The Why3 tool for deductive verification and verified OCaml libraries

Jean-Christophe Filliâtre
CNRS

Frama-C & SPARK Day 2019

plan

1. an overview of Why3
2. a short demo
3. verified OCaml libraries

started in 2001, as an intermediate language in the process of verifying C and Java programs ($\sim$ Boogie)

today, joint work with

François Bobot

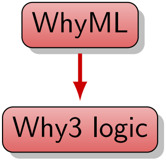
Claude Marché

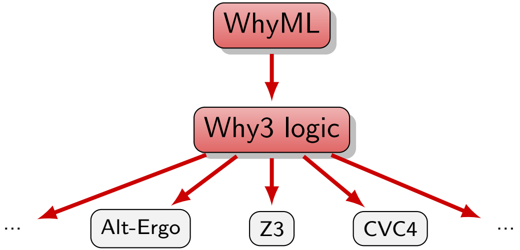
Guillaume Melquiond

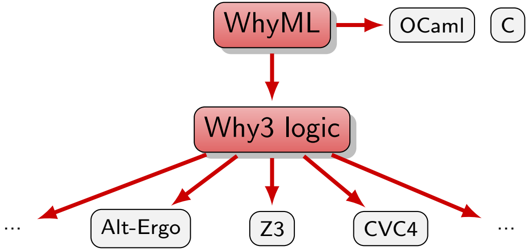
Andrei Paskevich

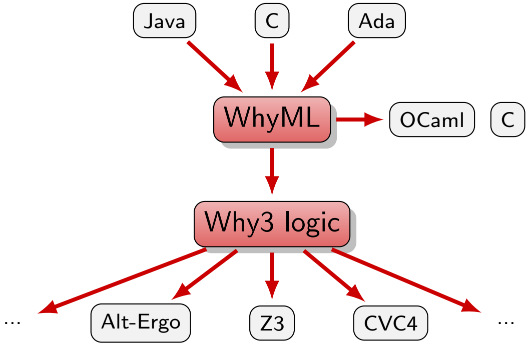
Why3: a deductive verification environment

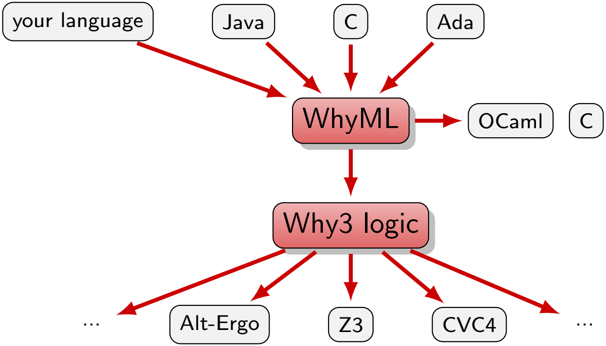
- a logic
- a programming language, WhyML, with a VCGen
- a logic and programming library
- an interface with theorem provers
- a toolbox to build/save/update/replay proofs

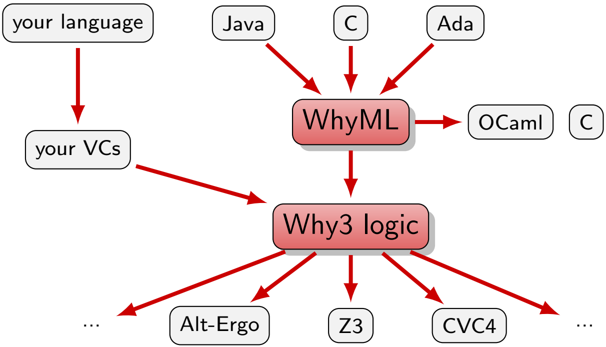

WhyML
Why3 logic


WhyML
Why3 logic
...
Alt-Ergo
Z3
CVC4
...


WhyML
OCaml
C
Why3 logic
...
Alt-Ergo
Z3
CVC4
...


