Extending SMTCoq, a Certified Checker for SMT (Extended Abstract)

Burak Ekici, Guy Katz, Chantal Keller, Alain Mebsout, Andrew Reynolds, Cesare Tinelli

Extending SMTCoq, a Certified Checker for SMT (Extended Abstract)

Burak Ekici
The University of Iowa
Iowa City, USA
burak-ekici@uiowa.edu

Guy Katz
New York University
New York, USA
guy.katz@nyu.edu

Chantal Keller
LRI, Université Paris-Sud
Orsay, France
Chantal.Keller@lri.fr

Alain Mebsout
The University of Iowa
Iowa City, USA
alain-mebsout@uiowa.edu

Andrew J. Reynolds
The University of Iowa
Iowa City, USA
andrew-reynolds@uiowa.edu

Cesare Tinelli
The University of Iowa
Iowa City, USA
cesare-tinelli@uiowa.edu

This extended abstract reports on current progress of SMTCoq, a communication tool between the Coq proof assistant and external SAT and SMT solvers. Based on a checker for generic first-order certificates implemented and proved correct in Coq, SMTCoq offers facilities both to check external SAT and SMT answers and to improve Coq's automation using such solvers, in a safe way. Currently supporting the SAT solver ZChaff, and the SMT solver veriT for the combination of the theories of congruence closure and linear integer arithmetic, SMTCoq is meant to be extendable with a reasonable amount of effort: we present work in progress to support the SMT solver CVC4 and the theory of bit vectors.

1 Introduction

SMTCoq[1] [1] is a tool that allows the Coq [2] proof assistant to communicate with external automatic solvers for Boolean satisfiability (SAT) and Satisfiability Modulo Theories (SMT). Its twofold goal is to:

- increase the confidence in SAT and SMT solvers: SMTCoq provides an independent and certified checker for SAT and SMT proof witnesses;
- safely increase the level of automation of Coq: SMTCoq provides starting safe tactics to solve a class of Coq goals automatically by calling external solvers and checking their answers (following a *skeptical* approach).

SMTCoq currently supports the SAT solver ZChaff [19] and the SMT solver veriT [10] for the quantifier-free fragment of the combined theory of linear integer arithmetic and equality with uninterpreted functions. For this combined theory, SMTCoq's certificate checker has proved to be as efficient as state-of-the-art certified checkers [1, 9].

There is a large variety of SAT and SMT solvers, with each solver typically excelling at solving problems in some specific class of propositional or first-order problems. While the SAT and SMT communities have adopted standard languages for expressing *input* problems (namely the DIMACS standard for SAT and the SMT-LIB [4] standard for SMT), agreeing on a common *output* language for proof witnesses has proven to be more challenging. Several formats [11, 21, 6] have been proposed but none has emerged as a standard yet. Each proof-producing solver currently implements its own variant of these formats.

[1] SMTCoq is distributed as free software at `https://github.com/smtcoq/smtcoq`.

