

Why3

A Multi-Prover Platform for Program Verification

Jean-Christophe Filliâtre
CNRS

joint work with
Andrei Paskevich, Claude Marché, and François Bobot

ProVal team, Orsay, France

IFIP WG 1.9/2.14 “Verified Software”
June 2011



INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

centre de recherche **BACLAY** - ÎLE-DE-FRANCE

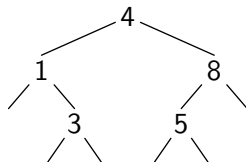
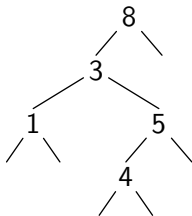


chapter 1

A Tale of Two Programs

Why3 Tutorial: Same Fringe

given two binary trees,
do they contain the same elements when traversed in order?



Same Fringe

a program is contained in a module

```
module SameFringe
```

we introduce an abstract type for elements

```
type elt
```

and an algebraic type for trees

```
type tree =  
  | Empty  
  | Node tree elt tree
```

Same Fringe

the list of elements traversed in order is

```
logic elements (t : tree) : list elt = match t with
  | Empty → Nil
  | Node l x r → elements l ++ Cons x (elements r)
end
```

so the specification is as simple as

```
let same_fringe t1 t2 =
  { }
  ...
  { result=True ↔ elements t1 = elements t2 }
```

Same Fringe: A Solution

a datatype for the left spine of a tree, as a bottom-up list

```
type enum =  
  | Done  
  | Next elt tree enum
```

and the list of its elements

```
logic enum_elements (e : enum) : list elt = match e with  
  | Done → Nil  
  | Next x r e → Cons x (elements r ++ enum_elements e)  
end
```

Same Fringe: A Solution

the left spine of a given tree

```
let rec enum t e =  
  { }  
  match t with  
  | Empty → e  
  | Node l x r → enum l (Next x r e)  
end  
{ enum_elements result = elements t ++ enum_elements e }
```

Same Fringe: A Solution

idea: comparing enums is easier

```
let rec eq_enum e1 e2 =  
  { }  
  match e1, e2 with  
  | Done, Done →  
    True  
  | Next x1 r1 e1, Next x2 r2 e2 →  
    x1 = x2 && eq_enum (enum r1 e1) (enum r2 e2)  
  | _ →  
    False  
end  
{ result=True ↔ enum_elements e1 = enum_elements e2 }
```


Same Fringe: A Solution

and it degenerates into a solution for the same fringe

```
let same_fringe t1 t2 =  
  { }  
  eq_enum (enum t1 Done) (enum t2 Done)  
  { result=True  $\leftrightarrow$  elements t1 = elements t2 }
```

all VCs are proved automatically

Same Fringe: Termination

additionally, we can prove termination of functions `enum` and `eq_enum`, by providing **variants**

```
let rec enum t e variant { leftlen t } =  
  ...
```

```
let rec eq_enum e1 e2 variant { length (enum_elements e1) }  
  ...
```

Why3 Tutorial: Sparse Arrays

from VACID-0

Verification of Ample Correctness of Invariants of Data-structures

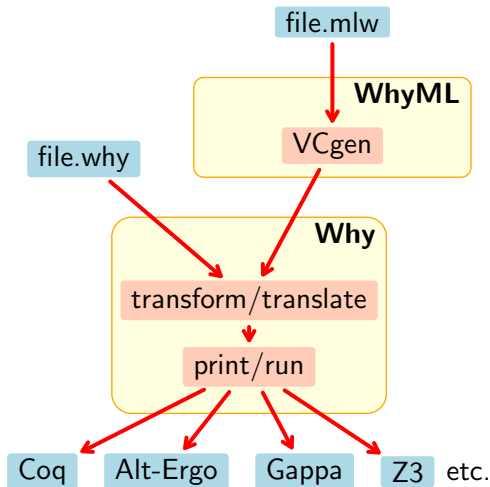
Rustan Leino and Michał Moskal

Sparse Arrays

chapter 2

Big Picture

Where Programs Meet Provers



Why3: Logic

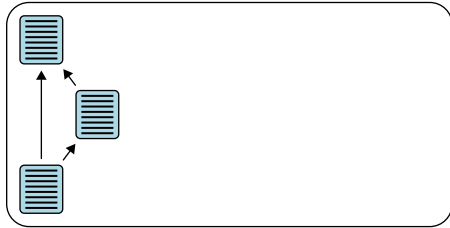
first-order polymorphic logic

- ▶ (mutually) (recursive) algebraic types
- ▶ (mutually) (recursive) functions and predicates
- ▶ (mutually) inductive predicates
- ▶ axioms / lemmas / goals

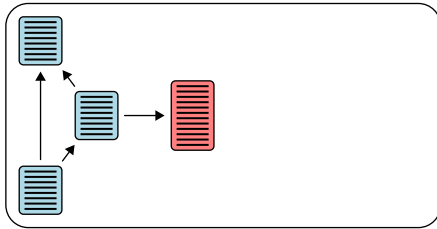
organized in theories

- ▶ a theory may **use** another theory (sharing)
- ▶ a theory may **clone** another theory (copy + substitution)

