SMTCoq: A plug-in for integrating SMT solvers into Coq*

Burak Ekici[1], Alain Mebsout[1], Cesare Tinelli[1], Chantal Keller[2], Guy Katz[3], Andrew Reynolds[1], and Clark Barrett[3]

[1] The University of Iowa, USA
[2] LRI, Univ. Paris-Sud, CNRS, Université Paris-Saclay, France
[3] Stanford University, USA

Abstract This paper describes SMTCoq, a plug-in for the integration of external solvers into the Coq proof assistant. Based on a checker for generic first-order proof certificates fully implemented and proved correct in Coq, SMTCoq offers facilities to check answers from external SAT and SMT solvers and to increase Coq's automation using such solvers, all in a safe way. The current version supports proof certificates produced by the SAT solver ZChaff, for propositional logic, and the SMT solvers veriT and CVC4, for the quantifier-free fragment of the combined theory of fixed-size bit vectors, functional arrays with extensionality, linear integer arithmetic, and uninterpreted function symbols.

1 Introduction and Overview

SMTCoq is an open-source plugin for the Coq proof assistant that allows users to dispatch proof goals to external solvers for Boolean satisfiability (SAT) and Satisfiability Modulo Theories (SMT). It is aimed mostly at users doing verification in Coq, and is a sophisticated tool that lets them harness the power of these solvers in a trustworthy way. SMTCoq sends selected Coq (sub)goals to external solvers. When the solvers succeed in proving a goal they are required to return a proof witness, or *certificate*, which is then used by SMTCoq to effectively reconstruct a proof of the goal within Coq in a fully automated way.

SMTCoq's two main objectives are (1) to increase the level of automation in Coq by providing safe tactics for solving a class of Coq goals automatically by means of external solvers, and (2) to increase the confidence in SAT and SMT solvers in general by providing an independent and certified checker for SAT/SMT proof witnesses. SMTCoq has been designed to be modular along two dimensions: supporting new theories, and incorporating new solvers. Correspondingly, its main checker is an extensible combination of several independent small checkers, and its kernel relies on a fairly general certificate format, close to the one proposed by Besson *et al.* [3].

* This work was partially sponsored by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under contracts FA8750-13-2-0241 and FA8750-15-C-0113. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of AFRL or DARPA.

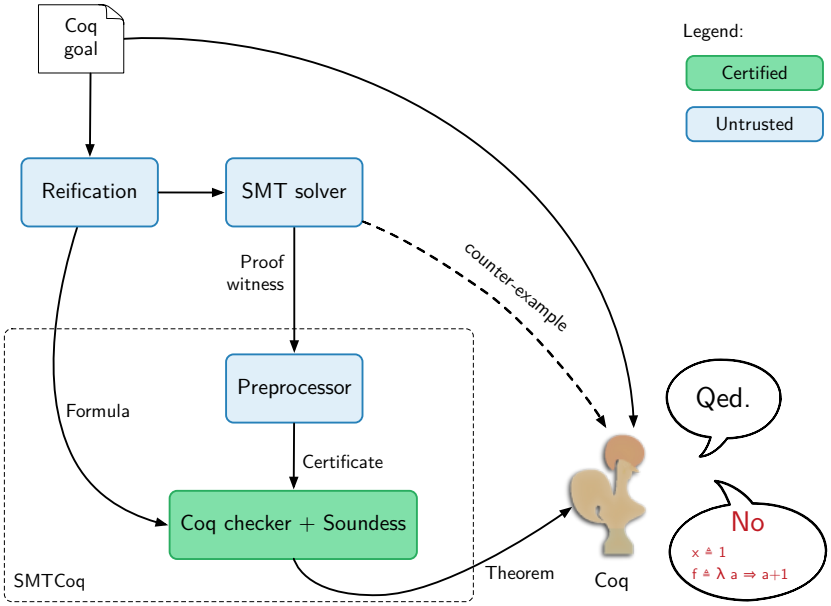

Figure 1: SMTCoq's architecture

An initial implementation of SMTCoq [11] incorporated only the ZChaff SAT solver [12] and the veriT SMT solver [5], supporting for the latter the combined theory of equality over uninterpreted symbols (EUF) and linear integer arithmetic (LIA). *We describe a recent major extension of the tool that incorporates the CVC4 SMT solver [1] and supports, additionally, the theory of arrays and the theory of fixed-sized bit vectors.*[4]

