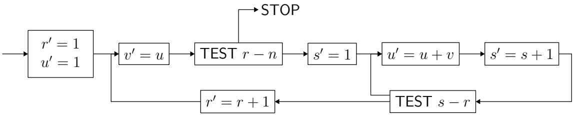
Deductive Program Verification with Why3

Jean-Christophe Filliâtre
CNRS

Parkas seminar, École Normale Supérieure
January 16, 2012



program + correctness property → mathematical statement → proof say we want to do modular arithmetic, with modulus m

we use unsigned integers on w bits

consider the multiplication

$$
\begin{array}{l}
\texttt{mul}\ x\ y \stackrel{\text{def}}{=} \\
\quad k \leftarrow 2^{w-2} \\
\quad r \leftarrow 0 \\
\quad \textbf{while}\ k \neq 0\ \textbf{do} \\
\quad\quad r \leftarrow 2r \mod m \\
\quad\quad \textbf{if}\ x \,\&\, k \neq 0\ \textbf{then} \\
\quad\quad\quad r \leftarrow (r + y) \mod m \\
\quad\quad k \leftarrow k/2 \\
\quad \textbf{done} \\
\quad \textbf{return}\ r
\end{array}
$$

$$
\begin{array}{cccccccc}
 & y & = & y_3 & y_2 & y_1 & y_0 \\
\times & x & = & 1 & 0 & 1 & 1 \\
\hline
 & & & y_3 & y_2 & y_1 & y_0 \\
 & & y_3 & y_2 & y_1 & y_0 & \\
 & 0 & 0 & 0 & 0 & & \\
y_3 & y_2 & y_1 & y_0 & & & \\
\hline
\end{array}
$$

let us prove this program returns r such that

$$(Q_1)\quad 0 \le r < m \qquad \text{and} \qquad (Q_2)\quad r \equiv xy \pmod{m}$$

assumptions are

$$\begin{array}{lll} (P_1) & 2 \le w & \text{at least 2 bits} \\ (P_2) & 0 < m \le 2^{w-1} & \text{modulus not too large} \\ (P_3) & 0 \le x, y < m & \text{arguments modulo } m \end{array}$$

$$(I_1) \quad 0 \leq r < m$$

$$(I_2) \quad k = 0 \quad \vee \quad \exists i.\, k = 2^i \wedge 0 \leq i \leq w - 2$$

$$(I_3) \quad xy \equiv \begin{cases} 2kr + (x \mod (2k))y & \pmod{m} \quad \text{if } k \neq 0, \\ r & \pmod{m} \quad \text{if } k = 0 \end{cases}$$

$$\texttt{mul}\ x\ y \stackrel{\text{def}}{=}$$

```
    preconditions (P1) (P2) (P3)
    k ← 2^(w−2)
    r ← 0
    while k ≠ 0 do   invariants (I1) (I2) (I3)
       r ← 2r  mod m
       if x & k ≠ 0 then
          r ← (r + y)  mod m
       k ← k/2
    done
    return r
    postconditions (Q1) (Q2)
```

no overflow when computing $2r$

$\texttt{mul}\ x\ y \stackrel{\text{def}}{=}$

preconditions

$k \leftarrow 2^{w-2}$

$r \leftarrow 0$

while $k \neq 0$ **do** invariants

$r \leftarrow \boxed{2r} \mod m$

if $x \,\&\, k \neq 0$ **then**

$r \leftarrow (r + y) \mod m$

$k \leftarrow k/2$

done

return r

postconditions

$$
\begin{array}{rl}
(P_1) & \ldots \\
(P_2) & 0 < m \leq 2^{w-1} \\
(P_3) & \ldots \\
(I_1) & 0 \leq r < m \\
(I_2) & \ldots \\
(I_3) & \ldots \\
\hline
 & 2r < 2^w
\end{array}
$$

involves

- 3 overflow checks
  - 2^{w-2}
  - $2r$
  - $r + y$
- invariant holds initially
  - 3 properties
- invariant is preserved
  - 3 properties $\times$ 2 cases $(x \& k)$ $\times$ 2 cases $(k = 0)$
- postcondition holds
  - $0 \leq r < m$
  - $r \equiv xy \pmod{m}$
- termination

A Quote

The point is that when we write programs today, we know that we could in principle construct formal proofs of their correctness if we really wanted to [...].

The point is that when we write programs today, we know that we could in principle construct formal proofs of their correctness if we really wanted to [...].

