Faites des paquets de preuves avec Why3 !

Loïc Correnson
CEA-LIST, Laboratoire de Sûreté Logicielle

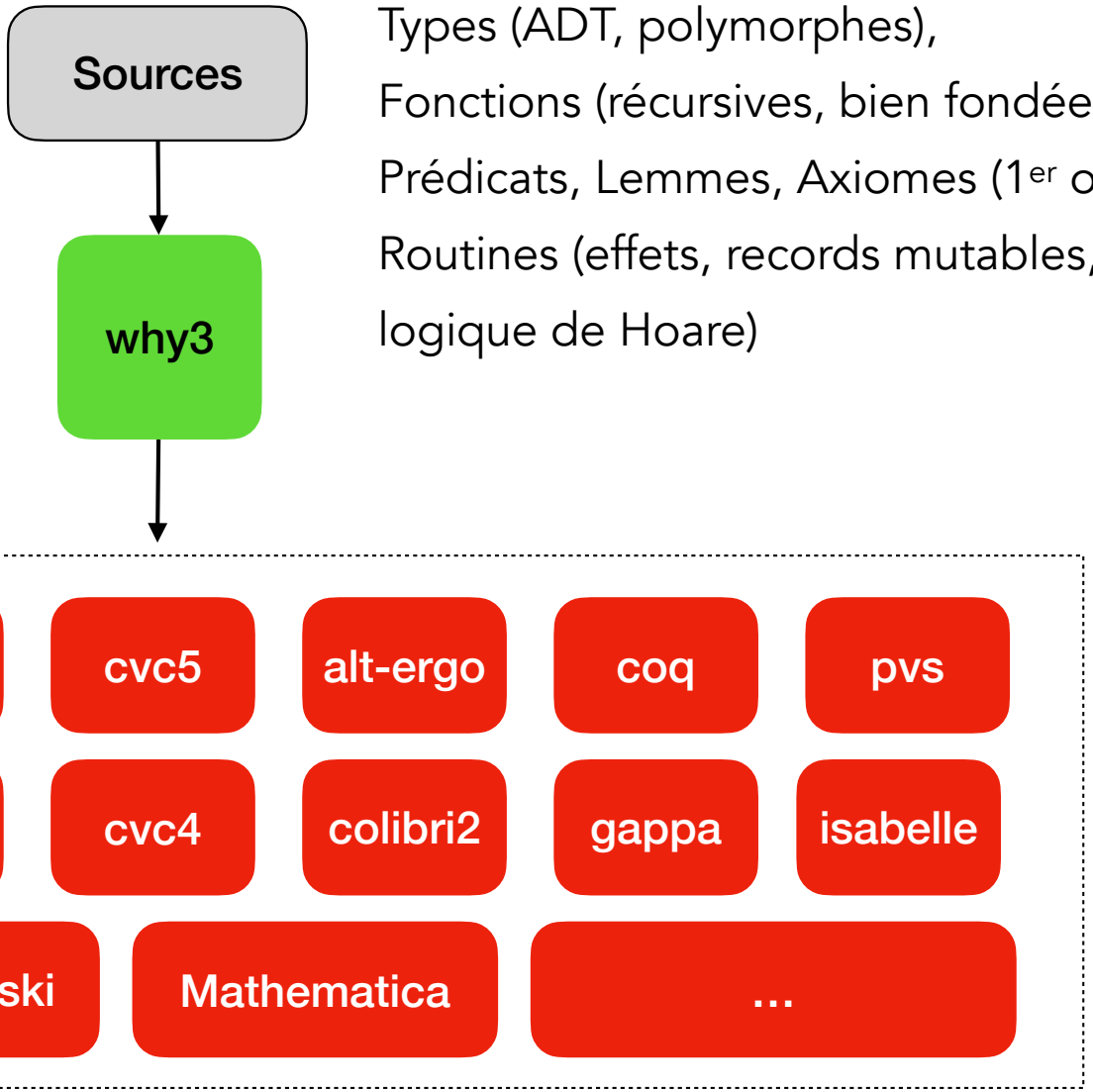
JFLA 2024

Why3

Faut-il le présenter ?

//why3.org

Why3
Where Programs Meet Provers



Types (ADT, polymorphes),

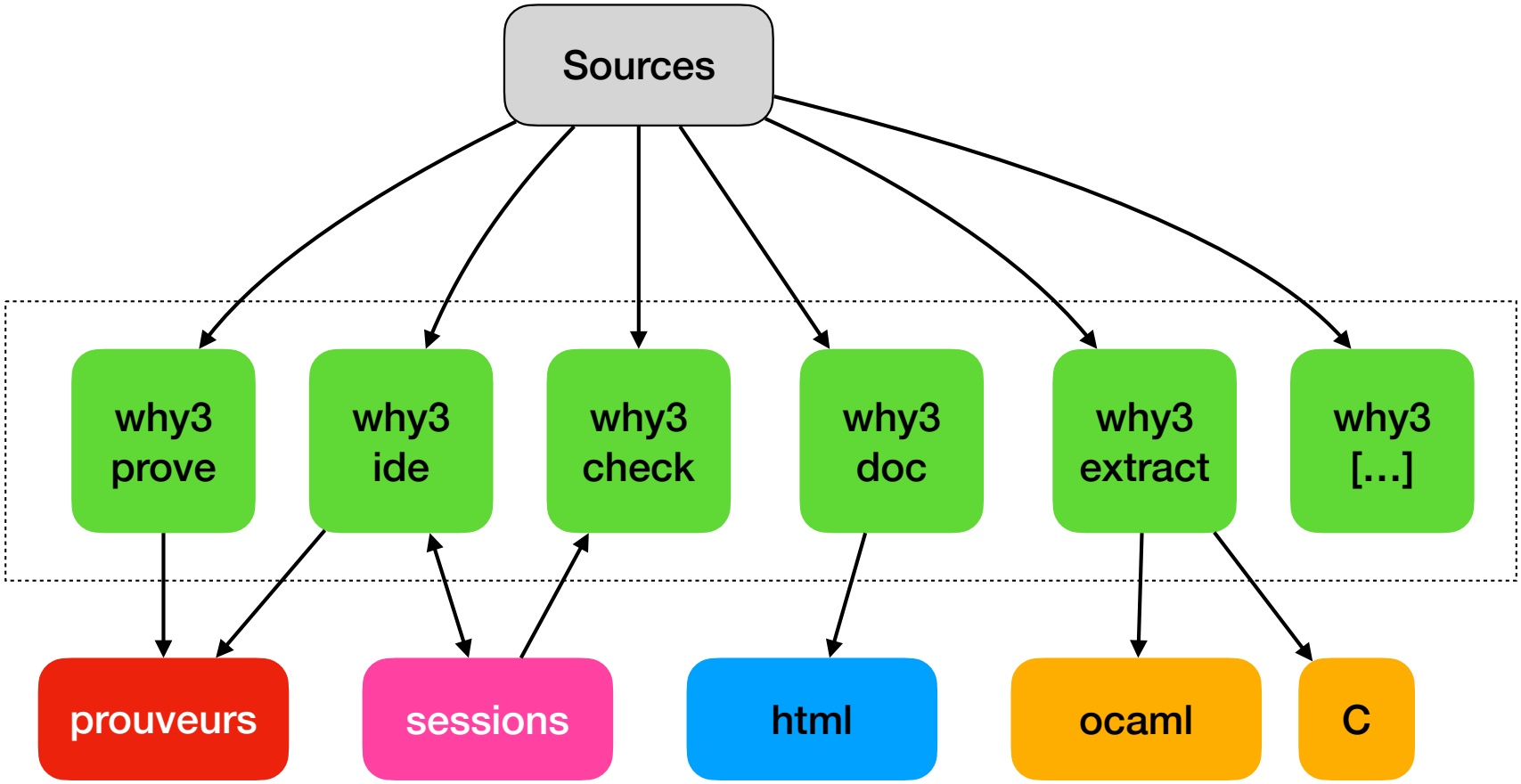
Fonctions (récursives, bien fondées)

