Combining Interactive and Automated Theorem Proving in Why3

Jean-Christophe Filliâtre
CNRS

Automation in Proof Assistants
Tallinn, March 31, 2012

$$f(n) = \begin{cases} n - 10 & \text{if } n > 100, \\ f(f(n+11)) & \text{otherwise.} \end{cases}$$

$$f(n) = \begin{cases} n - 10 & \text{if } n > 100, \\ f(f(n+11)) & \text{otherwise.} \end{cases}$$

when is f returning 91? does f always terminate?

$$f(n) = \begin{cases} n - 10 & \text{if } n > 100, \\ f(f(n+11)) & \text{otherwise.} \end{cases}$$

when is f returning 91? does f always terminate?
is f equivalent to the following code?

```
e ← 1
while e > 0 do
  if n > 100 then
    n ← n − 10
    e ← e − 1
  else
    n ← n + 11
    e ← e + 1
return n
```

an overview of Why3

- a tool for deductive program verification

$$\textit{program} + \textit{spec} \longrightarrow \textit{verification conditions} \longrightarrow \textit{proof}$$

- the SMT revolution makes it possible

- a similar tool: Boogie (Microsoft Research)

developed since 2001 at ProVal (LRI / INRIA)

applications:

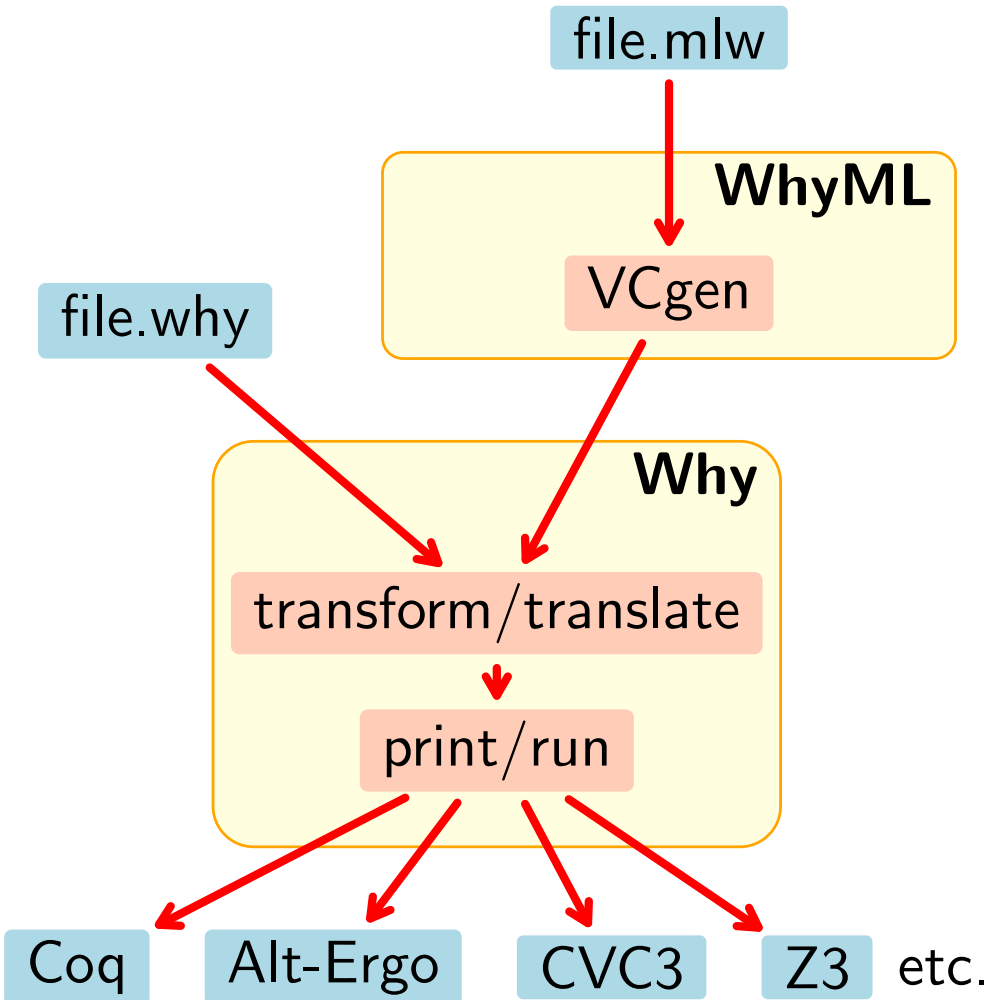
- Java programs: Krakatoa (Marché Paulin Urbain)
- C programs: Caduceus (Filliâtre Marché) formerly, Jessie plug-in of Frama-C (Marché Moy) today
- algorithms
- probabilistic programs (Barthe et al.)
- cryptographic programs (Vieira)