Donald E. Knuth
1974 ACM Turing Award Lecture
"Computer Programming as an Art"

A Breakthrough

at some point, we started mechanizing the logic
(AUTOMATH, de Bruijn 1967)

formulas are data structures

a program

- can check a proof
- can search for a proof

A Dedicated Logic

- Hoare Logic (Hoare 1969)
- KIV (Reif 1989)
- B (Abrial 1996)
- Separation Logic (Reynolds 2002)
    - Smallfoot (Berdine Calcagno O'Hearn 2006), HOLfoot (Tuerk 2010)
    - VeriFast (Jacobs Smans Piessens 2008)
- KeY (Hähnle 2003)
- ATS (Xi 2003)
- Guru (Stump 2009)

Another Approach

use general purpose theorem provers instead

- interactive proof assistants
  - Coq (Huet Coquand 1984, Paulin 1989)
  - ACL2 (Boyer Moore 1988, Kaufmann Moore 1994)
  - Isabelle (Paulson 1989)
  - PVS (Shankar Owre Rushby 1993)

- automated theorem provers
  - TPTP provers
    - Vampire (Voronkov), E (Schulz), SPASS (Max Planck Institut)
  - SMT solvers
    - Simplify (Nelson), Yices (Dutertre), Alt-Ergo (Conchon), Z3 (de Moura), CVC3 (Barrett Tinelli)
  - dedicated provers
    - Gappa (Melquiond), BAPA (Kuncak)

An Embedding

program $\longrightarrow$ a logical definition

- Coq (e.g. CompCert)
  - Ynot (Nanevski Morrisett Birkedal 2006)
  - Russell (Sozeau 2007)
  - CFML (Charguéraud 2010)
- PVS (e.g. use at NASA)
- ACL2 (e.g. floating point division AMD K5)
- Isabelle/HOL (e.g. L4.verified)
  - Simpl (Schirmer 2004)

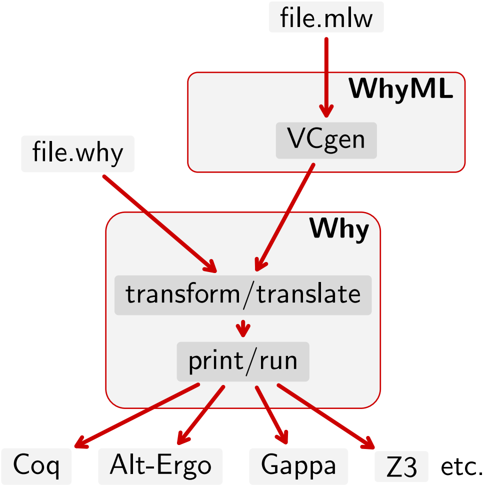
program + specification $\longrightarrow$ verification conditions

- Boogie (Barnett Leino 2006)
  - SPEC# (Barnett Leino Schulte 2004)
  - VCC (Cohen Moskal et al 2009)
  - Dafny (Leino 2010)
- Why (Filliâtre 2003)
  - Krakatoa (Marché Paulin Urbain 2004)
  - Caduceus (Filliâtre Marché 2004)
  - Frama-C (CEA/List 2008)
- Jahob (Zee Kuncak Rinard 2009)

Part II

Why3

joint work with C. Marché, A. Paskevich, F. Bobot, G. Melquiond



Simplicity

verification conditions should be made as simple as possible

- no memory store
- verification conditions about the contents of data structures

still all relevant details must be captured

- termination, array bound checking, etc.
- executable code

provide as much proof automation as possible

then turn to interactive proof to handle unproved VC

consequences

- VC must indeed be simple
- the logic of Why3 is a compromise
- still a lot of work to handle many provers

Influence of ML

Why3 features

- declaration-level polymorphism
- algebraic data types and pattern matching

WhyML has

- type inference
- currying
- abstract data types

A Problem

7	53	183	439	863	497	383	563	79	973	287	63	343	169	583
627	343	773	959	943	767	473	103	699	303	957	703	583	639	913
447	283	463	29	23	487	463	993	119	883	327	493	423	159	743
217	623	3	399	853	407	103	983	89	463	290	516	212	462	350
960	376	682	962	300	780	486	502	912	800	250	346	172	812	350
870	456	192	162	593	473	915	45	989	873	823	965	425	329	803
973	965	905	919	133	673	665	235	509	613	673	815	165	992	326
322	148	972	962	286	255	941	541	265	323	925	281	601	95	973
445	721	11	525	473	65	511	164	138	672	18	428	154	448	848
414	456	310	312	798	104	566	520	302	248	694	976	430	392	198
184	829	373	181	631	101	969	613	840	740	778	458	284	760	390
821	461	843	513	17	901	711	993	293	157	274	94	192	156	574
34	124	4	878	450	476	712	914	838	669	875	299	823	329	699
815	559	813	459	522	788	168	586	966	232	308	833	251	631	107
813	883	451	509	615	77	281	613	459	205	380	274	302	35	805