Prédicats, Lemmes, Axiomes (1er ordre, classique)

Routines (effets, records mutables, exceptions, logique de Hoare)

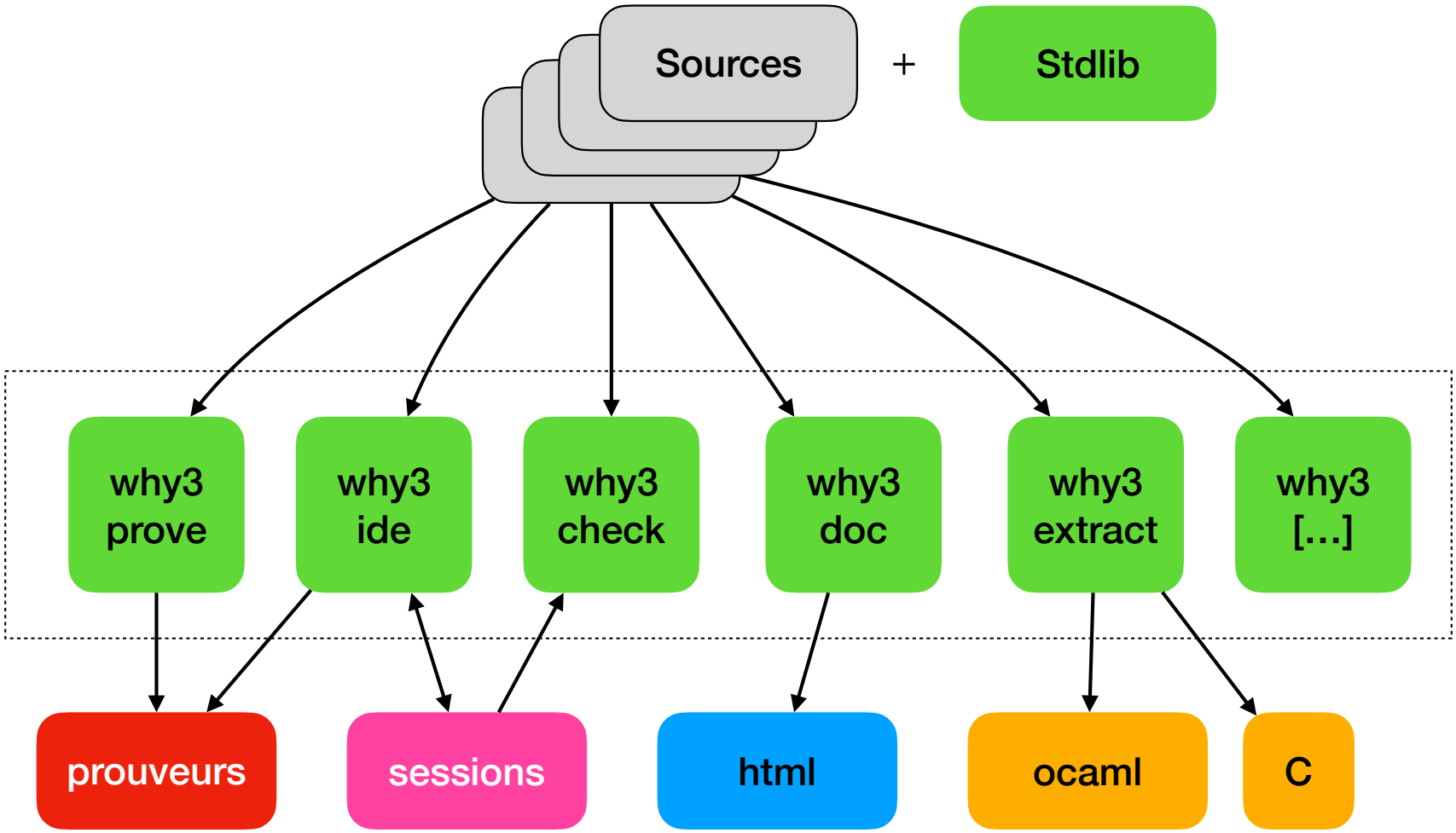
//why3.org

Why3
Where Programs Meet Provers



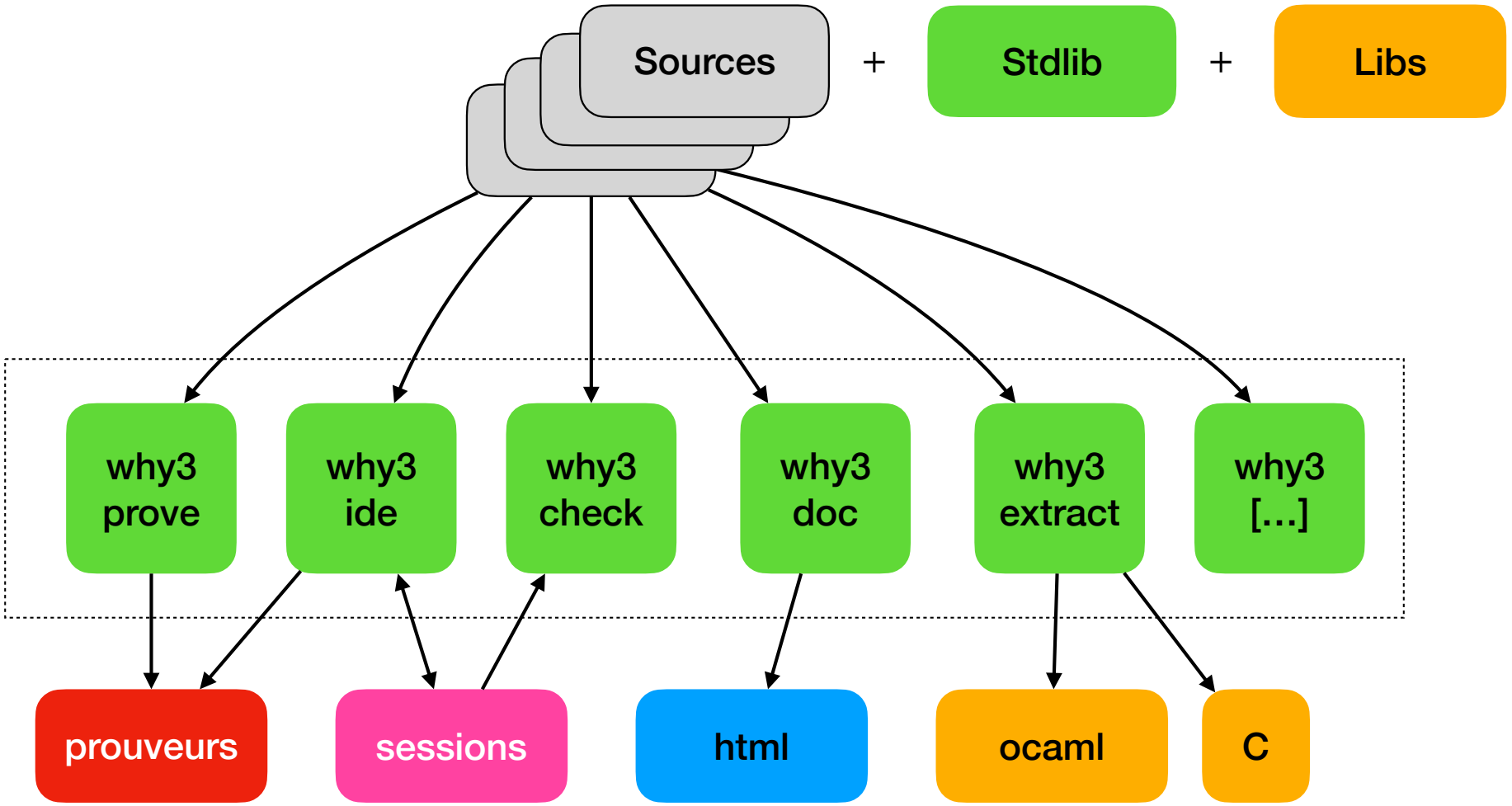
//why3.org

Why3
Where Programs Meet Provers



//why3.org

Why3
Where Programs Meet Provers

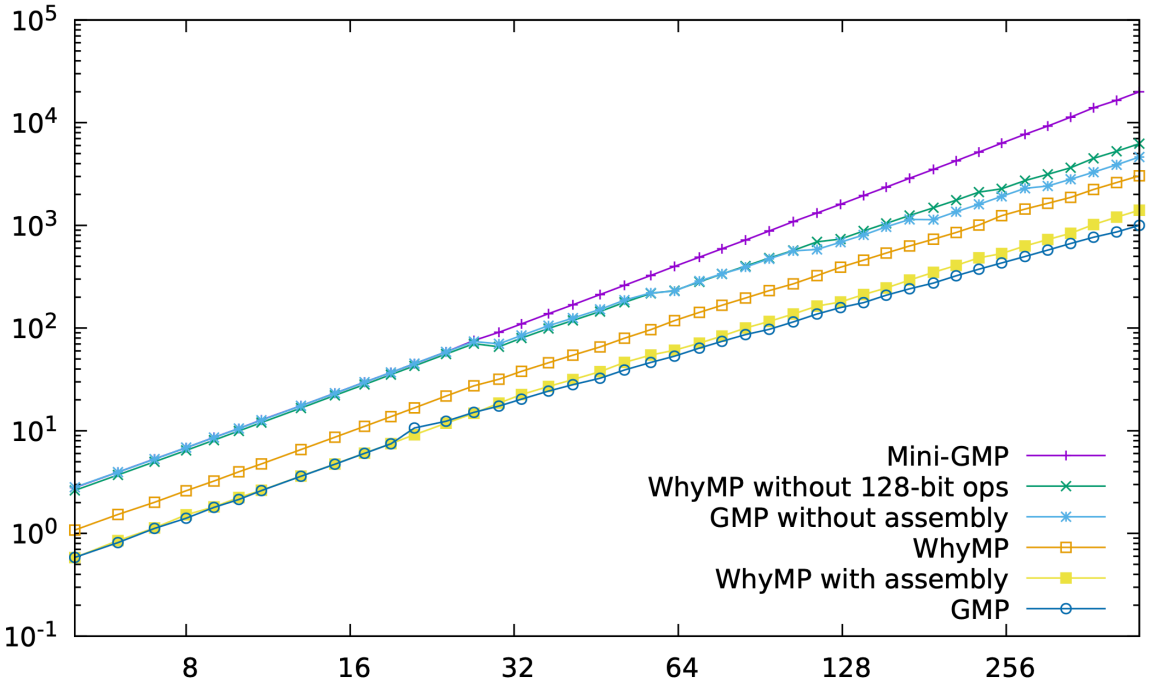


Plein de preuves !

A Why3 proof of GMP algorithms

Raphaël Rieu-Helft
TrustInSoft
Inria

Large-integer arithmetic algorithms are used in contexts where both their performance and their correctness are critical, such as cryptographic software. The fastest algorithms are complex enough that formally verifying them is desirable but challenging. We have formally verified a comprehensive arbitrary-precision integer arithmetic library that implements many state-of-the-art algorithms from the GMP library. The algorithms we have verified include addition, subtraction, Toom-Cook multiplication, division and square root. We use the Why3 platform to perform the proof semi-automatically. We obtain an efficient and formally verified C library of low-level functions on arbitrary-precision natural integers. This verification covers the functional correctness of the algorithms, as well as the memory safety and absence of arithmetic overflows of their C implementation. Using detailed pseudocode, we present the algorithms that we verified and outline their proofs.



1 SPM Documentation

1.1 Documents Index

1. SPM Documentation (this page)
2. ARG.TOOL Toolset Presentation
3. ARG.SPM Formal Model Presentation
4. ARG.CDS Correspondance with TOE Security Function Requirements (SFR)
5. ARG.FSP Connections to TOE Functional Specifications (TSFI)
6. ARG.PROOF Model Hypotheses Rationale

1.2 References Index

- JCRE JavaCard Runtime Environment
- JCVM JavaCard Virtual Machine
- SFR TOE Security Function Requirements
- TSFI TOE Functional Specifications
- SRS.VM COSMO JavaCard Virtual Machine
- SRS.BIOS COSMO Basic Input/Output System BIOS
- SPM.HYP SPM Hypotheses

1.3 SPM Security Properties

- spm Abstract & Concrete Models Consistency
- jcre JCRE Abstract Model
- security Security Properties
- firewall JCRE Firewall Specifications
- monitor Security Monitor

1.4 COSMO VM Main Modules

- card Card External Interface
- vm Internal VM State
- apdu APDU Interpreter
- bytecode Bytecode Interpreter
- vmarea Memory Area Manager
- vmstack Stack Frame Manager
- vmcard Card Management Entry Points
- bc_access Firewall Active Context Management