Java
C
Ada
WhyML
OCaml
C
Why3 logic
...
Alt-Ergo
Z3
CVC4
...


your language
Java
C
Ada
WhyML
OCaml
C
Why3 logic
...
Alt-Ergo
Z3
CVC4
...


your language
Java
C
Ada
your VCs
WhyML
OCaml
C
Why3 logic
...
Alt-Ergo
Z3
CVC4
...

a logic

a total, polymorphic first-order logic, extended with

- algebraic data types and pattern matching
- recursive definitions
- (co)inductive predicates
- mapping type $\alpha \to \beta$, λ-notation, application

[FroCos 2011, CADE 2013, VSTTE 2014]

a programming language

WhyML $\sim$ small subset of OCaml

- polymorphism
- pattern matching
- exceptions
- mutable data with controlled aliasing [ESOP 2013]
- ghost code and ghost data [CAV 2014]
- contracts, loop and type invariants

a library

- a logic library
  - integers, real numbers, lists, sets, maps, sequences
  - useful theories, e.g.

$$\texttt{sum f a b} \stackrel{\text{def}}{=} \sum_{a \le i < b} f(i)$$

$$\texttt{numof p a b} \stackrel{\text{def}}{=} \#\{i \mid a \le i < b \land p(i)\}$$

- a programming library
  - references, arrays, stacks, queues, sets, maps
  - floating-point arithmetic [ARITH 2007]
  - machine integers

an interface with theorem provers

Why3 currently supports 25+ ITPs and ATPs

for each prover, a special "driver" file controls [Boogie 2011]

- logical transformations to apply
- input/output format
- predefined symbols, axioms to be removed

users can extend Why3 with support for a new theorem prover

a toolbox to handle proofs

proofs are built by

- applying logical transformations (e.g. splitting, case analysis)
- calling theorem provers

proofs are saved, for edition/replay in the future

proofs are updated automatically/heuristically
when changes occur (code, spec, environment) [VSTTE 2013]

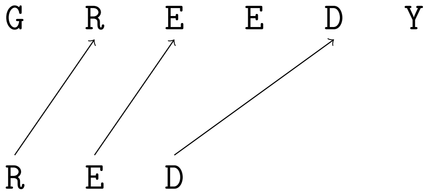
2

a short demo

a sequence v is a subsequence of u if v can be obtained by erasing elements of u (possibly none)

devise and implement an algorithm to check whether v is a subsequence of u in linear time

example



$$
\begin{aligned}
&\text{subsequence}(v,\ u) \stackrel{\text{def}}{=} \\
&\qquad i \leftarrow 0 \\
&\qquad j \leftarrow 0 \\
&\qquad \textbf{while } i < |v| \wedge j < |u| \\
&\qquad\qquad \textbf{if } v[i] = u[j] \\
&\qquad\qquad\qquad i \leftarrow i + 1 \\
&\qquad\qquad j \leftarrow j + 1 \\
&\qquad \textbf{return } i = |v|
\end{aligned}
$$

in Why3 syntax

```
type char = int32
type word = array char

let is_subsequence (v u: word) (lv lu: int32) : bool
= let ref i = 0 in
  let ref j = 0 in
  while i < lv && j < lu do
    if v[i] = u[j] then i <- i + 1;
    j <- j + 1
  done;
  i = lv
```

gallery of verified programs

http://toccata.lri.fr/gallery/why3.en.html

more than 160 examples

- data structures: AVL, red-black trees, skew heaps, Braun trees, ropes, resizable arrays, etc.
- sorting, graph algorithms, etc.
- solutions to most competition problems (VSComp, VerifyThis)

3

verified OCaml libraries

ANR-funded project VOCaL (2015–2020)

partners:

- LRI, Univ Paris-Sud
- Gallium, Inria Paris
- PACSS, Verimag
- TrustInSoft
- OCamlPro

VOCaL — a Verified OCaml Library

a general-purpose data structures and algorithms library

- priority queues
- hash tables
- sequences
- sets / maps
- resizable arrays
- graph algorithms
- sorting
- searching
- union-find
- text algorithms

possible clients: Coq, Frama-C, Astrée, Infer, Alt-Ergo, Cubicle, EasyCrypt, ProVerif, etc.

GOSPEL — a specification language for OCaml

interface files (`.mli`) are augmented with a formal specification

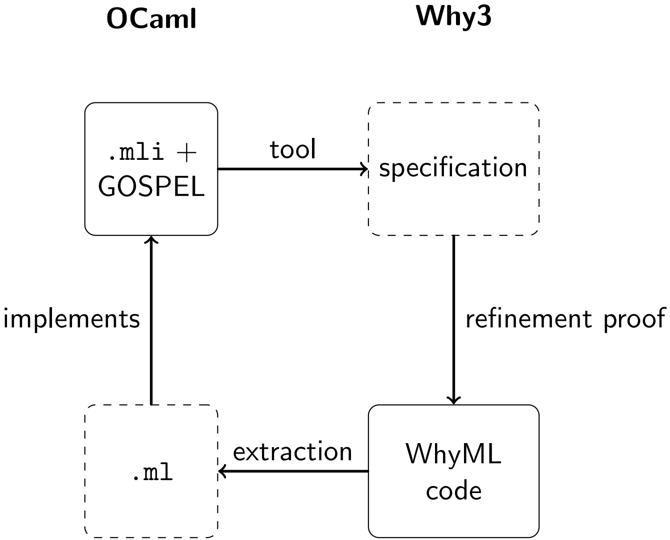
- within special comments (à la JML / ACSL)
- using a simple, first-order logic
- which can be ignored at first sight