A Problem

7	53	183	439	863	497	383	563	79	973	287	63	343	169	583
627	343	773	959	943	767	473	103	699	303	957	703	583	639	913
447	283	463	29	23	487	463	993	119	883	327	493	423	159	743
217	623	3	399	853	407	103	983	89	463	290	516	212	462	350
960	376	682	962	300	780	486	502	912	800	250	346	172	812	350
870	456	192	162	593	473	915	45	989	873	823	965	425	329	803
973	965	905	919	133	673	665	235	509	613	673	815	165	992	326
322	148	972	962	286	255	941	541	265	323	925	281	601	95	973
445	721	11	525	473	65	511	164	138	672	18	428	154	448	848
414	456	310	312	798	104	566	520	302	248	694	976	430	392	198
184	829	373	181	631	101	969	613	840	740	778	458	284	760	390
821	461	843	513	17	901	711	993	293	157	274	94	192	156	574
34	124	4	878	450	476	712	914	838	669	875	299	823	329	699
815	559	813	459	522	788	168	586	966	232	308	833	251	631	107
813	883	451	509	615	77	281	613	459	205	380	274	302	35	805

$$563 + 699 + \cdots + 522 + 451 = 7805$$

$$f(i, C) \quad = \quad \max_{j \in C} \ \mathtt{m}[i][j] + f(i+1, C \backslash \{j\})$$

A Solution

$$f(i, C) \quad = \quad \max_{j \in C} \mathtt{m}[i][j] + f(i+1, C \backslash \{j\})$$

```
let rec maximum i cols =
  if i = 15 then
    0
  else begin
    let r = ref 0 in
    for j = 0 to 14 do
      if mem j cols then
        r := max !r (m.(i).(j) +
                     maximum (i+1) (remove j cols))
    done;
    !r
  end

let answer = maximum 0 (interval 0 14)
```

$$f(i, C) \quad = \quad \max_{j \in C} \; \mathtt{m}[i][j] + f(i+1, C \backslash \{j\})$$

```
let rec maximum i cols =
  if i = 15 then
    0
  else begin
    let r = ref 0 in
    for j = 0 to 14 do
      if cols land (1 lsl j) > 0 then
        r := max !r (m.(i).(j) +
                     maximum (i+1) (cols - (1 lsl j)))
    done;
    !r
  end

let answer = maximum 0 (1 lsl 15 - 1)
```

A Long Computation

goes through all possibilities

there are too many ($15! \approx 1.3 \times 10^{12}$)

A Better Solution

we can easily memoize maximum

```
let table = Hashtbl.create 32749

let rec maximum i cols =

          ... memo (i+1) (cols - (1 lsl j)) ...



and memo i cols =
  try
    Hashtbl.find table (i,cols)
  with Not_found ->
    let res = maximum i cols in
    Hashtbl.add table (i,cols) res;
    res
```

the space is now $2^{16} - 1$

in no time, we find $13\,938 =$

7	53	183	439	863	497	383	563	79	**973**	287	63	343	169	583
627	343	773	959	943	767	473	103	699	303	**957**	703	583	639	913
447	283	463	29	23	487	463	**993**	119	883	327	493	423	159	743
217	623	3	399	**853**	407	103	983	89	463	290	516	212	462	350
960	376	682	**962**	300	780	486	502	912	800	250	346	172	812	350
870	456	192	162	593	473	915	45	989	873	823	965	425	329	803
973	965	905	919	133	673	665	235	509	613	673	815	165	**992**	326
322	148	**972**	962	286	255	941	541	265	323	925	281	601	95	973
445	721	11	525	473	65	511	164	138	672	18	428	154	448	**848**
414	456	310	312	798	104	566	520	302	248	694	**976**	430	392	198
184	829	373	181	631	101	**969**	613	840	740	778	458	284	760	390
821	461	843	513	17	**901**	711	993	293	157	274	94	192	156	574
34	124	4	878	450	476	712	914	838	669	875	299	**823**	329	699
815	559	813	459	522	788	168	586	**966**	232	308	833	251	631	107
813	**883**	451	509	615	77	281	613	459	205	380	274	302	35	805