1.5 Taint Analysis

- asset Abstract Assets
- taint Generic Taint Analysis

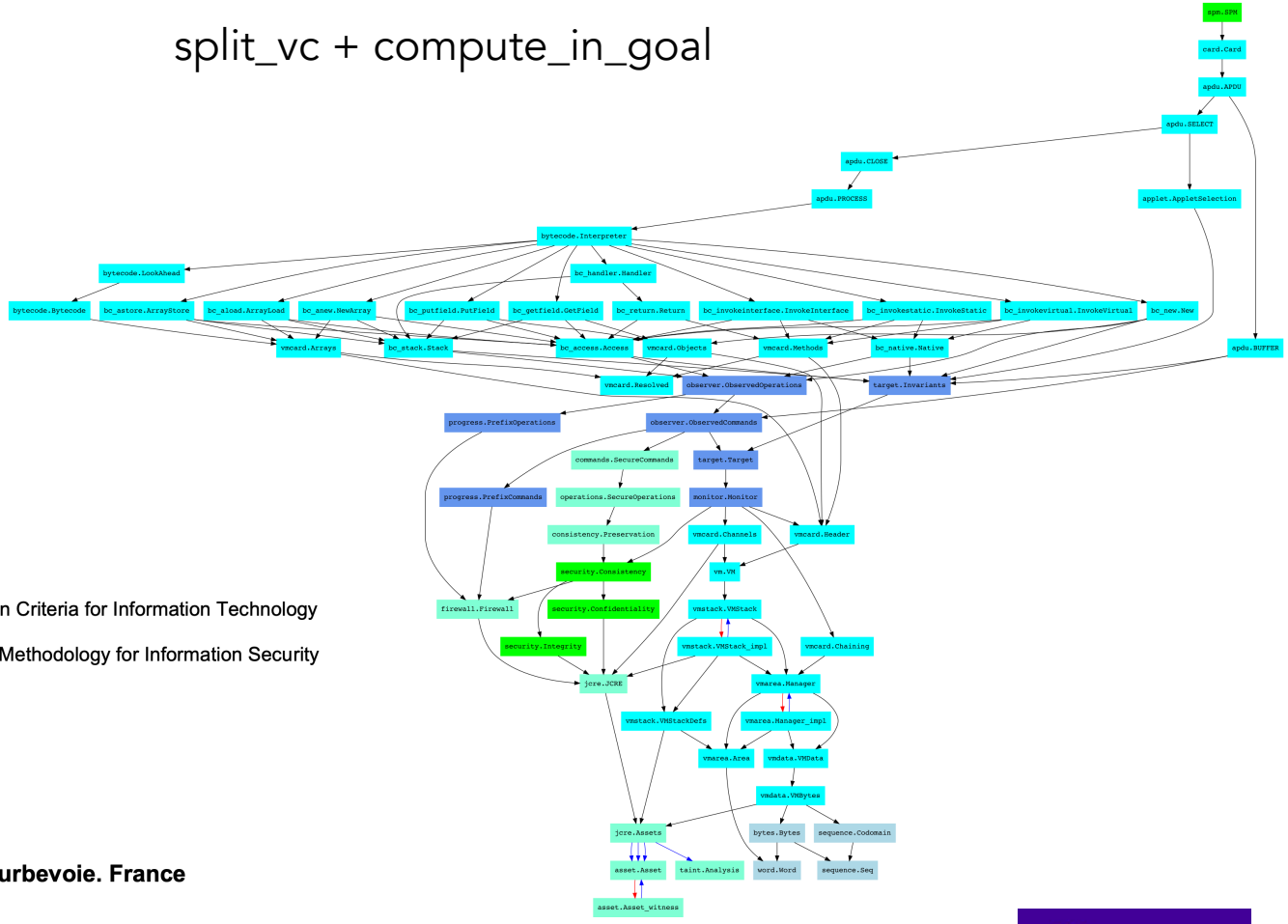
Sources 10 kloc (mlw)

Doc 5 kloc (md)

~2500 Preuves

Alt-Ergo + z3 + cvc4

split_vc + compute_in_goal



TRUSTCB

TRUST AND VERIFY

Certificate

Version 2023-07

Standard	ISO/IEC 15408-1 :2009, -2 :2008, -3 :2008, Common Criteria for Information Technology Security Evaluation (CC) v3.1 rev 5 and ISO/IEC 18045 :2008, Common Criteria Evaluation Methodology for Information Security Evaluation (CEM) v3.1 rev 5
Certificate number	**NSCIB-CC-2300050-01**
	TrustCB B.V. certifies:
Certificate holder and developer	**IDEMIA** **2 place Samuel de Champlain 92400, Courbevoie. France**
Product and assurance level	**ID-One Cosmo X** Assurance Package: ▪ EAL6 augmented with ALC_FLR.1 Protection Profile Conformance: ▪ Java Card System – Open Configuration Protection Profile, Version 3.0.5 December 2017, BSI-CC-PP-0099-2017

IDEMIA

cea list

Passage à l'échelle ?

Questions de génie logiciel :

- Modularité
- Preuves automatiques *vs* interactives
- Puissance de calcul (parallélisation)
- Maintenance (dépendances)
- Collaboration (versions, co-développement)
- Partage (bibliothèque)

Questions de confiance :

- Reproductibilité (configuration, temps de preuve)
- Base de confiance (hypothèses, axiomes)
- Documentation

why3find

faciliter :
le développement,
la vérification et l'audit,
la collaboration et le partage,
pour et avec Why3 📦 !

cea list

why3find

$ opam install why3find

cea list

why3find

README.md

Why3 Library Manager

The `why3find` utility is dedicated to the management of Why3 packages. In short, from `why3find` point of view, a why3 package is a collection of why3 source files and associated documentation and OCaml extracted code that are installed at predefined site(s). It is designed to be fully compatible with Dune and OPAM.

- Why3 Packages
- Package Development
- Package Configuration
- Package Proving
- Package Documentation
- Package Soundness
- OCaml Code Extraction

why3find

```
$ why3find --help
why3find [-h|--help]
why3find [-v|--version]
why3find where
why3find shared
why3find init
why3find list
why3find query [PKG...]
why3find config [OPTIONS] PROVERS
why3find prove [OPTIONS] PATH...
why3find doc [OPTIONS] PATH...
why3find extract [OPTIONS] MODULE...
why3find server OPTIONS
why3find worker OPTIONS
why3find install PKG PATH...
why3find uninstall [PKG...]
why3find compile [-p PKG] FILE
why3find ide [-p PKG] FILE
why3find replay [-p PKG] FILE
why3find CMD [ARGS...]
```