General Architecture The heart of SMTCoq is a checker for SMT proof certificates, in SMTCoq's own format, that is implemented and proved correct inside Coq (see Figure 1). The checker is written to be executable inside Coq. Its Coq signature is `checker : formula → certificate → bool` where `certificate` is the type of proof certificates and `formula` is a type representing SMT formulas. SMTCoq implements a *deep embedding* in Coq of SMT formulas via a function `⟦_⟧ : formula → bool` that interprets each SMT formula as a corresponding Boolean term in Coq. The correctness of the checker is guaranteed by a Coq lemma of the form:

```
Lemma checker_sound : ∀ f c, checker f c = true → ⟦f⟧ = true
```

stating that whenever the checker returns `true` for a given formula `f` and proof certificate `c` for `f`, the interpretation in Coq of `f` is a valid Boolean formula.

Using the type `bool` of Booleans as the codomain of the interpretation function `⟦_⟧` instead of Coq's own type `Prop` of (intuitionistic) propositions allows SMTCoq to

[4] A version incorporating this extension is available at https://smtcoq.github.io/ .

check the classical logic proofs returned by the external solvers without the need to add classical logic axioms to Coq. SMTCoq provides Coq tactics that automatically convert Coq goals (which are terms of type `Prop`) to `bool`, provided they fall in the first-order logic fragment supported by SMTCoq. This conversion relies on the `reflect` predicate from the Coq plugin SSReflect [6].

Use Cases The primary use case for SMTCoq, reflected in Figure 1, aims at increasing automation during interactive proofs in Coq. We have defined a few Coq tactics that can be invoked during an interactive proof session to dispatch goals to an external SMT solver. Currently, these goals must be universal formulas in the combination of supported theories listed earlier. If the SMT solver is able to prove a given goal (by proving that an encoding of its negation is unsatisfiable) and produce a proof certificate, and if SMTCoq's checker succeeds in validating the certificate, then the soundness of the checker yields, by *computational reflection*, a Coq proof of the original goal. In this use case, the trusted base consists only of Coq itself: if something else goes wrong (e.g., the SMT solver fails to prove the goal or SMTCoq's checker fails to validate the certificate), the tactic will fail. However, no unsoundness will be introduced into the system.

A secondary use of SMTCoq, not reflected in Figure 1 for space constraints, is simply as a correct-by-construction independent checker for proof certificates generated by certifying SMT solvers like CVC4 and veriT. In this case, the trusted base is both Coq and SMTCoq's parser for input problems written in the SMT-LIB standard format [2]. The parser must be trusted to guarantee that SMTCoq is effectively verifying a proof of the actual problem that was sent to the external solver—as opposed to a proof of some other formula. However, this parser is fairly straightforward, and so its correctness is easy to verify by other means (e.g., [9]), although that is left for future work.

A crucial aspect of both use cases is the need to *preprocess* proof certificates before sending them to SMTCoq's checker because, for instance, they may not be in the format accepted by the checker—which is the case with CVC4-generated certificates. However, there is no need to trust this preprocessing stage or prove anything about it. If the preprocessor is buggy, SMTCoq will fail to validate the proof certificate and so no unsoundness will be introduced into Coq in this case either. This preprocessing stage allows us to easily extend SMTCoq with new solvers as long as their certificates can be faithfully converted to the SMTCoq format. SMTCoq's current support for ZChaff, veriT and CVC4 is indeed provided through a preprocessor for each solver. Using a preprocessor is also beneficial for efficiency: proof certificates may be further compacted or simplified before or after their translation, thus potentially improving performance and scalability. We do all that in the extension described next.