rewritten from scratch, started Feb 2010

authors: F. Bobot, JCF, C. Marché, G. Melquiond, A. Paskevich

open source software (LGPL)

http://why3.lri.fr/


file.mlw
WhyML
VCgen
file.why
Why
transform/translate
print/run
Coq
Alt-Ergo
CVC3
Z3
etc.

the logic of Why3

logic of Why3 = polymorphic first-order logic, with

- (mutually) recursive algebraic data types
- (mutually) recursive function/predicate symboles
- (mutually) inductive predicates
- let-in, match-with, if-then-else

more details:

Expressing Polymorphic Types in a Many-Sorted Language (FroCos 2011)

- types
  - abstract: `type t`
  - alias: `type t = list int`
  - algebraic: `type list α = Nil | Cons α (list α)`
- function / predicate
  - uninterpreted: `function f int : int`
  - defined: `predicate non_empty (l: list α) = l ≠ Nil`
- inductive predicate
  - `inductive trans t t = ...`
- axiom / lemma / goal
  - `goal G: ∀ x: int. x ≥ 0 → x*x ≥ 0`

logic declarations organized in theories

a theory T_1 can be

- used (`use`) in a theory T_2
- cloned (`clone`) in another theory T_2

logic declarations organized in theories

a theory T_1 can be

- used (`use`) in a theory T_2
  - symbols of T_1 are shared
  - axioms of T_1 remain axioms
  - lemmas of T_1 become axioms
  - goals of T_1 are ignored
- cloned (`clone`) in another theory T_2

logic declarations organized in theories

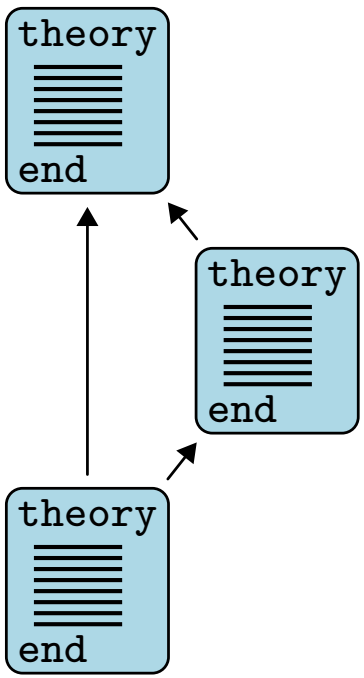
a theory T_1 can be

- used (`use`) in a theory T_2
- cloned (`clone`) in another theory T_2
  - declarations of T_1 are copied or substituted
  - axioms of T_1 remain axioms or become lemmas/goals
  - lemmas of T_1 become axioms
  - goals of T_1 are ignored

a technology to talk to provers

central concept: task

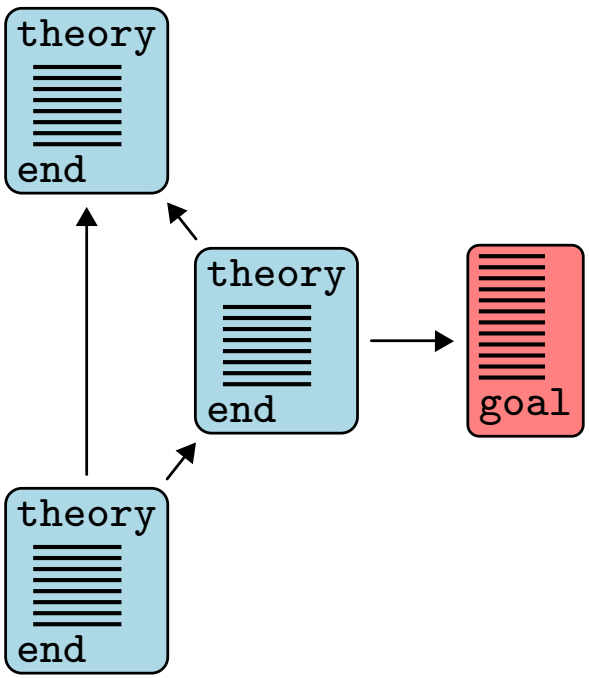
- a context (a list of declarations)
- a goal (a formula)



