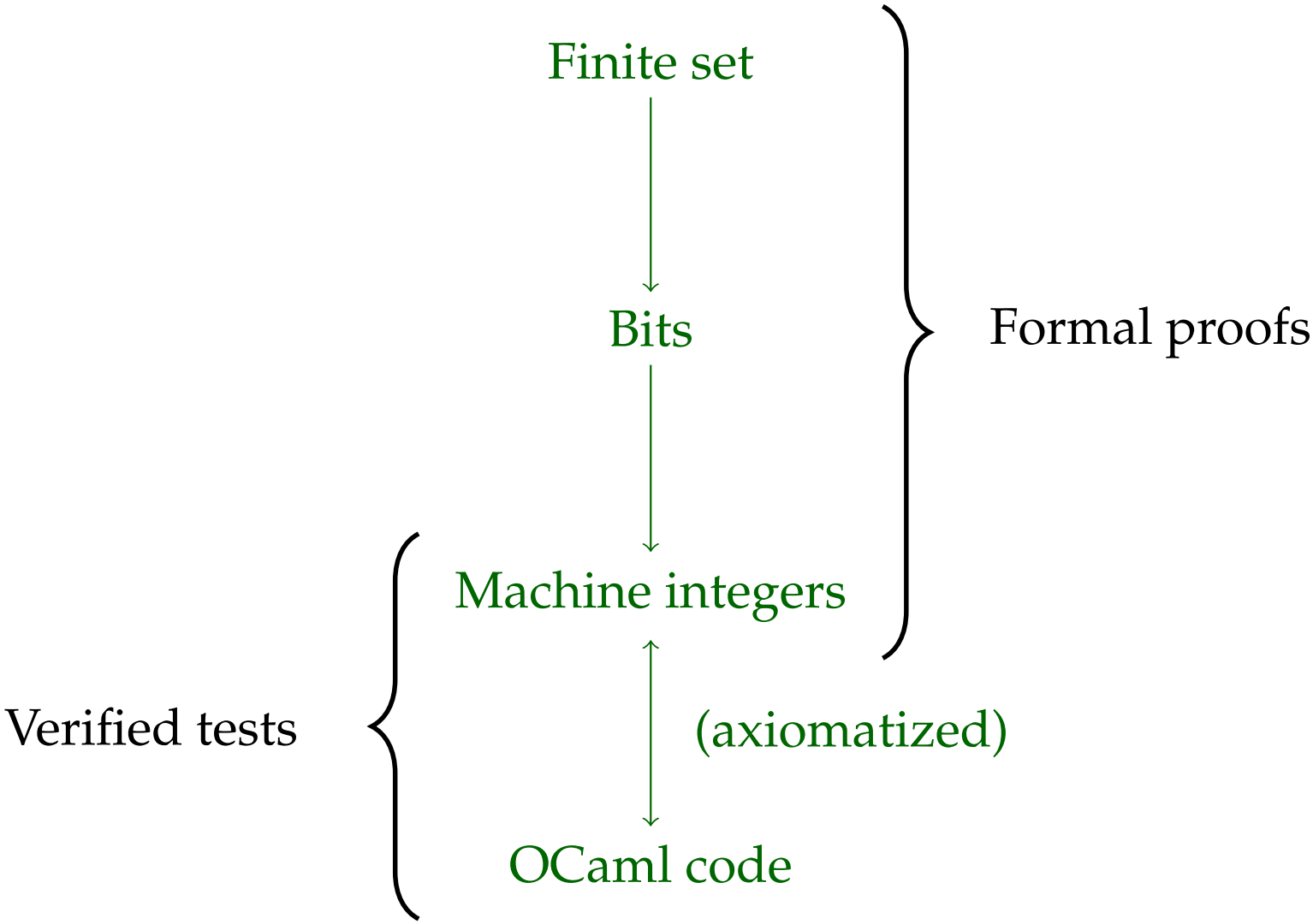
Coq for the Physicist

Experimental results

- On Coq-extracted tests: $4s$ for 8-bits
- On hand-crafted tests: $0.23s$ for 8-bits, $7\ h$ for 16 bits

Improvements

- Write aggregated, optimized tests
- Deduce individual specification
- Understand & address `BITS` performance


Finite set
Bits
Machine integers
Formal proofs
Verified tests
(axiomatized)
OCaml code

