Plan

- Why3
- demos
- conclusions

Goal

Write elegant programs
with elegant correctness proofs
+ training in program proofs

Why3 (1/8)

A programming language tells you what a program does, Why3 tells you why it works.

- 3rd release of system Why
- developed at LRI (orsay) + Inria

[Jean-Christophe Filliâtre,
Claude Marché,
Andrei Paskevich,
Guillaume Melquiond,
Vincent Bolot,
et al]
Why3 (2/8)

- small Pascal-like imperative programming language
  [ with ML syntax ![ ] ]
- invariants + assertions in Hoare logic
  [+ recursive functions, inductive datatypes, inductive predicates ]
- interfaces with modern SMT's
  [ alt-ergo, cvc3, cvc4, eprover, gappa, simplify, spass, yices, z3 ]
- interfaces with interactive proof assistants
  [ coq, pvs, isabelle-hol? ]

Why3 (3/8)

- programming language MLW

```
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 (4/8)

- Hoare logic

```
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[kimin] <= a[k] }
      if a[j] < a[imin] then imin := j
    done;
    swap a !imin i
  done
```

Why3 (5/8)

- theories on arrays

```
let swap (a: array int) (i: int) (j: int) =
  requires { 0 <= i < length a \land 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)
```

Why3 (6/8)

- theories on arrays

```plaintext
let selection_sort (a: array int) =
  ensures { sorted a \land permut (old a) a }
  'i':
  let imin = ref i in
  for i = 0 to length a - 1 do
    invariant { sorted_sub a 0 i \land permut (at a 'l) a}
    invariant { forall k1 k2: int. 0 <= k1 < i < k2 < length a -> a[k1] <= a[k2] }
    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
```

Why3 (7/8)

- interfaces with interactive proof assistants
- PVS [SRI, Shankar]
- Coq [Inria, Herbelin et al]
  - Why3 theories are translated to Coq
  - lengthy proofs are feasible
  - use SSreflect commands to shorten proofs [MSR-Inria, Gonthier et al]
- unfortunately Why3 is not fully compatible with SSreflect

Why3 (8/8)

- SMT tool successful if «good assertion»
  - impact on writings of Hoare logic formulae
  - impact on program text
- Alt-Ergo among best [LRI, Conchon, et al]
- Z3 is excellent [MSRR, Bjorner/de Moura]
- CVC3 top on recursive datatypes
- Gappa for real numbers [Inria, Melquiond]
A few sorting algorithms

- demos
- insertion sort

A few sorting algorithms

- quicksort

Conclusions

Conclusion (1/3)

- **Automatic** part of proof for **tedious** case analyzes
- **Interactive** proofs for the **conceptual** part of the algorithm

  - the ideal world

- From interactive part, one must call the automatic part
  - possible extensions of Why3 theories
  - but typing problems (inside Coq)
Conclusion (2/3)

- 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 is a «back-end for other systems».
- Plan to experiment on graph algorithms and prove all Sedgewick’s book on algorithms.

Conclusion (3/3)

- Why3 is excellent for mixing formal proofs and SMT’s calls
- Interface 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.