Preliminary Report. Final version to appear in:
HaTT'16

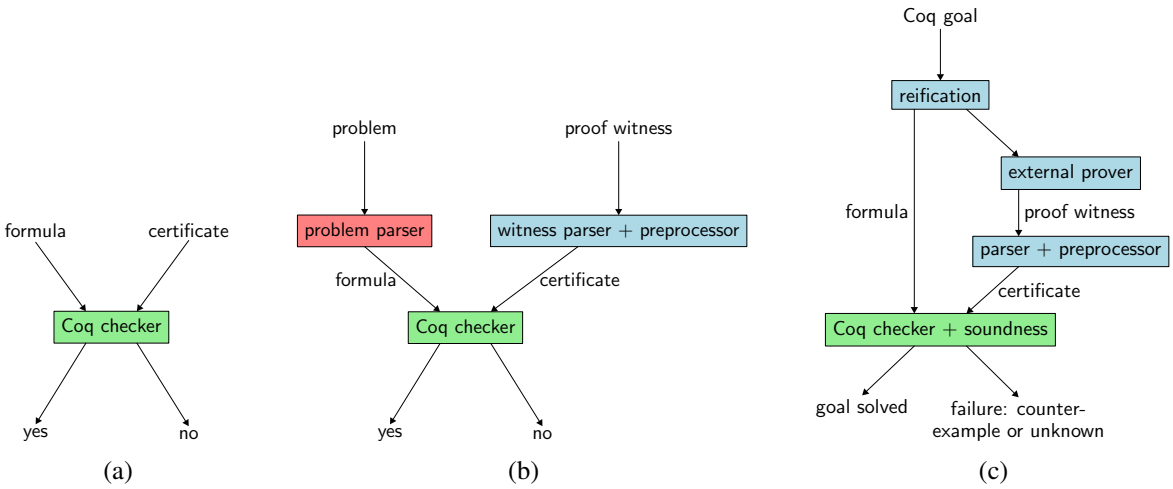

Figure 1: SMTCoq's main checker and its uses

To be able to combine the advantages of multiple SAT and SMT solvers despite the lack of common standards for representing proof certicates, SMTCoq has been designed to be modular along two dimensions:

- supporting new theories: SMTCoq's main checker is an extendable combination of independent *small checkers*;
- supporting new solvers: SMTCoq's kernel relies on a generic certificate format that can encode most SAT and SMT reasonings for supported theories; the encoding can be done during a *preprocessing* phase, which does not need to be certified.

In this abstract, we emphasize the key ideas behind the modularity of SMTCoq, and validate this by reporting on work in progress on the integration of the SMT solver CVC4 [3] and the theory of bit vectors. We simultaneously aim at:

- offering to CVC4 users the possibility to formally check its answers in a trusted environment like Coq;
- bringing the power of a versatile and widely used SMT solver like CVC4 to Coq;
- providing in Coq a decision procedure for bit vectors, a theory widely used, for instance, for verifying circuits or programs using machine integers.

2 The SMTCoq Tool

2.1 General Idea

The heart of SMTCoq is a checker for a generic format of certificates (close to the format proposed by Besson *et al.* [6]), implemented and proved correct inside Coq (see Figure 1a). Taking advantage of Coq's computational capabilities the SMTCoq checker is fully executable, either inside Coq or after extraction to a general-purpose language [18].

The Coq signature of this checker is the following:

```
checker : formula → certificate → bool
```

where the type `formula` represents the deep embedding in Coq of SMT formulas, and the type `certificate` represents SMTCoq's format of certificates.

The checker's soundness is stated with respect to a translation function from the deep embedding of SMT formulas into Coq terms:

$[\![\bullet]\!]$: `formula` $\rightarrow$ `bool`

that interprets every SMT formula into its Coq Boolean counterpart. The correctness of the checker:

`checker_sound` : $\forall$ `f c, checker f c = true` $\rightarrow$ $[\![$`f`$]\!]$

thus means that, given a formula and a certificate for which the checker answers positively, then the interpretation in Coq of the formula is valid.

The choice of the type of Booleans `bool` as the codomain of the translation function $[\![\bullet]\!]$, instead of the type of (intuitionistic) propositions `Prop`, allows us to handle the checking of the classical reasoning made by SMT solvers without adding any axioms. The SSReflect [12] plugin for Coq can be used to bridge the gap between propositions and Booleans for the theories considered by SMTCoq. The major shortcoming of this approach is that it does not allow quantifiers inside goals sent to SMT solvers, although it does not prevent one from feeding these solvers universally quantified lemmas. To increase the expressivity of SMTCoq with respect to quantifiers, one will need to switch to propositions, and handle classical logic either by axioms or by restricting attention to decidable atoms of the considered combined theory.

The first use case of this correct-by-construction checker is to check the validity of a proof witness, or proof *certificate* coming from an external solver against some input problem (Figure 1b). In this use case, the trusted base is both Coq and the parser of the input problem. The parse is part of the trusted based because we need to make sure we are effectively verifying a proof of the problem we sent to the external solver. However, this parser is fairly straightforward.

The second use case is within a Coq tactic (Figure 1c). We can give a Coq goal to an external solver and get a proof certificate for it. If the checker can validate the certificate, the soundness of the checker allow us to establish a proof of the initial goal. This process is known as *computational reflection* as it uses a computation (here, the execution of the checker) inside a proof. In this use case, the trusted base consists only of Coq: if something else goes wrong (e.g., the checker cannot validate the certificate), the tactic will fail, but nothing unsound will be added to the system.