Application

Methodology

- Write a *polymorphic* program,
  quantifying over operations
- For proofs: instantiate to finite sets
- For extraction: instantiate to machine integers
- Parametricity theorem: refinement property

Application

Bloom filters

```
Module bloom (S: FINSET).

  Definition insert (S: T)(elt: P): T := (..).
  Definition mem (S: T)(e: P) : bool := (..).

End bloom.

Module bloom_Int8 := bloom_def BitsetInt8.
Module bloom_Finset := bloom_def Finset.
```

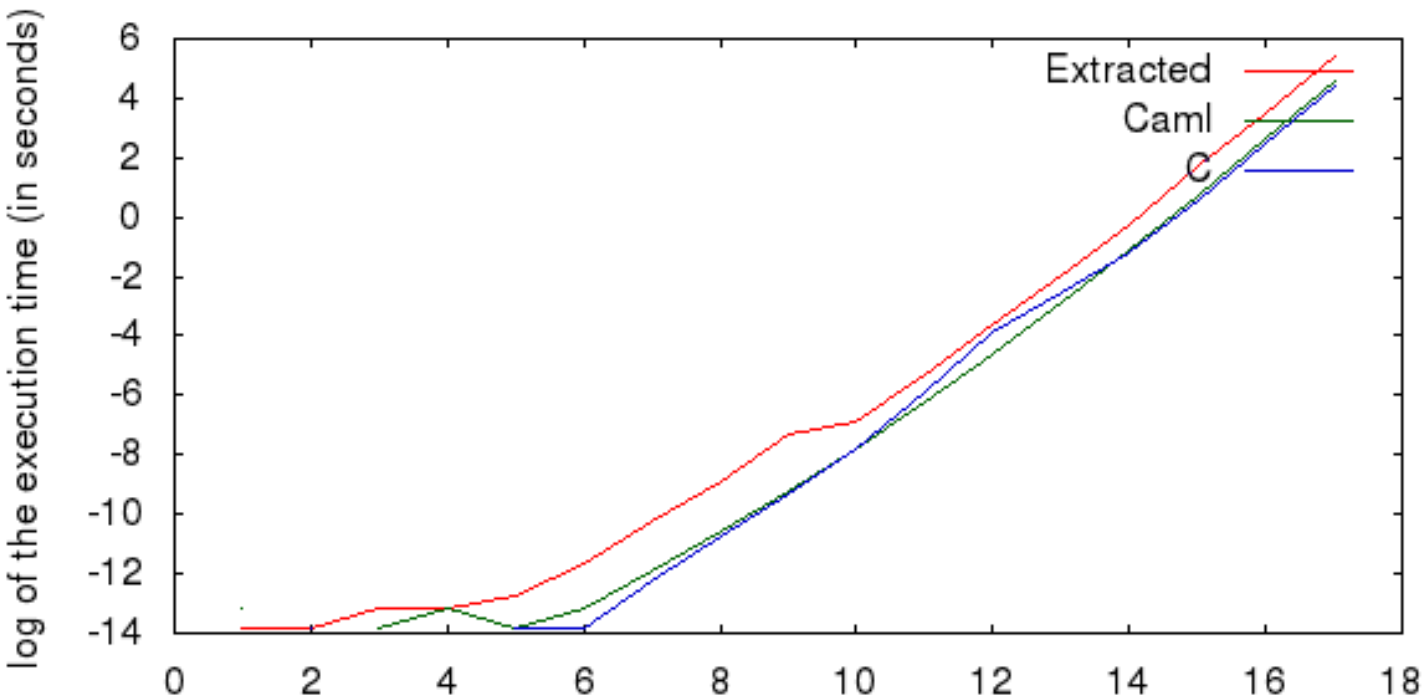
- Write the code once
- Get the refinement for free
- Prove in the abstract

Application

n-queens

Correctness

```
Theorem queens_correct n: n > 0 → n < wordsize →
countNQueens n = #(#|valid_pos n|).
```



Conclusion

We have. . .

- formally related operations on finite sets and bitsets,
  for "Hacker's and Prover's Delight"
- devised verified experiments,
  for trustworthy extraction of machine integers
- applied our formalism to obtain
  (easy) proofs and (efficient) programs

Future work & Applications

- Integrate into Cohen *et al.*'s "Refinements for free"
- Support arbitrary bit vector
- Ensure structural invariants (*eg.* CPU hierarchy)
- Automation?