JFLA 2024

A. Preuves
B. Hypothèses
C. Documentation

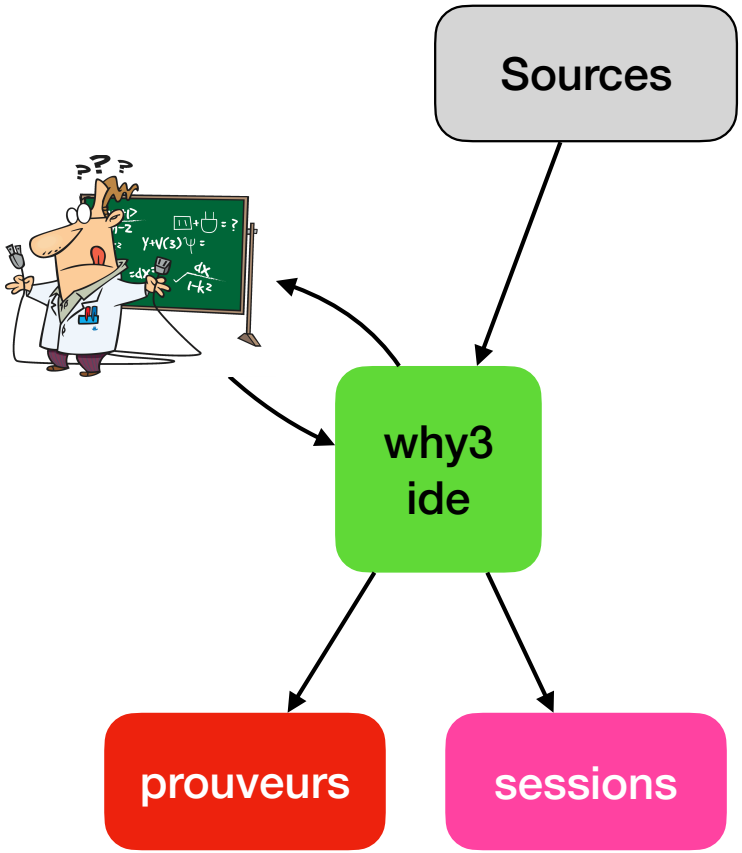
Preuves

Automatisation

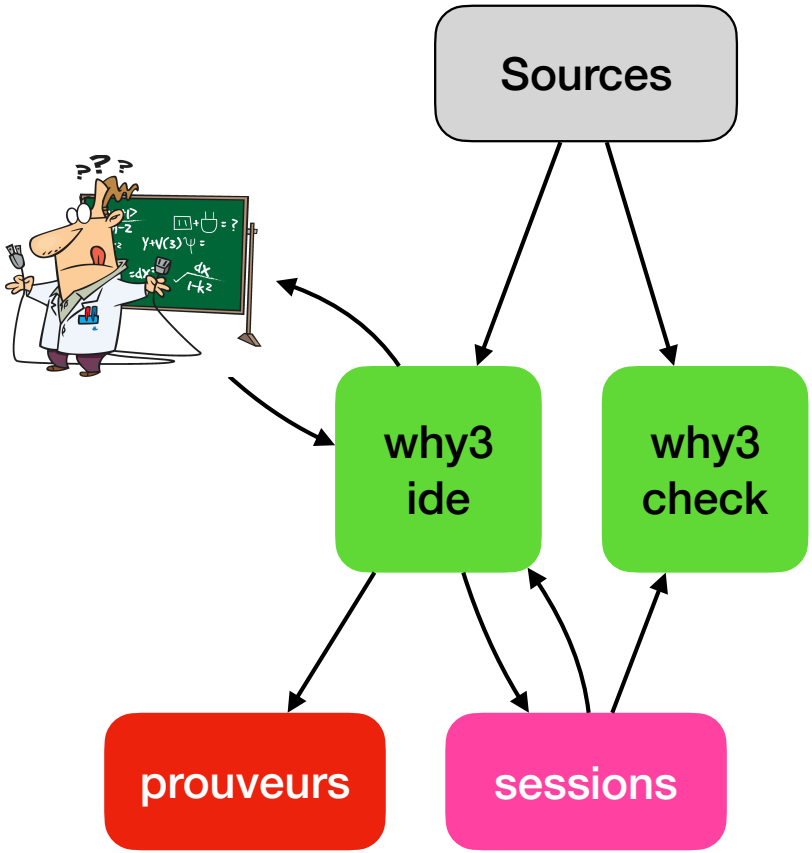
Maintenance

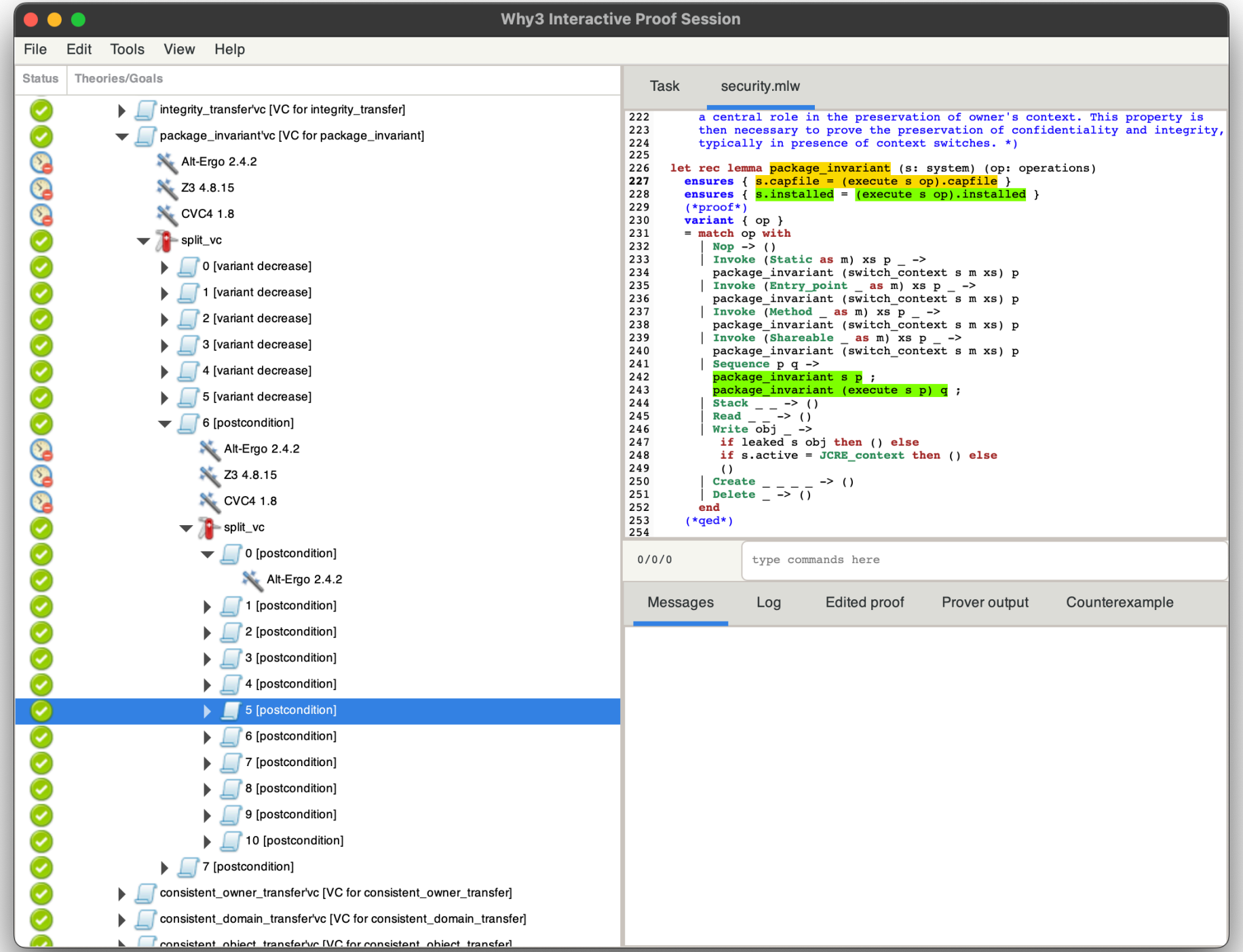
Reproductibilité

Why3 workflow

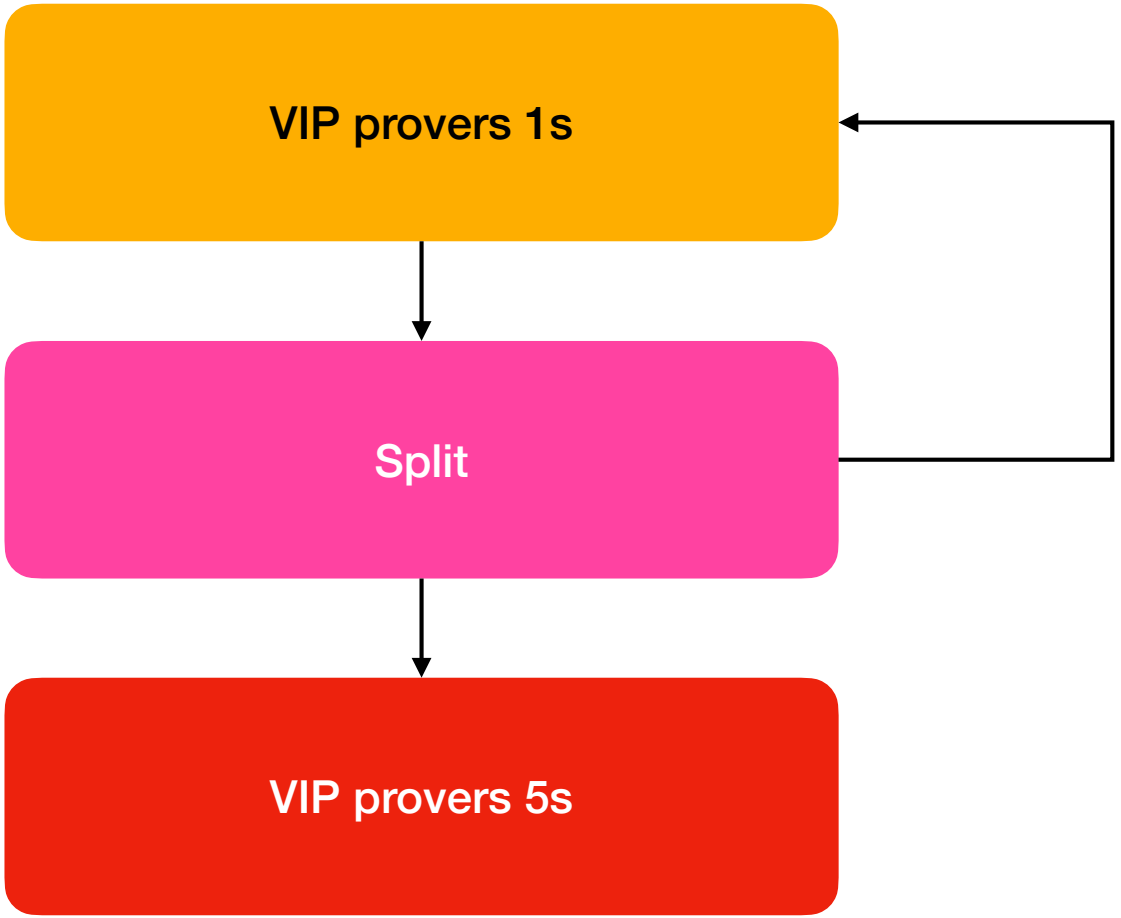


Why3 workflow




Why3 Interactive Proof Session
File
Edit
Tools
View
Help
Status
Theories/Goals
integrity_transfer'vc [VC for integrity_transfer]
package_invariant'vc [VC for package_invariant]
Alt-Ergo 2.4.2
Z3 4.8.15
CVC4 1.8
split_vc
0 [variant decrease]
1 [variant decrease]
2 [variant decrease]
3 [variant decrease]
4 [variant decrease]
5 [variant decrease]
6 [postcondition]
Alt-Ergo 2.4.2
Z3 4.8.15
CVC4 1.8
split_vc
0 [postcondition]
Alt-Ergo 2.4.2
1 [postcondition]
2 [postcondition]
3 [postcondition]
4 [postcondition]
5 [postcondition]
6 [postcondition]
7 [postcondition]
8 [postcondition]
9 [postcondition]
10 [postcondition]
7 [postcondition]
consistent_owner_transfer'vc [VC for consistent_owner_transfer]
consistent_domain_transfer'vc [VC for consistent_domain_transfer]
Task
security.mlw
a central role in the preservation of owner's context. This property is
then necessary to prove the preservation of confidentiality and integrity,
typically in presence of context switches. *)
let rec lemma package_invariant (s: system) (op: operations)
ensures { s.capfile = (execute s op).capfile }
ensures { s.installed = (execute s op).installed }
(*proof*)
variant { op }
= match op with
| Nop -> ()
| Invoke (Static as m) xs p _ ->
package_invariant (switch_context s m xs) p
| Invoke (Entry_point _ as m) xs p _ ->
package_invariant (switch_context s m xs) p
| Invoke (Method _ as m) xs p _ ->
package_invariant (switch_context s m xs) p
| Invoke (Shareable _ as m) xs p _ ->
package_invariant (switch_context s m xs) p
| Sequence p q ->
package_invariant s p ;
package_invariant (execute s p) q ;
| Stack _ _ -> ()
| Read _ _ -> ()
| Write obj _ ->
if leaked s obj then () else
if s.active = JCRE_context then () else
()
| Create _ _ _ _ -> ()
| Delete _ -> ()
end
(*qed*)
0/0/0
type commands here
Messages
Log
Edited proof
Prover output
Counterexample

why3 ide — auto-level 2 & 3



Les preuves à l'épreuve du réel…

Automatisation

- Prouveurs (interactifs, automatiques)
- Transformations (stratégies, transformations)

Maintenance

- Recherche incrémentale
- Cache de prouveurs
- Dépendances

Reproductibilité

- Temps limite de preuve (changement de machine)
