SMTCoq: Safe and efficient automation in Coq, and its application to blockchains verification

Valentin Blot and Chantal Keller

Université Paris-Saclay

September, 21st 2020

Main goal of the project: improve automation in Coq

Using Satisfiability Modulo Theories (SMT) solvers:

- automatic solvers
- first-order logic
- combination of built-in theories: linear arithmetic, bit vectors, arrays, ...

	SMT solvers	Coq
Expressivity	First-order logic	**CIC**
Safety	Trust the whole software	**Kernel for proof checking**
Automation	**Automatic proof**	User-guided proof

$\hookrightarrow$ benefit from both worlds

SMTCoq: skeptical approach

Certified ATP:

- prove the correctness of the code of the ATP

\+ once and for all

\+ completeness possible

\- not flexible nor modular; freezes an implementation

\- hard

Certifying ATP:

- the ATP gives certificates that can be checked

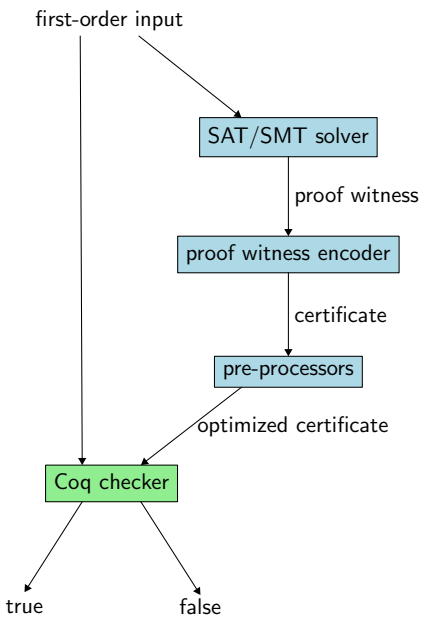
\- certificates to check each time (but efficient)

\- no completeness (or at the meta-level)

\+ very flexible and modular

\+ easier (certified checker)

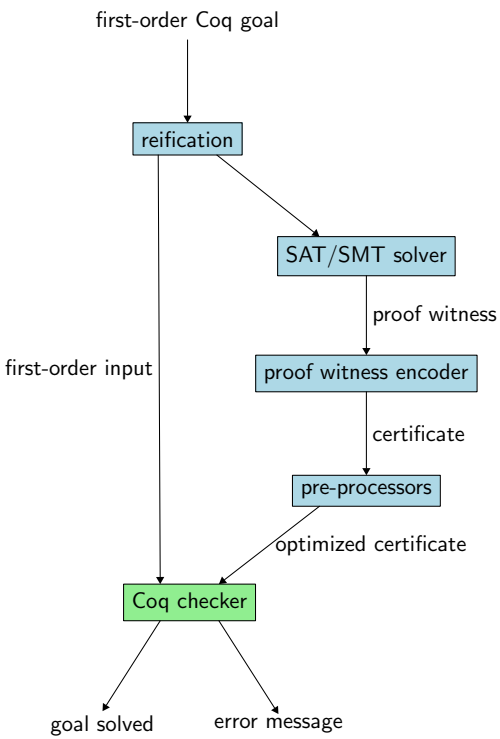
SMTCoq today



A certified checker:

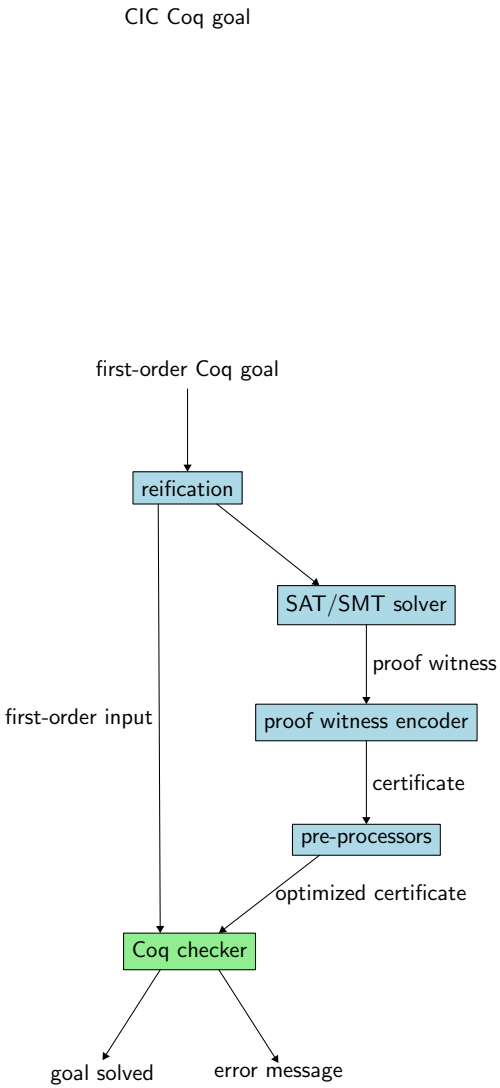
- `checker : formula → certif → bool`
- `correctness` : $\forall \phi$ `c`, `checker` ϕ `c = true` $\rightarrow |\phi|$
- $|\bullet| :$ `formula` $\rightarrow$ `Prop` is an interpretation function (implemented in Coq)