2 The CVC4 extension

For this work, we leveraged the proof-producing capabilities of the CVC4 SMT solver [7, 10]. CVC4 uses the LFSC proof format [14] also used by its predecessor CVC3 [13]. LFSC is appealing because of its flexibility, generality, and ability to represent fine-grained proofs. To increase compactness and scalability, LFSC extends the Edinburgh Logical Framework (LF) [8] by allowing types with computational *side conditions*,

explicit checks defined as programs in a simple functional programming language. Proof rules in LFSC are defined as operators whose type may include a side condition on the rule's premises. Proof certificates are then encoded as LFSC terms and proof checking reduces to type checking, with the twist that, when checking the application of a proof rule with a side condition, an LFSC checker must compute the actual parameters and then execute the side condition, rejecting the rule application as ill-typed if the side condition fails.

The validity of an LFSC proof certificate thus relies on the correctness of the proof rules it uses, including the correctness of their side conditions, if any. CVC4 defines its own proof rules, in LFSC modules called *signatures*, for the various theories it supports. These signatures contain definitions for variables, clauses, and rules for propositional reasoning; they also contain definitions for terms, formulas, and rules for theory reasoning. More specifically, the signatures include: resolution rules; CNF-conversion rules; rules for equality and congruence over uninterpreted functions; types, constructors and rules for the theory of arrays with extensionality; and types and constructors for the theory of fixed-size bit vectors together with *bit-blasting* rules for that theory. They currently do not include definitions and rules for integer arithmetic since CVC4 does not yet produce proof certificates for that theory.

To extend SMTCoq to CVC4 we have developed (in OCaml) a translator/preprocessor from the fragment of LFSC used by CVC4 to the certificate format used by SMTCoq. A proof certificate in SMTCoq is a sequence of proof steps, where each proof step is either an input clause or the application of a *rule* to a (potentially empty) list of *derived* clauses together with a resulting clause. Each clause in the certificate is identified by a unique number. Before our extension, SMTCoq already had a set of predefined rules whose checkers had been proven correct in Coq.

A major difference between the two proof formats lies in the presentation of the deduction rules. In SMTCoq, the small checkers deduce new formulas from already known formulas (possibly with the help of a certificate produced by an external solver), making the proof format inherently linear. In contrast, the LFSC proof format is more general, not only because of the side conditions but also because it allows natural deduction-style rules which enable the construction of nested proofs. As a consequence, in some cases, our LFSC-to-SMTCoq translator needs to replay parts of the side conditions in LFSC rule applications in order to produce the premises, conclusions, and certificates needed by the small checkers in SMTCoq. The translator also linearizes nested proofs, adding intermediate resolution steps to the generated SMTCoq certificate. The whole transformation is feasible because LFSC proofs are more *fine-grained* than SMTCoq proofs; hence, they contain all the information needed by the SMTCoq checker. In particular, and contrary to other proof formats such as the one used by the Z3 SMT solver [4], there is no need to perform proof reconstruction on LFSC proof certificates.

2.1 Bit Vectors and Functional Arrays

We instrumented CVC4 to produce LFSC proofs for the quantifier-free fragment of the SMT theories of *fixed-width bit vectors* and *functional arrays with extensionality* [7, 10]. We extended SMTCoq correspondingly to accept and check proof certificates in these new theories. This involved two major steps:

1. We extended the Coq `formula` type with the function symbols of the bit vector and array theories, and provided a suitable encoding of bit vectors and arrays in Coq.
2. We extended the SMTCoq proof format with proof rules for the new theories, extended SMTCoq's checker correspondingly to check certificates containing applications of those rules, and proved the extended checker correct in Coq.

Step 2 was achieved by adding new certified small checkers to the SMTCoq checker for the new theories. Thanks to SMTCoq's original design, neither the existing small checkers nor their proofs of correctness had to be changed as a result of this addition.

Since Coq itself has limited support for bit vectors and arrays as defined in the SMT-LIB standard supported by CVC4, for Step 1 we had to design and implement complete Coq libraries for them.[5] Our bit vector library represents bit vectors with a dependent type `bitvector` parameterized by a positive integer n, for the bit vector size. It provides a complete formalization in Coq of the SMT-LIB theory of bit vectors with the exception of the division operator which is currently unsupported. The type `bitvector` is implemented as a list of Booleans together with a proof that the list has length n.