- Sessions why3 (machine, volatile, merge)
- Rejeu, audit ?

No More
Interactive Provers !

Go to IDE
for DEBUG only !

Use Split &
Lemma Functions !

Des routines avec des contrats…

```
let wmpn_copyi (r x: ptr uint64) (n: int32) : unit
  requires { valid x n ∧ valid r n }
  ensures  { forall i. 0 ≤ i < n → r[i] = x[i] }
  ensures  { forall i. i < 0 ∨ n ≤ i → r[i] = old r[i] }
= let ref i = 0 in
  while (Int32.(<) i n) do
    variant { n - i }
    invariant { forall j. 0 ≤ j < i → r[j] = x[j] }
    invariant { forall j. j < 0 ∨ i ≤ j → r[j] = old r[j] }
    r[i] ← x[i];
    i ← i+1;
  done
```

Logique de Hoare

let **foo** $x\ y : \tau$

requires $H(x, y)$

returns $r \mapsto P(x, y, r)$

= (* code *)

Logique de Hoare

let **foo** $x\ y : \tau$

requires $H(x, y)$

returns $r \mapsto P(x, y, r)$

= (* code *)

theorem **foo-wp :**

$$\forall x, y\,.\,H(x, y) \rightarrow \text{let}\, r = \text{foo}\, x\, y\, \text{in}\, P(x, y, r)$$

Logique de Hoare & Isormorphisme de Curry-Howard

let **foo** $x\ y : \tau$

requires $H(x, y)$

returns $r \mapsto P(x, y, r)$

= (* code *)

theorem **foo-lemma :**

$$\forall x, y\,.\,H(x, y) \rightarrow \exists r\,.\,P(x, y, r)$$

Lemma Functions

```
let lemma prod_compat_r (a b c:int)
  requires { 0 ≤ a ≤ b }
  requires { 0 ≤ c }
  ensures { c * a ≤ c * b }
= ()
```

$$\forall a, b, c \in \mathbb{Z}, 0 \leq a \leq b \wedge 0 \leq c \implies ca \leq cb$$

```
let lemma prod_compat_lr (a b c d:int)
  requires { 0 ≤ a ≤ b }
  requires { 0 ≤ c ≤ d }
  ensures  { a * c ≤ b * d }
= prod_compat_r c d a;  (* ac ≤ ad *)
  prod_compat_r a b d   (* da ≤ db *)
```

$$\forall a, b, c, d \in \mathbb{Z}, 0 \leq a \leq b \wedge 0 \leq c \leq d \implies ac \leq bd$$

Lemma Functions

```
let lemma fact_div (x y z:int)
  requires { y > 0 }
  ensures  { div (x + y * z) y = (div x y) + z }
=
  assert { div (x + y * z) y = (div x y) + z
           by x + y * z = y * (div (x + y * z) y) + mod (x + y * z) y
           so mod (x + y * z) y = mod (y * z + x) y = mod x y
           so x + y * z = y * (div (x + y * z) y) + mod x y
           so x = y * div x y + mod x y
           so x + y * z = y * div x y + mod x y + y * z
           so y * (div (x + y * z) y) + mod x y
              = y * div x y + mod x y + y * z
           so y * (div (x + y * z) y)
              = y * div x y + y * z
              = y * ((div x y) + z)
           so div (x + y * z) y = div x y + z }
```