In both cases, a crucial aspect for modularity purposes is the possibility to *preprocess* proof certificates before sending them to the SMTCoq checker, without having to prove anything about this preprocessing stage. Again, if the preprocessor is buggy, the checker will fail to validate the proof certificate (by returning `false`), which means that while nothing is learned, nothing unsafe is added to Coq's context. This allows us to easily extend SMTCoq with new solvers: as long as the certificate coming from the new solver can be logically encoded into SMTCoq's certificate format, we can implement this encoding at the preprocessing stage. As a result, SMTCoq's current support for both ZChaff and veriT is provided through the implementation of a preprocessor for each solver. Both preprocessors convert to the same proof format, thus sharing the same checker.

Using a preprocessor is also beneficial for efficiency: proof certificates may be encoded more compactly before being sent to the SMTCoq checker, which may improve performance.

2.2 The Checker

We now provide more details on the checker of SMTCoq. As presented in Figure 2, it consists of a *main checker* obtained as the combination of several *small checkers*, each specialized in one aspect of proof

checking in SMT (e.g., CNF conversion, propositional reasoning, reasoning in the theory of equality, linear arithmetic reasoning, and so on).

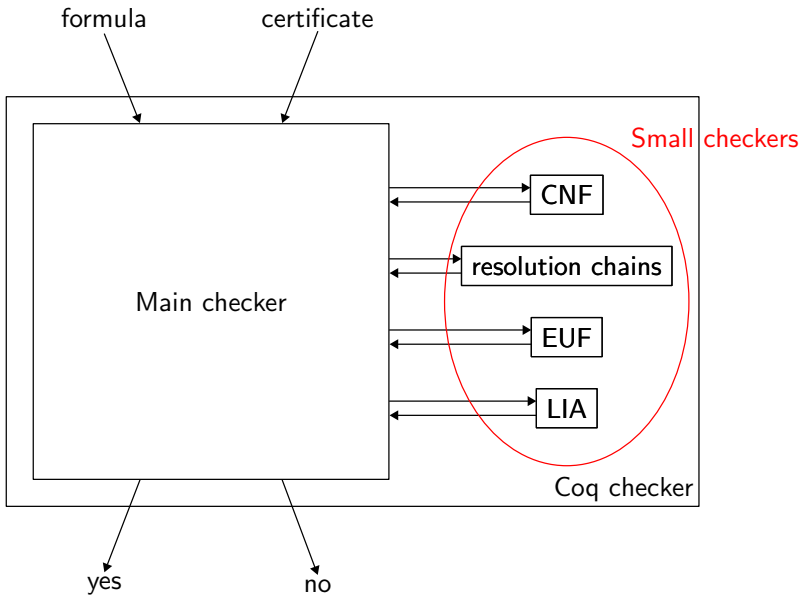

Figure 2: Internals of the Coq checker

The type `certificate` is actually the aggregation of specialized types, one for each small checker. The role of the main checker is thus to dispatch each piece of the certificate to its dedicated small checker, until the initial formula is proved.

A small checker is a Coq program that, given a (possibly empty) list of formulas and a certificate associated with it (which may be just a piece of the input certificate), computes a new formula:

```
small_checker : list formula → certificate_sc → formula
```

The soundness of the `checker` comes from the soundness of each small checker, stated as follows:

$$\texttt{small\_checker\_sound} : \forall\ \texttt{f}_1\ \ldots\ \texttt{f}_n\ \texttt{c},\quad [\![\texttt{f}_1]\!] \wedge \ldots \wedge [\![\texttt{f}_n]\!] \rightarrow [\![\texttt{small\_checker}\ [\texttt{f}_1;\ldots;\texttt{f}_n]\ \texttt{c}]\!]$$

meaning that the small checker returns a formula which is implied (after translation into Coq's logic) by the conjunction of its premises. Note that the list of premises may be empty: in such a case, the small checker returns a tautology in Coq.

Here are some examples of small checkers.