SMTCoq today



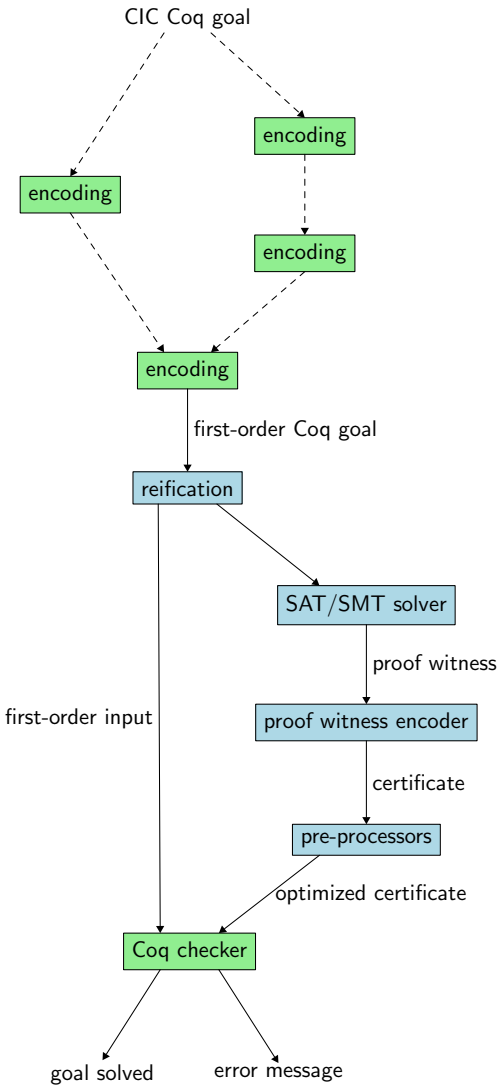
A tactic on top of it

SMTCoq today



Objective: towards CIC

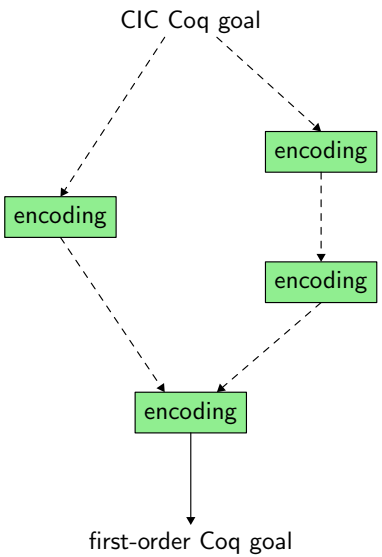
SMTCoq today



Objective: towards CIC

Outline

Framework



Correctness of an encoding:

- `enc` : `formula` $\rightarrow$ `formula`
- `correctness` :
  $\forall \phi, \texttt{valid enc}(\phi) \rightarrow \texttt{valid } \phi$
- encodings can mix autarkic and skeptical approaches
- examples of useful encodings:
  - **monomorphisation**
  - inductive types and predicates
  - defunctionalization
  - dependent types

Validity?

$\forall \phi, \texttt{valid enc}(\phi) \rightarrow \texttt{valid } \phi$

Difficulty: a concrete Coq goal may not be `valid` ϕ

$\hookrightarrow$ Relate `valid` ϕ to the concrete Coq goal, with ϕ a CIC formula

Two approaches:

- \- carefully add an axiom
- \+ state validity in terms of Coq validity

State validity in terms of Coq validity?

Remember correctness of the SMTCoq checker:

- $\texttt{correctness} : \forall \phi_{\mathsf{FO}}\ \texttt{c}, \texttt{checker}\ \phi_{\mathsf{FO}}\ \texttt{c} = \texttt{true} \rightarrow |\phi_{\mathsf{FO}}|$
- $|\bullet| : \texttt{formula} \rightarrow \texttt{Prop}$ is an interpretation function
- reification of a concrete first-order goal g:
  - an external program provides ϕ_{FO}
  - Coq kernel checks $|\phi_{\mathsf{FO}}| \equiv g$

Objective: $\texttt{valid}\ \phi_{\mathsf{CIC}} \triangleq |\phi_{\mathsf{CIC}}|$, for ϕ_{CIC} in (a subset of) CIC

Many applications

Implementation

An internalization of CIC: MetaCoq library

Software quality

- **continuous integration, packaging, good error messages**
- intensive testing
- robustness (caching of certificates)
- follow Coq and provers evolution
- performance evaluation and improvement
- user support

Outline

Context

General automation is "hard"
↪ less incomplete for specific domains

Expertise is needed to formulate automatable goals
↪ provide a surface language

Domains of application

Tezos and NL formalizations

- Mi-Cho-Coq, Albert compiler, Coq Tezos of OCaml, ...
- input welcome

Proof of programs

System verification

Outline

Thanks

`smtcoq.github.io`

Many thanks to Nomadic Labs and Inria

Mais

NON À LA LP(P)R