no easy way to check this answer

let us prove this program correct with Why3

the matrix `m` has size $\texttt{n} \times \texttt{n}$; both `m` and `n` are global

first, we need to agree on a specification

```
axiom n_nonneg: 0 ≤ n
axiom m_pos: ∀ i j: int. 0 ≤ i < n → 0 ≤ j < n → 0 ≤ m[i][j]
```

```
axiom n_nonneg: 0 ≤ n
axiom m_pos: ∀ i j: int. 0 ≤ i < n → 0 ≤ j < n → 0 ≤ m[i][j]
```

$$\sum_{i \leq k < j} \mathtt{m}[k][s[k]]$$

```
axiom n_nonneg: 0 ≤ n
axiom m_pos: ∀ i j: int. 0 ≤ i < n → 0 ≤ j < n → 0 ≤ m[i][j]
```

$$\sum_{i \leq k < j} \mathtt{m}[k][s[k]]$$

```
function sum (map int int) int int : int
axiom sum0:
  ∀ s: map int int, i j: int. j ≤ i →
  sum s i j = 0
axiom sum1:
  ∀ s: map int int, i j : int. i < j →
  sum s i j = m[i][s[i]] + sum s (i+1) j
```

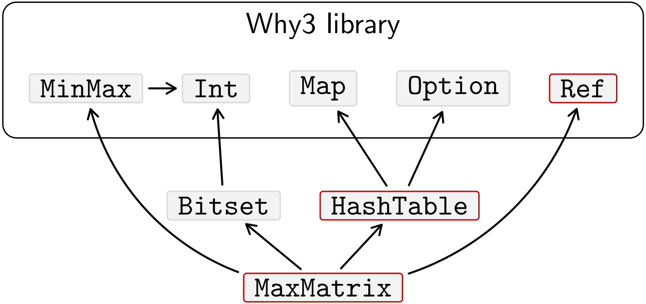
```
axiom n_nonneg: 0 ≤ n
axiom m_pos: ∀ i j: int. 0 ≤ i < n → 0 ≤ j < n → 0 ≤ m[i][j]
```

$$\sum_{i \le k < j} \mathtt{m}[k][s[k]]$$

```
function sum (map int int) int int : int
axiom sum0:
  ∀ s: map int int, i j: int. j ≤ i →
  sum s i j = 0
axiom sum1:
  ∀ s: map int int, i j : int. i < j →
  sum s i j = m[i][s[i]] + sum s (i+1) j
```

```
predicate permutation (s: map int int) =
  (∀ k: int. 0 ≤ k < n → 0 ≤ s[k] < n) ∧
  (∀ k1 k2: int. 0 ≤ k1 < k2 < n → s[k1] ≠ s[k2])

let answer () =
  ...
  { (∃ s: map int int. permutation s ∧ result = sum s 0 n) ∧
    (∀ s: map int int. permutation s → result ≥ sum s 0 n) }
```


Why3 library
MinMax
Int
Map
Option
Ref
Bitset
HashTable
MaxMatrix

```
module HashTable
  type t α β model {| mutable contents: map α (option β) |}
  ...
end
```

in the logic: an immutable, record type

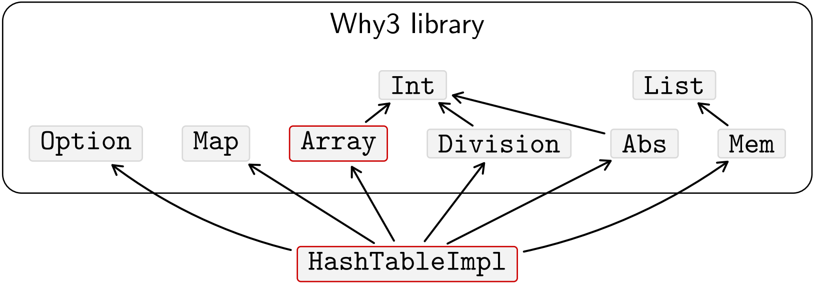
```
module HashTable
  type t α β = {| contents: map α (option β) |}
  ...
end
```

in the programming language: a mutable, abstract type

```
module HashTable
  type t α β
  ...
end
```

```
module HashTable
  ...
  val find (h: t α β) (k: α) :
    {}
    β
    reads h raises Not_found
    { h[k] = Some result } | Not_found → { h[k] = None }
  ...
end
```