- For propositional resolution chains, the checker takes as input a list of premises and returns a resolvent if it exists, or a trivially true clause otherwise. In this case, a certificate is not required as part of the small checker's input.
- For the theory of equality with uninterpreted functions (EUF), the checker takes as input a formula in this theory formulated as a certificate (corresponding to a theory lemma produced by the SMT solver), and returns the formula if it is able to check it, or a trivially true clause otherwise. In this case, no premises are given.

- For linear integer arithmetic (LIA), the checker works similarly to the EUF checker, but checks the formula using Micromega [5], an efficient decision procedure for this theory implemented in Coq.

The only thing that small checkers need to share is the type `formula`, and its interpretation into Coq Booleans. Each small checker may then reason independently, using separate pieces of the certificate. Again, this is crucial for modularity: to extend SMTCoq with a new theory, one only has to extend the type `formula` with the signature of this theory and, independently of the already existing checkers, implement a small checker for this theory and prove its soundness.

Notice that "small checker" can be understood in a very general sense: any function that, given a list of first-order formulas, returns an implied first-order formula, can be plugged into SMTCoq as a small checker. In principle, such a checker could even be as complex as an SMT solver, as long as it can be proved correct in Coq.

3 Work in Progress: Extensions to CVC4 and Bit Vector Arithmetic

3.1 Support for CVC4

CVC4 is a proof-producing SMT solver, whose proof format is based on the Logical Framework with Side Conditions (LFSC) [21]. LFSC extends the Edinburgh Logical Framework (LF) [14] by allowing types with computational *side conditions*, explicit computational checks defined as programs in a small but expressive functional first-order programming language. The language has built-in types for arbitrary precision integers and rationals, ML-style pattern matching over LFSC type constructors, recursion, a minimal support for exceptions, and a very restricted set of imperative features. One can define proof rules in LFSC as typing rules that may optionally include a side condition written in this language. When checking the application of such proof rules, an LFSC checker computes actual parameters for the side condition and executes its code; if the side condition fails, the LFSC checker rejects the rule application. The validity of an LFSC proof witness thus relies on the correctness of the side condition functions used in the proof. LFSC comes with a set of pre-defined side conditions for various theories, used by the CVC4 proof production mechanism.

The key differences between LFSC and the SMTCoq format are presented in Table 1.

	LFSC	SMTCoq
Rules	deduction + computation	deduction + certificate
Nested proofs	supported	not supported

Table 1: Main differences between the LFSC and SMTCoq certificate formats

The major difference lies in the presentation of the deduction rules. In SMTCoq, the small checkers deduce a new formula from already known formulas, possibly with the help of a piece of certificate that depends on the theory. The LFSC format is more uniform, thanks to the side conditions described above.

To support LFSC, and so CVC4, we are in the process of implementing (in OCaml) an untrusted preprocessor that transforms LFSC proofs into SMTCoq proofs. To this end, for some theories, we need to replay parts of the side conditions, in order to produce the corresponding SMTCoq premises, conclusion and piece of certificate that will be passed to the small checkers. This encoding, however, is relatively straightforward:

- for propositional reasoning, LFSC side conditions use the same logical content as SMTCoq rules;

- CNF conversion and EUF proofs are nested in LFSC, so they require some processing for the moment;
- for linear integer arithmetic, since SMTCoq relies on an existing decision procedure in Coq, it only needs to know what theory lemma is being proved, and can ignore the actual proof steps in the LFSC certificate.

One difficulty in translating LFSC proofs to the SMTCoq format comes from to the possibility in LFSC of using natural-deduction-style proofs, where one can nest one proof inside another. For instance, it is possible to have lemmas inside an LFSC proof whose witnesses are themselves LFSC proofs. The architecture of the main and small checkers of SMTCoq does not currently allow this sort of nesting: every clause produced by the small checkers needs to be a direct consequence of input clauses or clauses that were previously produced. To encode an LFSC proof into SMTCoq, our preprocessor thus linearizes nested proofs. The LFSC proofs generated by CVC4 are constructed in such a way that this does not cause a blow-up in practice; however, to support LFSC in general, we plan to extend SMTCoq certificates with nested proofs. Again, this extension should be made easier by the modularity inside the checker. It should impact only the main checker, and not the various small checkers already in SMTCoq.

