Simple proofs for simple programs

Jean-Jacques Lévy
Iscas - Inria
Cardelli’s 60th anniversary
Cambridge, 2014-09-08

Plan

• Why3
• demo with merge sort
• conclusions

Goal

Write elegant proofs for elegant programs
+ training in program proofs checked by computers

History

• 1980 King’s buildings, Edinburgh
• powerdomains, sticks-stones, ML, fam, amber, squeak, quest, substitutions, objects, … biology … ???
• colleagues at DEC, MSR-INRIA Joint Centre (Cédric•Georges)
• “salon de la bijouterie” !

Happy 60th-birthday, Luca !
Welcome to the club !

.. with Chen Ran
**Why3**

- LRI (orsay) + Inria + Cnrs [Filliâtre, Paskevich, Marché…]
- small Pascal-like imperative programming language
  
  [with ML syntax 😳 !!]
- invariants + assertions in Hoare logic
  
  [+ recursive functions, inductive datatypes, inductive predicates ]
- interfaces with modern automatic provers
  
  [alt-ergo, cvc3, cvc4, eprover, gappa, simplify, spass, yices, z3, … ]
- interfaces with interactive proof assistants
  
  [coq, pvs, isabelle ]

**Hoare logic**

```ml
let swap (a: array int) (i: int) (j: int) =
  let v = a[i] in
  a[i] <- a[j];
  a[j] <- v

let selection_sort (a: array int) =
  for i = 0 to length a - 1 do
    let imin = ref i in
    for j = i + 1 to length a - 1 do
      invariant { i <= imin < j }
      invariant { forall k: int. i <= k < j -> a[imin] <= a[k] }
      if a[j] < a[imin] then imin := j
    done;
    swap a !imin i
done
```

---

**MLW programming language**

```ml
let swap (a: array int) (i: int) (j: int) =
  let v = a[i] in
  a[i] <- a[j];
  a[j] <- v

let selection_sort (a: array int) =
  for i = 0 to length a - 1 do
    let imin = ref i in
    for j = i + 1 to length a - 1 do
      if a[j] < a[imin] then imin := j
    done;
    swap a !imin i
done
```

---

**Why3 theories**

- theories about arrays

```ml
let swap (a: array int) (i: int) (j: int) =
  requires { 0 <= i < length a \&\& 0 <= j < length a }
  ensures { exchange (old a) a i j };
  let v = a[i] in
  a[i] <- a[j];
  a[j] <- v
```

(see the why3 libraries)

An example
Mergesort (3/3)

Full program (2/2)

```
4: let i = ref lo in
    let j = ref hi in
    for k = lo to hi-1 do
        invariant (lo <= k < hi ∧ lo <= i ∧ j < hi)
        invariant (k = i + hi - j)
        invariant (sorted_sub a lo k)
        invariant (forall k1 k2: int. lo <= k1 < k2 < j → a[k1] <= b[k2])
        invariant (bitonic b [1..j])
        invariant (modified_inside a (at a '4') lo hi)
        assert (i < j)
        if b[i] < b[j - 1] then
            begin a[k] <- b[i]; i := i + 1 end
        else
            begin j := j - 1; a[k] <- b[j] end
        done
```

Full program (1/2)

```
let rec mergesort1 (a: array int) (lo hi: int) =
  requires (Array.length a = Array.length b ∧
    0 <= lo <= (Array.length a) ∧ 0 <= hi <= (Array.length a))
  ensures (sorted_sub a lo hi ∧ modified_inside (old a) a lo hi)
  let m = div (lo+hi) 2 in
  assert (lo < m < hi)
  mergesort1 a lo m;
  '2': mergesort1 a m hi;
  assert (array_eq_sub (at a '2') a lo m);
  for i = lo to m-1 do
      invariant (array_eq_sub b a lo i)
      b[i] <- a[i]
      done;
  assert (array_eq_sub a b lo m);
  for j = m to hi-1 do
      invariant (array_eq_sub_rev_offset b a m j (hi - j))
      invariant (array_eq_sub a b lo m)
      b[j] <- a[m + hi - 1 - j]
      done;
  assert (array_eq_sub a b lo m);
  assert (sorted_sub b lo m);
  assert (array_eq_sub_rev_offset b a n hi 0);
  assert (sorted_sub b m hi);
```

Full program (logic 1/2)

```
module MergeSort

use import Int
use import int.EuclideanDivision
use import Int.Div2
use import Ref
use import array.Array
use import array.ArraySorted
use import array.ArrayPerm
use import array.ArrayEq
use map.Map as M
clone map.MapSorted as N with type elt = int, predicate le = (<=)

predicate map_eq_sub_rev_offset (a1 a2: M.Map int int) (1 u: int) (offset: int) =
  forall 1 i2 : int. 1 <= i2 < u → M.get a1 i2 = M.get a2 (offset + 1 + u - 1 - i2)
predicate array_eq_sub_rev_offset (a1 a2: array int) (1 u: int) (offset: int) =
  map_eq_sub_rev_offset a1.els a2.els 1 u offset
predicate map_map_sorted_sub (a: M.Map int int) (1 u: int) =
  forall 1 i2 : int. 1 <= i2 < u → M.get a i2 <= M.get a 1
predicate sorted_sub (a: array int) (1 u: int) =
  map_map_sorted_sub a.els 1 u
```
Full program (logic 2/2)

```coq
predicate map_bitonic_sub (a: M.map int int) (l u: int) = l < u -> exists i: int. l <= i < u /
    M.sorted_sub a l i /
    map_bitonic_sub a i u

predicate bitonic (a: array int) (l u: int) =
    map_bitonic_sub a e.lts l u

lemma map_bitonic_incr : forall a: M.map int int, l u: int.
    map_bitonic_sub a l u -> map_bitonic_sub a (l+1) u

lemma map_bitonic_decr : forall a: M.map int int, l u: int.
    map_bitonic_sub a l u -> map_bitonic_sub a (l-1) u

predicate modified_inside (a1 a2: array int) (l u: int) =
    (Array.length a1 = Array.length a2) /
    array_eq_sub a1 a2 @ l /
    array_eq_sub a1 a2 @ (l+1) /
    array_eq_sub a1 a2 @ (l+2)
```

Coq files

```coq
Lemma sorted_sub_weakening: forall (a:(Mmap.Map.map Z Z Z Z Z)) (l u: Z) (l' l" u) (l' <= l " <= u),
    (l1 <= l2) -> (u' <= u2) -> sorted_sub2 a l u -> sorted_sub2 a l' u'.
Proof.
move=> a l u' l' u' ll le u Hl u_sorted.
unfold sorted_sub2 => ll l2 [ll le ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll ll
Conclusion 1

- **Automatic** part of proof for **tedious** case analyzes
- **Interactive** proofs for the **conceptual** part of the algorithm
- From interactive part, one can call the automatic part
  - possible extensions of Why3 theories
  - but typing problems (inside Coq)

Conclusion 2

- Hoare logic prevents to write awkward denotational semantics
- Nobody cares about termination?! 😳
- Explore **simple** programs about algorithms before jumping to **large** programs.
- Why3 **memory model** is naive. It’s a «back-end for other systems».
- Also experimenting on **graph** algorithms and prove all algorithms in Sedgewick’s book.

Conclusion 3

- Why3 is **excellent** for mixing formal proofs and SMT’s calls
- Still **rough** for beginners
- Concurrency ?
- Functional programs ?
- Hoare logic   vs   Type refinements (F* [MSR])
- **Frama-C** project at french CEA extends Why3 to C programs.