Our array library is inspired by Coq's own `FMapList` library. We encode arrays, which in SMT-LIB are just maps from some index type to some value type, as finite maps from keys to values together with a default value for all key that are not explicitly listed in the finite map.[6] This library provides a type `farray` parameterized by the key type and the value type, and makes extensive use of Coq's type classes to express necessary properties of the parameter types. For instance, key types are required to be endowed with a *total ordering*. This ordering ensures that arrays have a unique finite map encoding, allowing array equality to be reduced to structural equality over finite maps.

2.2 Coq Tactics and Proof Holes

SMTCoq brings the power of SAT and SMT solvers to Coq users by providing sound Coq tactics for solving subgoals with the aid of such solvers. Those tactics add an unprecedented level of automation to Coq, one that is still lacking in most interactive theorem provers. The original SMTCoq had two tactics, respectively for ZChaff and for veriT. We have developed two additional ones: `cvc4`, which relies only on CVC4, and `smt`, which takes advantage of CVC4 and veriT by falling back on the `cvc4` and the `verit` tactics as needed. Coq users may resort to these tactics whenever they can express some of their goals in terms of the SMTCoq libraries for EUF, LIA, bit vectors and functional arrays. We believe the latter is possible in many instances where Coq is used for verification purposes, given the usefulness of those theories in modeling verification problems, and considering the generality and extensibility of our Coq libraries.

Currently, CVC4 is not fully proof-producing for the theories above. As a result, its proof certificates may contain *holes*; that is, they may refer to lemmas that, while proven valid by CVC4 internally, are assumed without proof in the proof certificate. These include linear arithmetic lemmas as well as lemmas corresponding to equivalence-preserving simplification steps performed by CVC4 on the input formula encoding the

[5] These libraries are publicly available as part of the SMTCoq distribution.
[6] While not faithful to the SMT-LIB semantics, this encoding is adequate for universal goals.

```
Goal
 ∀ (a b: farray Z Z) (v w x y: Z)
   (r s: bitvector 4)
   (f: Z → Z)
   (g: farray Z Z → Z)
   (h: bitvector 4 → Z),
   a[x ← v] = b ∧ a[y ← w] = b →
   r = s ∧ h r = v ∧ h s = y →
   v < x + 1 ∧ v > x - 1 →
   f (h r) = f (h s) ∨ g a = g b.
Proof.
  cvc4.
  verit.
Qed.
```

(a)

```
1 subgoal, subgoal 1 (ID 10899)
  a, b : farray Z Z
  v, w, x, y : Z
  r, s : bitvector 4
  f : Z → Z
  g : farray Z Z → Z
  h : bitvector 4 → Z
  ====================================
  (a[x ← v] = b ∧ a[y ← w] = b →
   r = s ∧ h r = v ∧ h s = y →
   v < x + 1 ∧ x - 1 < v →
   f (h r) = f (h s) ∨ g a = g b)
  ↔ (a[x ← v] = b ∧ a[y ← w] = b →
      r = s ∧ h r = v ∧ h s = y →
      1 ≤ v + -1 * x ∧ 0 ≤ v + -1 * x →
      f (h r) = f (h s) ∨ g a = g b)
```

(b)

Figure 2: SMTCoq tactics. $a[i \leftarrow v]$ denotes array updates; `Z` is Coq's own integer type.

goal. To account for these *partial proofs*, we have extended SMTCoq's proof system with a new rule that permits holes in SMTCoq's proof certificates by introducing *cuts* in the proof. The result of this addition is that SMTCoq recognizes unproven lemmas and lifts them to Coq subgoals that are either further processed by the tactic that invoked SMTCoq or are returned to the user.

To illustrate this with a concrete example, Figure 2a shows a Coq goal involving all the theories now supported by SMTCoq. The first call to the `cvc4` tactic in the proof would be able to prove the goal fully automatically if it wasn't for the presence of a hole in the proof certificate produced by CVC4 due to an unproven rewriting step. The corresponding *subgoal* returned by the tactic, shown in Figure 2b, requires the user to prove the equivalence of a formula and its rewritten version (with the rewritten parts highlighted). This is a goal that the `verit` tactic, invoked by the user later in the proof, is able to solve in full by considering the symbols from the theory of arrays and bit vectors, which are not supported by veriT, as uninterpreted. The composite tactic `smt` can actually solve the original goal of Figure 2a with no user assistance by combining the other tactics automatically.