3.2 Support for Bit Vector Arithmetic

CVC4 has been recently extended to produce LFSC proofs for the quantifier-free fragment of the SMT theory of bit vectors [13]. To check proof certificates in this theory, SMTCoq needs be extended with it. As explained in Section 2.2, to do that one needs to:

1. extend the Coq representation of formulas with the signature of the bit vector theory and the interpretation function into Coq terms;
2. implement (new) small checkers and their corresponding certificates for this theory, and prove their correctness.

Step 1 is a simple extension on the SMTCoq side. The major difficulty is that Coq itself has limited support for bit vectors. Its bit vector library provides only the implementation of bitwise operations (and not arithmetic operations), and no proofs. We are thus currently implementing a more complete library for this theory. Step 2 involves implementing and adding new certified Coq programs (the small checkers). As mentioned, however, because of SMTCoq's design, none of the previous small checkers and their proofs of correctness need to be changed as a result of this addition.

LFSC proofs for bit vectors produce by CVC4 mainly involve the following two kinds of deduction steps:

- *bit-blasting* steps that reduce the input bit vector formula to an equisatisfiable propositional formula;
- standard propositional reasoning steps (based on resolution).

The propositional steps can be handled directly by previous small checkers. For the bit-blasting steps, we implemented new small checkers that relate terms of the bit vector theory with lists of Boolean formulas representing their bits; we are currently working on producing proofs of correctness in Coq for these small checkers.

LFSC proofs generated by CVC4 involve a third kind of step: formula simplifications based on the equivalence of two bit-vector terms or atomic formulas (for instance, by normalizing inequalities). Currently, these simplification steps are not provided a detailed LFSC subproof by CVC4, although there

are plans to do so in the near future. In the current SMTCoq implementation then, we assume those steps, as in the LFSC proof coming from CVC4, or let the user prove them, in the case of tactics. Since those steps correspond to applications of CVC4-defined rewriting and simplification rules, we plan for now to prove the correctness of these rules once and for all at the Coq level, and to pre-process simplification steps into applications of these rules.

4 Related Work

In addition to related work already discussed throughout the paper, we now briefly mention a few more notable projects. Heule *et al.* implemented an efficient checker for state-of-the-art SAT techniques, verified in ACL2 [15, 24]. It is mainly based on a generalization of extended resolution [22, 17] and on reverse unit propagation [11]. SMTCoq currently handles only standard extended resolution for its propositional part.

Efficient proof reconstruction for SAT and SMT solvers has been implemented in proof assistants based on higher-order logic [23, 9]. Some of these reconstructions also handle the theory of bit vectors [8]. This approach is based on translating SAT/SMT certificates to applications of the inference rules of the kernels of these proof assistants. In contrast, our approach in Coq is based on computational reflection: the certificate is directly processed by the reduction mechanism of Coq's kernel.

Based on an efficient encoding of a large subset of HOL goals into first-order logic, the Sledgehammer tactic [20] allows HOL-based proof assistants to efficiently and reliably help manual proving. Proofs are replayed using either the proof reconstruction mechanism described above or a built-in first-order prover. We hope that SMTCoq can help in adding such techniques into Coq and other Type Theory-based proof assistants, by providing a proof replay mechanism based on certificates.

5 Conclusion and Future Work

SMTCoq has been designed to be modular in such a way that facilities its extension with new solvers and new theories. In particular, such extensions should not require any changes in existing checkers or in their proofs of soundness. Thus, while it may require some effort to certify new small checkers or to translate new proof formats into the SMTCoq format, such extensions require only local changes. Our current extensions to CVC4 and bit vectors arithmetic validate this goal: indeed, the work so far consisted mostly in implementing an untrusted preprocessor for certificates and adding new, independent checkers. One limiting aspect of SMTCoq is the lack of support for nested proofs, which we plan to add. Thanks to the modularity of the checker, we believe this feature too can be added locally.