Lemma Functions (inductive case)

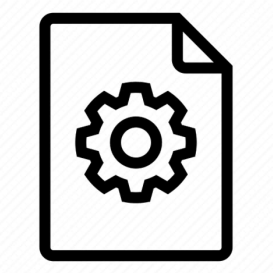
$$\forall n \in \mathbb{Z}, 0 \leq n \implies n \leq 2^n$$

Lemma Functions & preuves par induction

$$\forall n \in \mathbb{Z}, 0 \leq n \implies n \leq 2^n$$

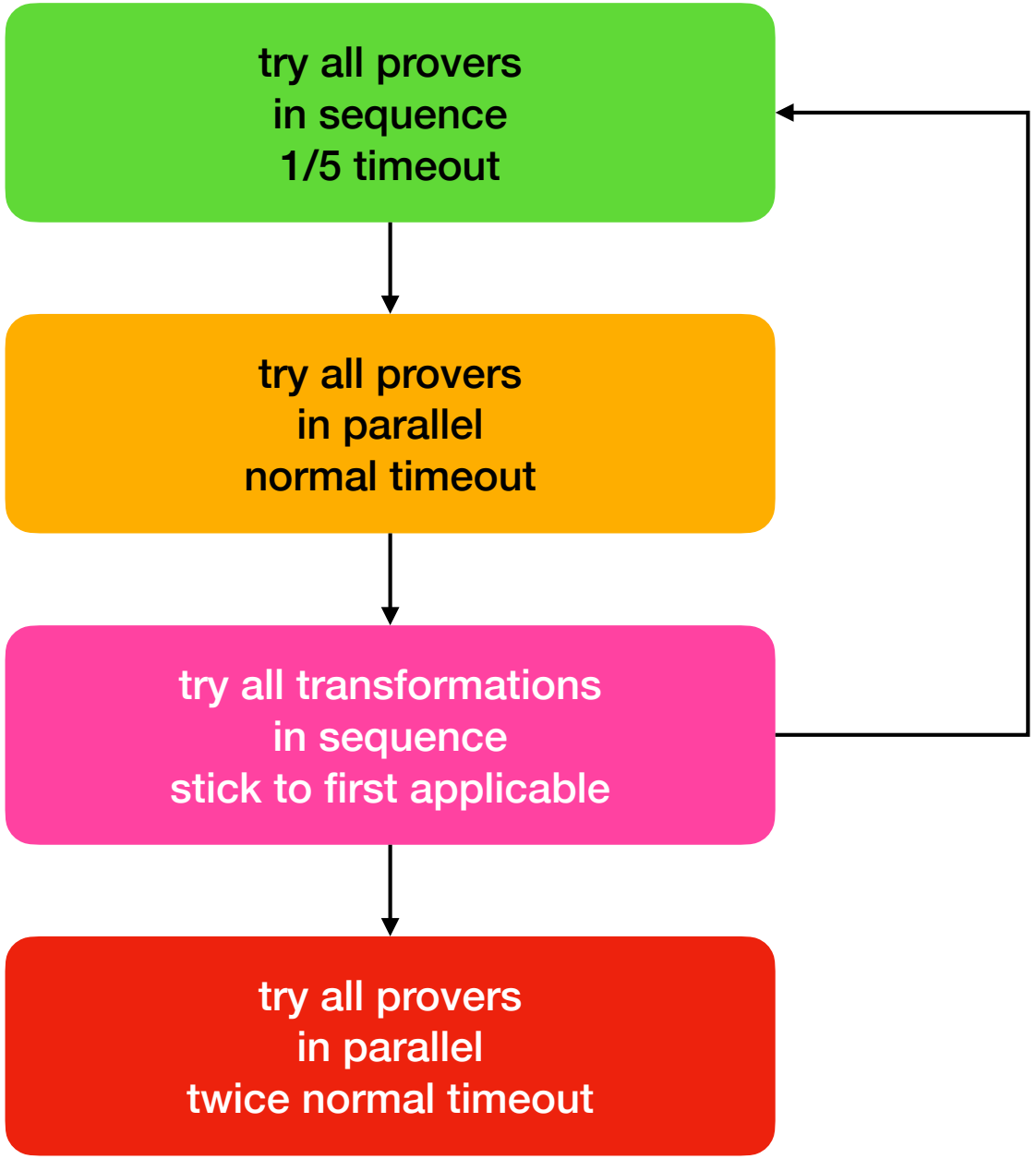
```
let rec lemma pow2_gt (n:int)
  requires { 0 ≤ n }
  ensures { n ≤ power 2 n }
  variant { n }
= if n > 0 then pow2_gt (n-1)
```

why3find proof strategy



why3find config

why3find prove [options]