Alt-Ergo

Z3

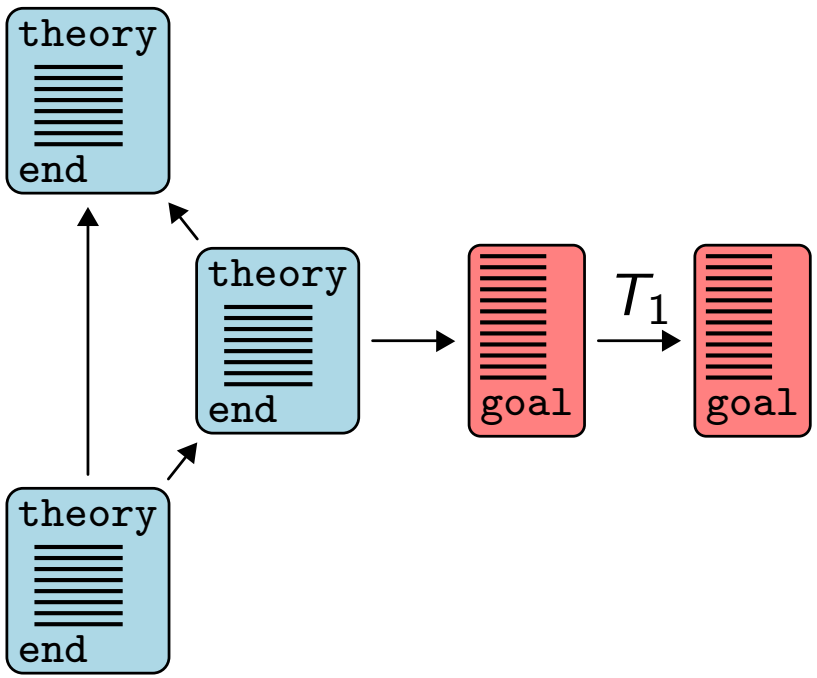
Vampire



Alt-Ergo

Z3

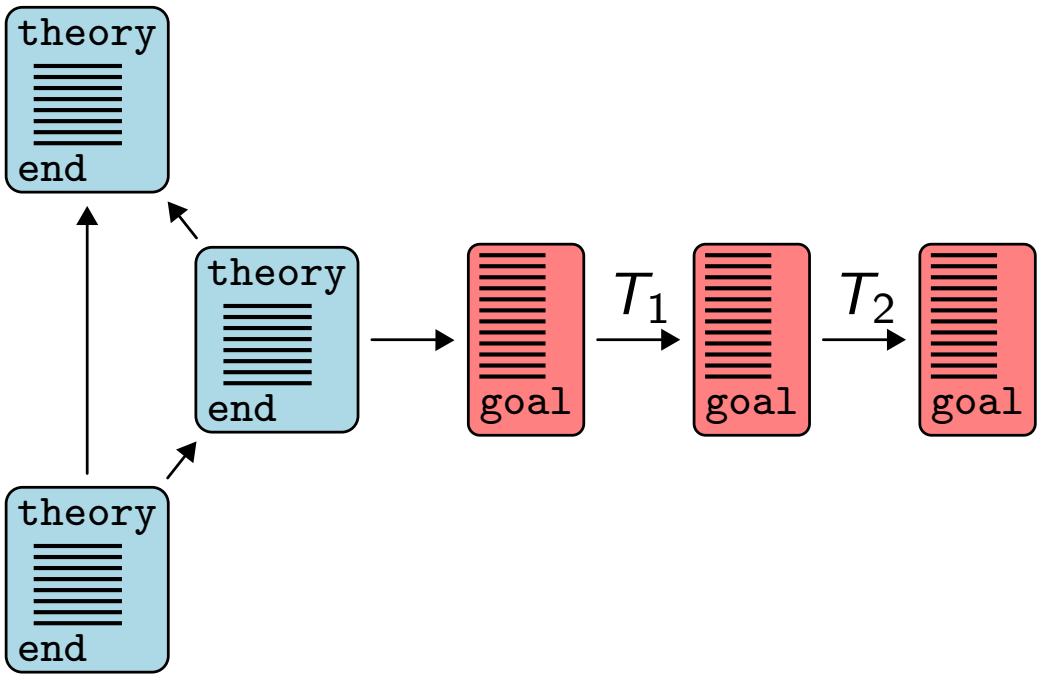
Vampire



Alt-Ergo

Z3

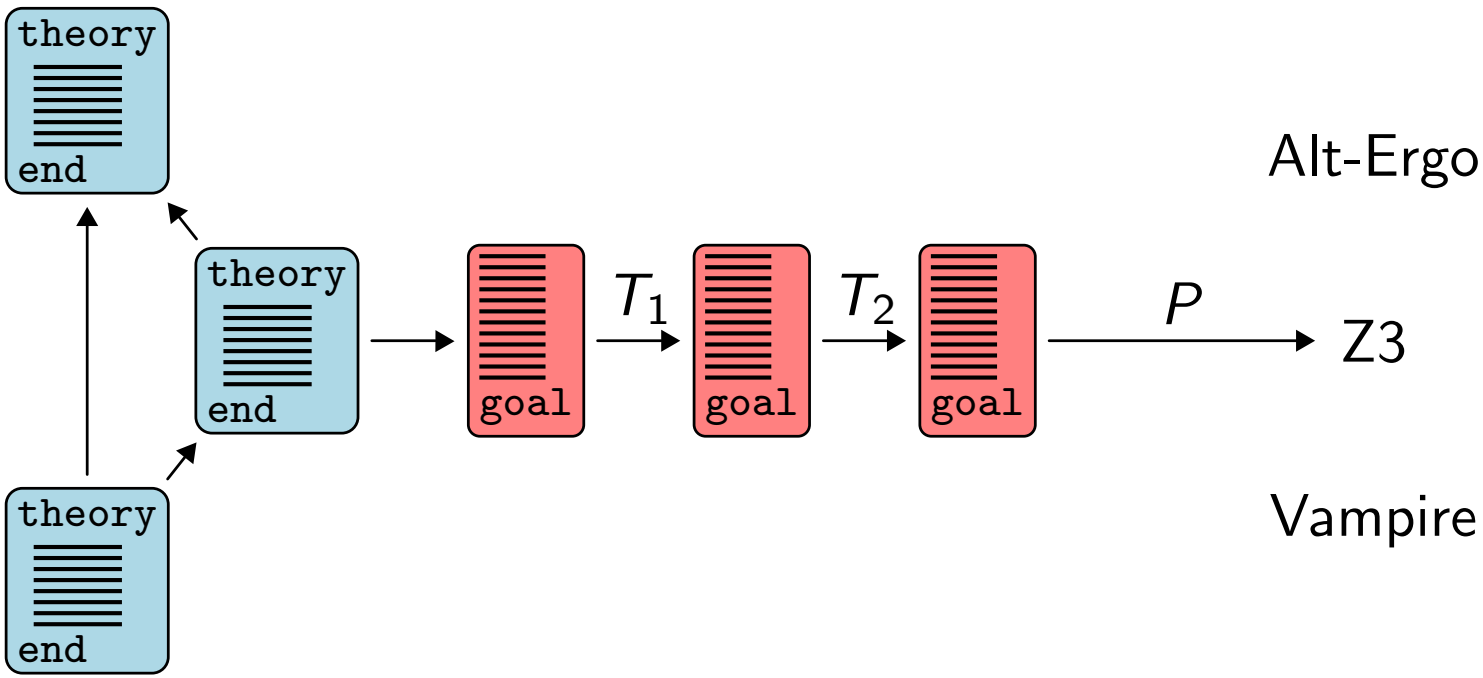
Vampire



Alt-Ergo

Z3

Vampire


theory
end
theory
end
theory
end
goal
goal
goal
T_1
T_2
P
Alt-Ergo
Z3
Vampire

- eliminate algebraic data types and match-with
- eliminate inductive predicates
- eliminate if-then-else, let-in
- encode polymorphism, encode types
- etc.

efficient: results of transformations are memoized

a task journey is driven by a file

- transformations to apply
- prover's input format
  - syntax
  - predefined symbols / axioms
- prover's diagnostic messages

more details: *Why3: Shepherd your herd of provers* (Boogie 2011)