3 Conclusion and Future Work

SMTCoq has been designed to be easily extensible to new theories and/or solvers. We presented one such extension with CVC4 as a background solver and the theories of bit vectors and arrays as new supported theories. We argued that the various tactics provided by SMTCoq greatly increase the automation capabilities of Coq. We have verified that by developing and testing several examples not discussed here but available in the SMTCoq distribution. In our extension, SMT solvers are not required to justify all their steps in a proof thanks to a cut mechanism that allows part of the proof to be discharged manually or by other means. We see this as a crucial feature in practice since it allows SMTCoq

to take advantage of the automated reasoning power of SMT solvers, even when they do not provide full proof-generation support for all theories of interest to a particular problem.

We are working on extending the SMTCoq proof system to allow nested proofs, which will simplify the conversion from CVC4 proofs. In future work, we plan to continue expanding the scope of CVC4's proof generator to additional theories and proof steps, and extend SMTCoq correspondingly. Finally, we plan to add proof support in both tools for goals with quantifier alternations. This is a major extension that might require us to represent SMT formulas in Coq natively as propositions, instead of Boolean terms, and handle classical reasoning either by adding classical logic axioms to Coq or by restricting attention to decidable predicates.

References

1. C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proc. Computer Aided Verification (CAV)*, pages 171–177, 2011.
2. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In *Proc. 8th Int. Workshop on Satisfiability Modulo Theories (SMT)*, page 14, 2010.
3. F. Besson, P. Fontaine, and L. Théry. A Flexible Proof Format for SMT: a Proposal. In *Proc. 1st Int. Workshop on Proof eXchange for Theorem Proving (PxTP)*, pages 15–26, 2011.
4. S. Böhme and T. Weber. Fast LCF-Style Proof Reconstruction for Z3. In *Proc. 1st Int. Conf. on Interactive Theorem Proving (ITP)*, pages 179–194, 2010.
5. T. Bouton, D. de Oliveira, D. Déharbe, and P. Fontaine. veriT: An Open, Trustable and Efficient SMT-Solver. In *Proc. 22nd Int. Conf. on Automated Deduction (CADE)*, pages 151–156, 2009.
6. G. Gonthier and A. Mahboubi. An Introduction to Small Scale Reflection in Coq. *J. Formalized Reasoning*, 3(2):95–152, 2010.
7. L. Hadarean, C. Barrett, A. Reynolds, C. Tinelli, and M. Deters. Fine Grained SMT Proofs for the Theory of Fixed-Width Bit-Vectors. In *Proc. 20th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 340–355, 2015.
8. R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *J. ACM*, 40(1):143–184, 1993.
9. J. Jourdan, F. Pottier, and X. Leroy. Validating LR(1) Parsers. In *Proc. 21st European Symposium on Programming (ESOP)*, pages 397–416, 2012.
10. G. Katz, C. Barrett, C. Tinelli, A. Reynolds, and L. Hadarean. Lazy Proofs for DPLL(T)-Based SMT Solvers. *Proc. 16th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 93–100, 2016.
11. C. Keller. *A Matter of Trust: Skeptical Communication Between Coq and External Provers*. PhD thesis, École Polytechnique, June 2013.
12. Y. Mahajan, Z. Fu, and S. Malik. Zchaff2004: An Efficient SAT Solver. In *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT)*, pages 360–375, 2004.
13. A. Reynolds, L. Hadarean, C. Tinelli, Y. Ge, A. Stump, and C. Barrett. Comparing Proof Systems for Linear Real Arithmetic with LFSC. In *Proc. 8th Int. Workshop on Satisfiability Modulo Theories (SMT)*, 2010.
14. A. Stump, D. Oe, A. Reynolds, L. Hadarean, and C. Tinelli. SMT Proof Checking Using a Logical Framework. *Formal Methods in System Design*, 41(1):91–118, Feb. 2013.