$$
\begin{array}{rcl}
\tau & ::= & \alpha \\
 & | & s\ \tau\ \ldots\ \tau \\
 & | & x : \tau \rightarrow \kappa \\
\kappa & ::= & \{f\}\ \tau\ \epsilon\ \{q\} \\
\epsilon & ::= & \text{reads } r, \ldots, r \text{ writes } r, \ldots, r \text{ raises } E, \ldots, E \\
q & ::= & f, E \rightarrow f, \ldots, E \rightarrow f
\end{array}
$$




```
module HashTableImpl
  use import module array.Array
  use import list.List
  use ...
  type t α β = array (list (α, β))

end
```

the data type

```
theory List
  type list α = Nil | Cons α (list α)
end
```

the data type

```
theory List
  type list α = Nil | Cons α (list α)
end
```

can be used in both programs and specifications

```
let rec lookup (k: α) (l: list (α, β)) : β =
  {}
  match l with
  | Nil → raise Not_found
  | Cons (k', v) r → if k = k' then v else lookup k r
  end
  { mem (k, result) l }
  | Not_found → { ∀ v: β. not (mem (k, v) l) }
```

explanation: weakest preconditions commute with pattern matching

```
module Array
 type array α model {| length: int; mutable elts: map int α |}
 ...
```

```
module Array
 type array α model {| length: int; mutable elts: map int α |}
 ...
```

internally, it is a type $\texttt{array}_\rho\ \alpha$ where ρ is a region

effects are sets of regions

$$a : \texttt{array}_\rho\ \alpha \to i : \texttt{int} \to v : \alpha \to \{\dots\}\ \text{writes}\ \rho\ \{\dots\}$$

weakest preconditions rebuild variable values according to region values

$$\sum_{i \leq k < j} \mathtt{m}[k][s[k]]$$

```
function sum (map int int) int int : int
axiom sum0:
  ∀ s: map int int, i j: int. j ≤ i →
  sum s i j = 0
axiom sum1:
  ∀ s: map int int, i j : int. i < j →
  sum s i j = m[i][s[i]] + sum s (i+1) j
```

$$\sum_{i \le k < j} \mathtt{m}[k][s[k]]$$

```
function sum (map int int) int int : int
axiom sum0:
  ∀ s: map int int, i j: int. j ≤ i →
  sum s i j = 0
axiom sum1:
  ∀ s: map int int, i j : int. i < j →
  sum s i j = m[i][s[i]] + sum s (i+1) j
```

```
function f (s: map int int) (i: int) : int =
   m[i][s[i]]

clone import sum.Sum
   with type container = map int int,
        function f = f
```

$$\sum_{i \le k < j} f(k)$$

Why3 Recap

strengths

- rich logic, readily usable in programs
- many provers working together
- modularity, model types, regions

weaknesses

- program and specification tied together
- there are data structures you can't define in Why3
    - e.g. mutable trees, union-find

(yet you can give them signatures)

it is always possible to model the heap, or anything else, and to use Why3 as a verification condition generator

- C (formerly Caduceus, now Frama-C/Jessie)
- Java (Krakatoa)
- CAO, a DSL for cryptographic protocols
  - *Practical realisation and elimination of an ECC-related software bug attack* (Brumley, Barbosa, Page, Vercauteren, Nov 2011)

Part III

Perspectives

We should be able to pick up any algorithm from a standard textbook and prove it to be correct in time and space no larger than the textbook proof.

Verification of Algorithms

currently, only a dream

but it's quickly improving

- recent competitions (VSTTE 2010 2012, FoVeOOS 2011) and benchmarks (VACID-0 2010)
- analogous to the POPLmark challenge

a need for good libraries

obvious candidates for verification: compilers (Leroy 2009), abstract interpretors (Pichardie 2010), theorem provers (Lescuyer 2011), model checkers, slicers, partial evaluators, etc.

arguments in favor of Why3

- algebraic data types
  - to define abstract syntax
- inductive predicates
  - to define semantics
- both automated and interactive proof

several critical parts in verification tools

- type-checking
- weakest preconditions
- logical transformations

currently: (a simplified version of) the Frama-C chain from C to FOL verified in Coq (Herms 2012)

tomorrow: a bootstrapped Why3?

thank you

user-defined variants for loops and recursive functions

- optional
- any type (defaults to type `int`)
- any order relation (defaults to $0 \leq x \wedge y < x$)

```
let rec maximum i c  variant { 2*n - 2*i } =
  ... memo (i+1) (remove j c) ...
with memo i c  variant { 2*n - 2*i + 1 } =
  ... maximum i c ...
```