Why3 has an OCaml API

- to build terms, declarations, theories, tasks
- to call provers

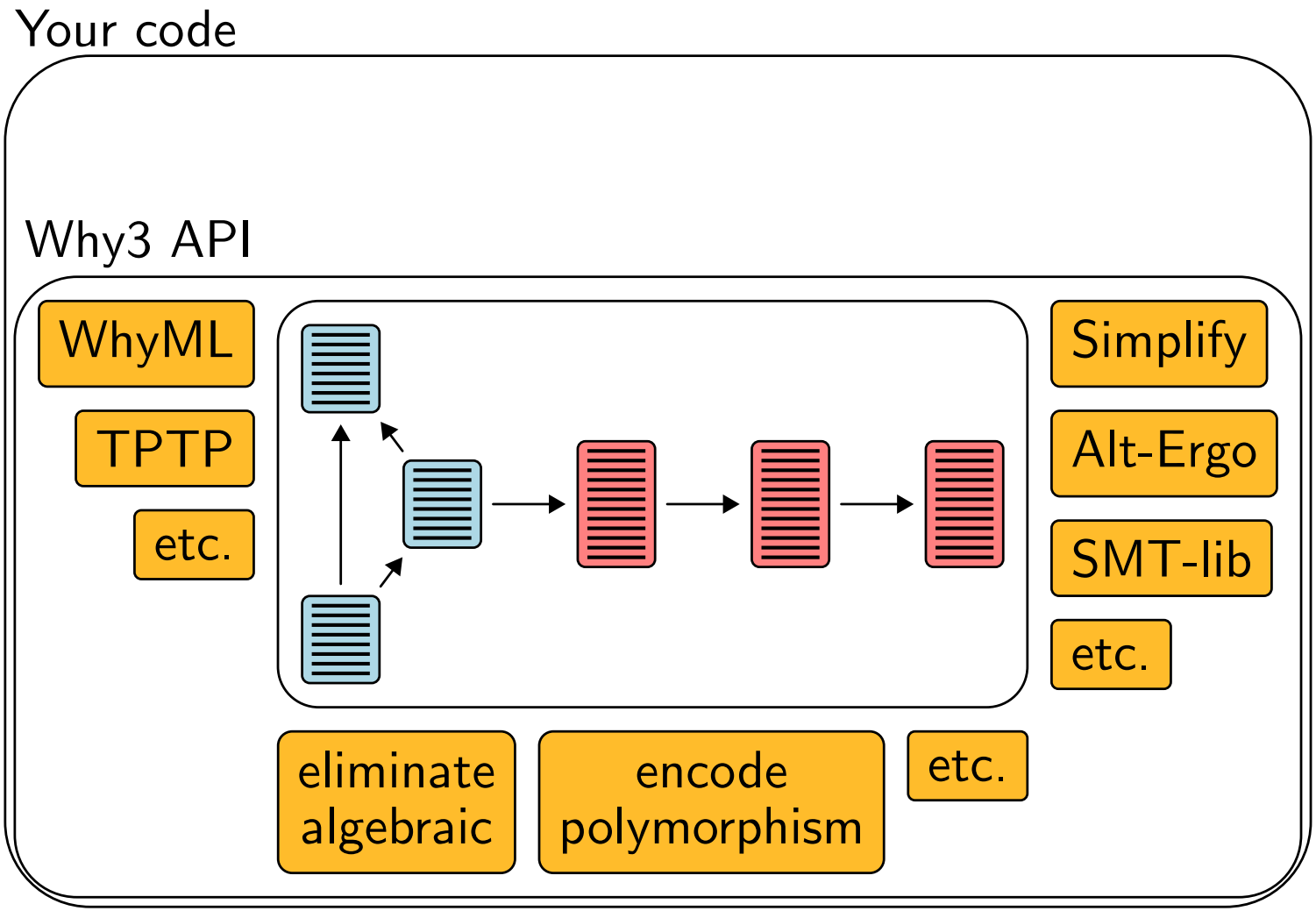
defensive API

- well-typed terms
- well-formed declarations, theories, and tasks

Why3 can be extended via three kinds of plug-ins

- parsers (new input formats)
- transformations (to be used in drivers)
- printers (to add support for new provers)

API and plug-ins



Why3 and proof assistants

similar to ATPs

- driver (transformations, printer, etc.)