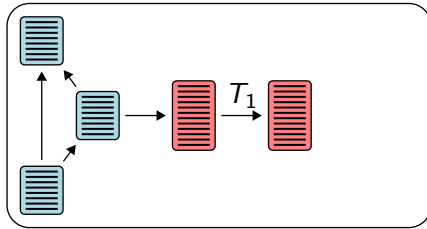
Why3: Architecture



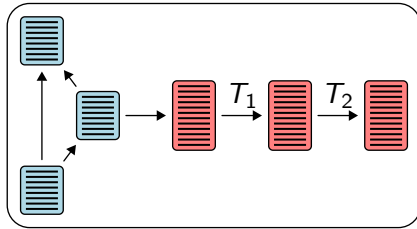
Why3: Architecture



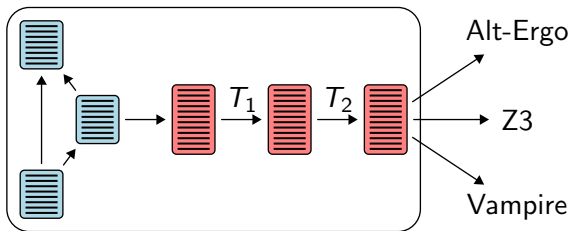
Why3: Architecture



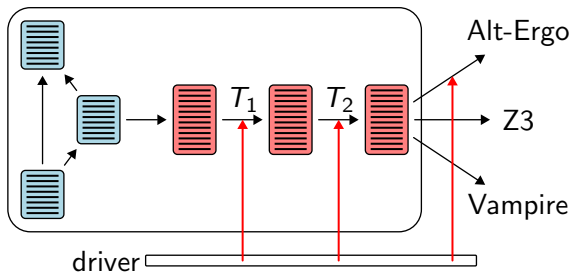
Why3: Architecture



Why3: Architecture



Why3: Architecture



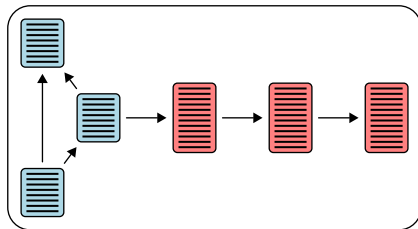
Why3: Prover Drivers

- ▶ transformations to apply
- ▶ output format
 - ▶ pretty-printer
 - ▶ built-in symbols
 - ▶ built-in axioms
- ▶ prover's diagnostic messages

Why3: Ocaml API and Plugins

Your code

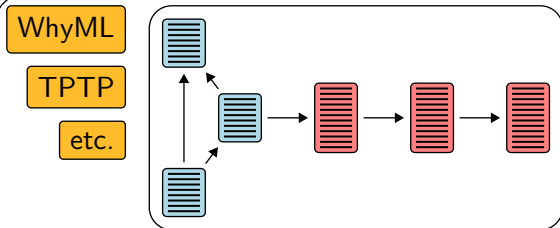
Why3 API



Why3: Ocaml API and Plugins

Your code

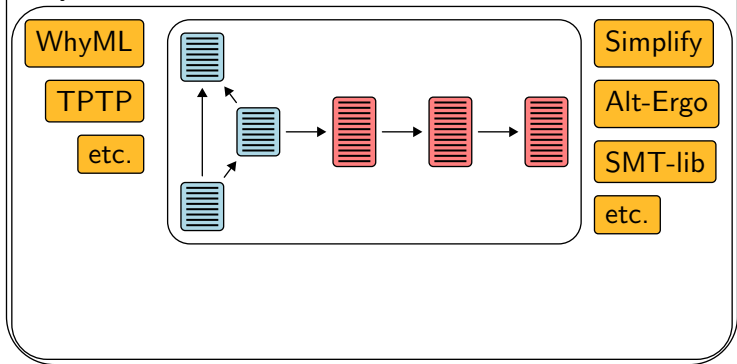
Why3 API



Why3: Ocaml API and Plugins

Your code

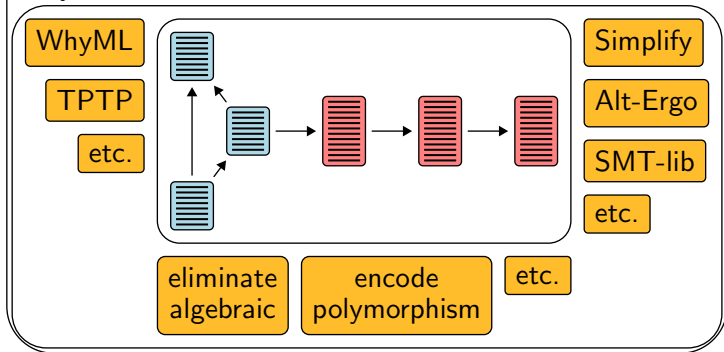
Why3 API



Why3: Ocaml API and Plugins

Your code

Why3 API



Why3: A Programming Language

WhyML

- ▶ first-order ML
 - ▶ purely applicative expressions (logical terms)
 - ▶ effects: mutable data structures, exceptions, non-termination
 - ▶ encapsulation
- ▶ a notion of modules, analogous to theories
- ▶ annotations (pre/post/assert/loop invariants)

appendix A

Cloning Theories

appendix B

The Essence of Hoare Logic