implementation based on the OCaml parser

```
(** Resizable arrays. ... *)

type 'a t
(** The type of resizable arrays. *)
(*@ ephemeral *)
(*@ mutable model view: 'a seq *)
(*@ invariant length view <= Sys.max_array_length *)

val init: dummy:'a -> int -> (int -> 'a) -> 'a t
(** [init dummy n f] creates a new ... *)
(*@ a = init ~dummy n f
    requires 0 <= n <= Sys.max_array_length
    ensures  length a.view = n
    ensures  forall i. 0 <= i < n -> a.view[i] = f i *)

...
```

we use a combination of three tools

- Why3
- CFML [Charguéraud, ICFP 2010]
  - higher-order separation logic, within Coq
  - targets pointer programs
- Coq
  - automated translation to OCaml
  - targets purely applicative programming


OCaml
Why3
.mli + GOSPEL
tool
specification
implements
refinement proof
.ml
extraction
WhyML code

challenges

- higher-order functions
- RTAC or not?
- proofs of complexity
- mutable state
- machine arithmetic

sometimes you can assume functions to be pure

example:

```
val binary_search:
   ('a -> 'a -> int) -> 'a array -> int -> int -> 'a -> int
(*@ r = binary_search cmp a fromi toi v
    requires is_pre_order cmp
    requires forall i j.
             fromi <= i <= j < toi -> cmp a.(i) a.(j) <= 0
    ...
```

sometimes you cannot

```
val iter: (elt -> unit) -> set -> unit
```

two challenges here

- how to specify the iteration
- how to verify the implementation

we contributed

- a new way to specify iteration [NFM 2016]
- verified iterators, cursors, and lazy sequences [CPP 2017]

runtime assertion checking, or not?

VOCaL modules can be used in

- verified code
  $\Rightarrow$ we prove that all preconditions are met
- unverified code
  $\Rightarrow$ which behavior for a precondition that is not met?

we distinguish `checks` (runtime check) and `requires`

we provide two versions for each function
(with and without runtime checks)

```
val binary_search:
   ('a -> 'a -> int) -> 'a array -> int -> int -> 'a -> int
(*@ r = binary_search cmp a fromi toi v
    requires is_pre_order cmp
    checks   0 <= fromi <= toi <= Array.length a
    requires forall i j. fromi <= i <= j < toi ->
               cmp a.(i) a.(j) <= 0
    ensures  ...
```

one precondition cannot be checked at runtime
one precondition would be too costly to check at runtime

beyond functional correctness,
we also prove worst-case complexity bounds

using time credits [JAR 2017, ESOP 2018]

non-trivial case study: union-find

```
val find: 'a elem -> 'a elem
(*@ r = find x [uf]
    requires mem x uf
    requires $(2 * alpha(card uf) + 4)
    ensures  r = repr uf x *)
```

OCaml features mutable data structures, which means

- aliasing
  - what about `Vector.append v v` ?
    (so far, we assume disjoint arguments)

- ownership and permissions
  - what about a container with mutable elements?
    (so far, we assume owned elements)

contribution:
separation logic with read-only permissions [ESOP 2017]

we prove the absence of arithmetic overflows

risk of specification explosion

- additional preconditions in client code
- sometimes difficult to exhibit bounds
- precondition/proof sometimes not even possible

solution: machine integers with limited growth
[VSTTE 2015]

VOCaL so far

- eight verified modules (`https://github.com/vocal-project`)

module	loc	tool
`HashTable`	150	CFML
`UnionFind`	60	Why3,CFML
`Lists`	50	Coq
`Vector`	150	Why3
`PairingHeap`	42	Why3
`ZipperList`	58	Why3
`Arrays`	63	Why3
`PriorityQueue`	81	Why3

- publications
  - project overview [ML 2017]
  - complexity proofs [JAR 2017, ESOP 2018]
  - case studies [VSTTE 2016 $\times 2$, JAR 2017]
  - iteration [NFM 2016, CPP 2017]
  - mutable state [ESOP 2017]