with a few differences

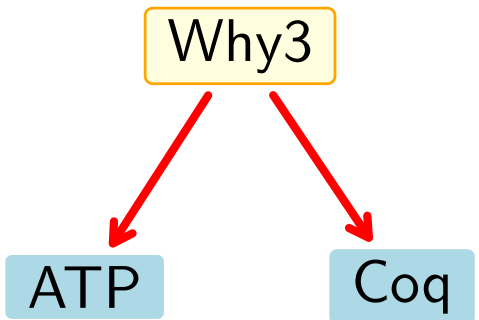
- distinction between editor/checker
- proof scripts saved in proof session

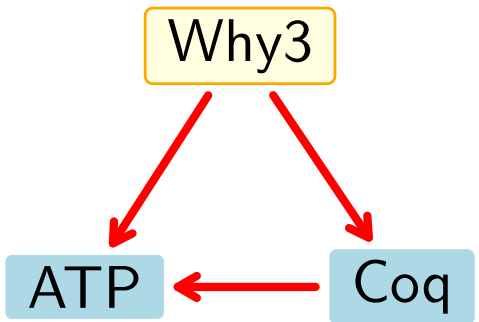
currently, support for

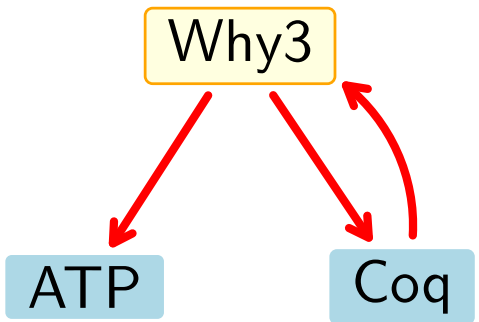
- Coq
- PVS (work in progress)

a proof assistant can be used to

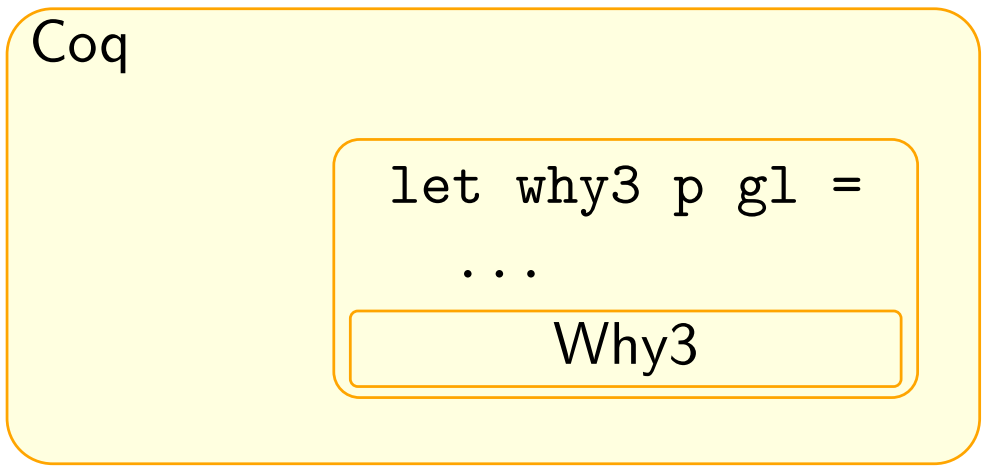
- handle a VC that is not proved automatically
  - to perform a diagnosis
  - to discharge it
- prove that a Why3 theory is consistent
  - provide definitions for uninterpreted symbols
  - prove axioms


Why3
ATP
Coq


Why3
ATP
Coq


Why3
ATP
Coq

Coq has OCaml plug-ins, Why3 provides an OCaml API



the tactic builds a task, then calls prover `p`

currently translated

- polymorphism
- algebraic data types
- inductive predicates
- arithmetic (currently `Z`)

not translated

- higher-order
- dependent types

what cannot be translated is left uninterpreted or discarded

minimum requirement: anything coming from Why3 is sent back

the plug-in translates

- local hypotheses from the goal
- global declarations from the current module

translation is performed lazily
results are memoized (as well as dependencies)

significant differences w.r.t. Isabelle's Sledgehammer

- no higher-order
- ATP used as oracles (no proof reconstruction)
- a different encoding of types (FroCos 2011)
- theories: arithmetic, arrays, etc.

conclusion

Why3 implements

- a polymorphic first-order logic,
  with algebraic data types and inductive predicates
- deductive program verification

Why3 provides

- a technology to talk to provers
- an OCaml API

- a Coq plug-in to call ATPs as oracles

open-source software `http://why3.lri.fr/`

thank you