Calibrage des prouveurs

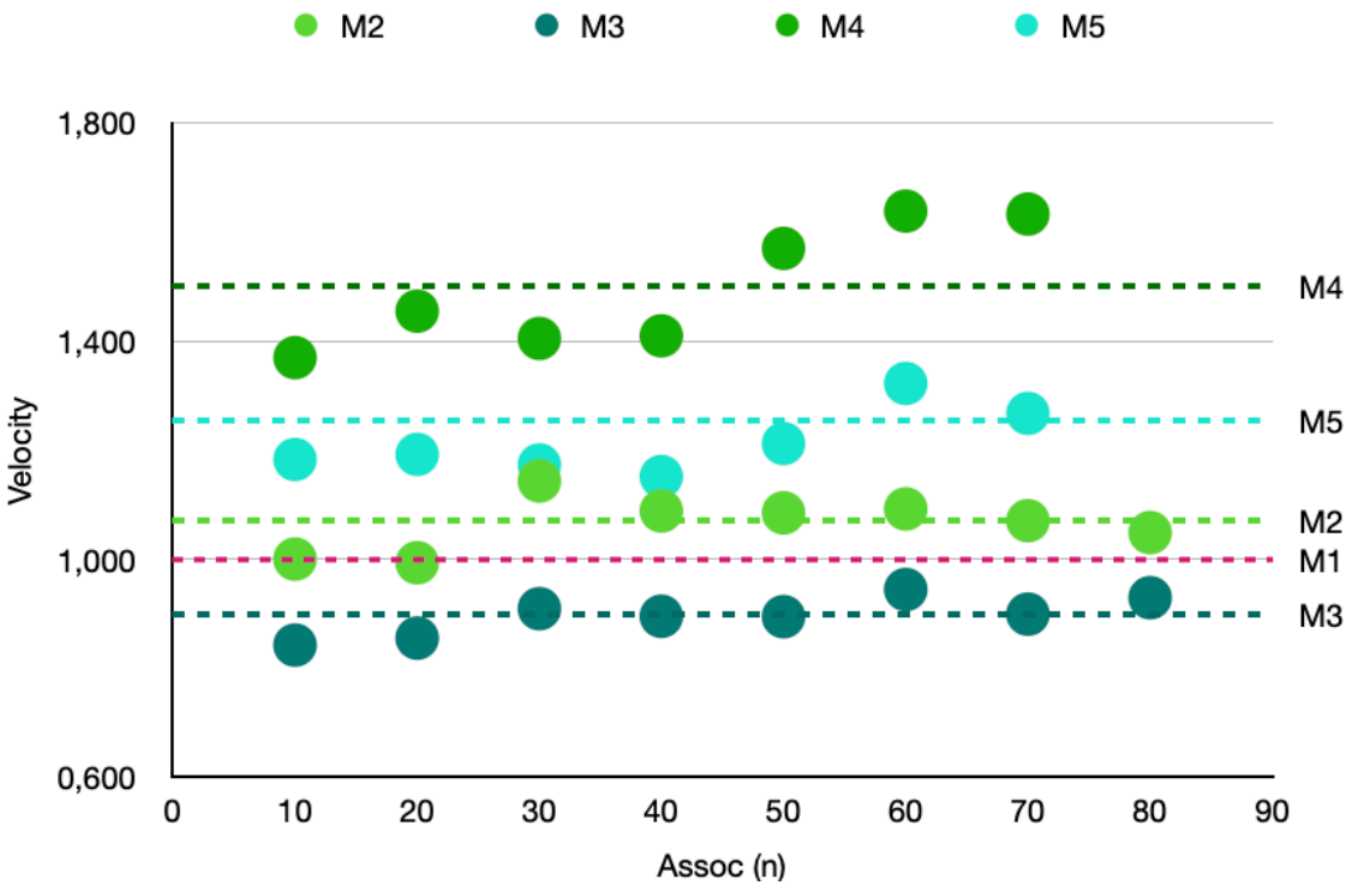


Figure 1. Experimenting the Velocity Law.

Témoins de preuve

$$c ::= \bot \mid \pi : t \mid \tau(c_1, \ldots, c_n)$$

```
$ cat sequence/proof.json
{
  "profile": [
    { "prover": "Alt-Ergo,2.4.2", "size": 17, "time": 0.8175 },
    { "prover": "CVC4,1.8", "size": 40, "time": 0.36 },
    { "prover": "Z3,4.8.15", "size": 34, "time": 0.87 }
  ],
  "proofs": {
    "Seq": {
      "mixfix []": { "prover": "alt-ergo", "time": 0.007 },
      "head": { "prover": "alt-ergo", "time": 0.004 },
      "tail": { "prover": "alt-ergo", "time": 0.004 },
      "reflexivity": { "prover": "alt-ergo", "time": 0.003 },
      "extensivity": { "prover": "alt-ergo", "time": 0.041 },
      "equal": { "prover": "alt-ergo", "time": 0.033 },
      "hd": { "prover": "alt-ergo", "time": 0.004 },
      "tl": { "prover": "alt-ergo", "time": 0.003 },
      "create": { "prover": "alt-ergo", "time": 0.006 },
      "reset": { "prover": "alt-ergo", "time": 0.003 },
      "init": { "prover": "alt-ergo", "time": 0.009000000000000001 },
      "map": { "prover": "alt-ergo", "time": 0.01 },
      "infix ++": { "prover": "alt-ergo", "time": 0.026000000000000002 },
      "mixfix [..]": { "prover": "alt-ergo", "time": 0.038 },
      "mixfix [.._]": { "prover": "alt-ergo", "time": 0.006 },
      "mixfix [_..]": { "prover": "alt-ergo", "time": 0.006 },
      "split": {
        "tactic": "split_vc",
        "children": [
          { "prover": "alt-ergo", "time": 0.02 },
          { "prover": "alt-ergo", "time": 0.004 }
        ]
      },
      "fullsplit": {
        "tactic": "split_vc",
        "children": [
          { "prover": "alt-ergo", "time": 0.032 },
          { "prover": "alt-ergo", "time": 0.01 }
        ]
      },
      "infix +.": { "prover": "alt-ergo", "time": 0.007 },
      "infix .+": { "prover": "alt-ergo", "time": 0.008 },
      "mixfix [<-]": { "prover": "alt-ergo", "time": 0.02 },
      "memcpy": { "prover": "alt-ergo", "time": 0.049 }
    },
    "Codomain": { "codomain": { "prover": "z3", "time": 0.18 } }
  }
}%
```

Tout en ligne de commande !

```
$ why3find prove -i      # debug within IDE
$ why3find prove -f      # force rebuild proof
$ why3find prove -m      # minimize proof tree
$ why3find prove -r      # replay from certificate
```

Hypothèses

Traçabilité

Incohérences

RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

Requirements on the Use of Coq in the Context of Common Criteria Evaluations

French National Cybersecurity Agency (ANSSI)
INRIA

V1.0 23/09/2020

4.1 Axioms and Hypotheses

Coq allows for introducing opaque, well-typed constants which lack a body, that is a term of the expected type. One notable way to achieve this is the `Parameter` vernacular command (or its synonyms, *e.g.*, `Variable`, `Hypothesis`, etc.).

> **Evaluators Rule 4.1.** Starting from Coq 8.5, the `Print Assumptions` command shall be used in order to determine which constants lacking a definition body have been used to prove key theorems.

C’est de la triche !

```
val f () : int
  ensures { result > 0 }
  ensures { result < 0 }
```

Modules & clones

module *A*

$$x_1 : t_1 \ \ldots \ x_n : t_n$$

end

$\texttt{module}\ M$

$\quad \ldots$

$\quad \texttt{clone}\ A\ \texttt{with}\ x_1 = a_1, \ldots, x_n = a_n$

$\quad \ldots$

$\texttt{end}$

Envoyez les clones !

```
module M
  ...
  clone A with x_1 = a_1, ..., x_n = a_n
  ...
end
```

C’est (encore) de la triche !

```
let f () : int
  ensures { result > 0 }
  ensures { result < 0 }
  = assume { false } ; any int
```

Consolidation des Hypothèses

Tout module avec des hypothèses
doit avoir une instance (transitivement) close

La construction « assume » est interdite

Traçabilité dans la documentation

Documentation

Spécifications *vs.* Code *vs.* Preuves

Verified Correctness, Accuracy, and Convergence of a Stationary Iterative Linear Solver: Jacobi Method

Mohit Tekriwal[1], Andrew W. Appel[2], Ariel E. Kellison[3], David Bindel[3], and Jean-Baptiste Jeannin[1]

[1] University of Michigan, USA {tmohit, jeannin}@umich.edu
[2] Princeton University, USA appel@princeton.edu
[3] Cornell University, USA {ak2485, bindel}@cornell.edu

Abstract. Solving a sparse linear system of the form $Ax = b$ is a common engineering task, e.g., as a step in approximating solutions of differential equations. Inverting a large matrix A is often too expensive, and instead engineers rely on iterative methods, which progressively approximate the solution x of the linear system in several iterations, where each iteration is a much less expensive (sparse) matrix-vector multiplication.

We present a formal proof in the Coq proof assistant of the correctness, accuracy and convergence of one prominent iterative method, the Jacobi iteration. The accuracy and convergence properties of Jacobi iteration are

why3find doc

Documentation

- Un peu de style (markdown)
- Structure, pagination (literate)
- Parties cachées (preuves, remarques, etc.)
- Pages de documentation (hors sources)
- Liens inter-paquets

Consolidation

- Témoins de preuve
- Complétude des preuves
- Traçabilité des hypothèses
- Signature des clones
- Clones sans instance close

Conclusion

Faire des paquets de preuves ?

- ➡ Développement
- ➡ Maintenance
- ➡ Reproductibilité
- ➡ Documentation
- ➡ Distribution

Happy Proving !

$ opam install why3find

cea list