In the future we plan to continue extending the expressivity of SMTCoq, and in particular to offer support for the SMT theory of arrays (for which CVC4 is also proof-producing). We believe we can match, and perhaps even improve upon existing work in terms of efficiency.

The current major limitation of SMTCoq resides in its set of tactics: presently, it can only handle goals that are directly provable by SMT solvers, without much encoding of Coq logic into first-order logic. Our longer term plan is to combine ongoing work on *hammering* [7] for proof assistants based on Type Theory (such as Coq) with the certificate checking capabilities offered by SMTCoq.

Acknowledgments. We wish to thank the anonymous reviewers for their helpful and constructive feedback. This work was supported in part 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 above are those of the authors and do not necessarily reflect the views of AFRL or DARPA.

References

[1] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry & Benjamin Werner (2011): *A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses*. In Jouannaud & Shao [16], pp. 135–150, doi:10.1007/978-3-642-25379-9_12.

[2] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliâtre, E. Gimenez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy et al. (2000): *The Coq proof assistant: reference manual*. Technical Report, INRIA.

[3] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds & Cesare Tinelli (2011): *CVC4*. In Ganesh Gopalakrishnan & Shaz Qadeer, editors: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, *Lecture Notes in Computer Science* 6806, Springer, pp. 171–177, doi:10.1007/978-3-642-22110-1_14.

[4] Clark Barrett, Aaron Stump & Cesare Tinelli (2010): *The SMT-LIB Standard: Version 2.0*. In A. Gupta & D. Kroening, editors: *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*.

[5] F. Besson (2006): *Fast Reflexive Arithmetic Tactics the Linear Case and Beyond*. In Thorsten Altenkirch & Conor McBride, editors: *TYPES*, *Lecture Notes in Computer Science* 4502, Springer, pp. 48–62, doi:10.1007/978-3-540-74464-1_4.

[6] F. Besson, P. Fontaine & L. Théry (2011): *A Flexible Proof Format for SMT: a Proposal*. In: *PxTP 2011: First International Workshop on Proof eXchange for Theorem Proving August 1, 2011 Affiliated with CADE 2011, 31 July-5 August 2011 Wrocław, Poland*, pp. 15–26.

[7] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson & Josef Urban (2016): *Hammering towards QED*. *J. Formalized Reasoning* 9(1), pp. 101–148, doi:10.6092/issn.1972-5787/4593.

[8] Sascha Böhme, Anthony C. J. Fox, Thomas Sewell & Tjark Weber (2011): *Reconstruction of Z3's Bit-Vector Proofs in HOL4 and Isabelle/HOL*. In Jouannaud & Shao [16], pp. 183–198, doi:10.1007/978-3-642-25379-9_15.

[9] Sascha Böhme & Tjark Weber (2010): *Fast LCF-Style Proof Reconstruction for Z3*. In Matt Kaufmann & Lawrence C. Paulson, editors: *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, *Lecture Notes in Computer Science* 6172, Springer, pp. 179–194, doi:10.1007/978-3-642-14052-5_14.

[10] T. Bouton, D.C.B. de Oliveira, D. Déharbe & P. Fontaine (2009): *veriT: An Open, Trustable and Efficient SMT-Solver*. In R. A. Schmidt, editor: *CADE*, *Lecture Notes in Computer Science* 5663, Springer, pp. 151–156, doi:10.1007/978-3-642-02959-2_12.

[11] Allen Van Gelder (2012): *Producing and verifying extremely large propositional refutations - Have your cake and eat it too*. *Ann. Math. Artif. Intell.* 65(4), pp. 329–372, doi:10.1007/s10472-012-9322-x.

[12] Georges Gonthier & Assia Mahboubi (2010): *An introduction to small scale reflection in Coq*. *J. Formalized Reasoning* 3(2), pp. 95–152, doi:10.6092/issn.1972-5787/1979.

[13] Liana Hadarean, Clark W. Barrett, Andrew Reynolds, Cesare Tinelli & Morgan Deters (2015): *Fine Grained SMT Proofs for the Theory of Fixed-Width Bit-Vectors*. In Martin Davis, Ansgar Fehnker, Annabelle McIver & Andrei Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, *Lecture Notes in Computer Science* 9450, Springer, pp. 340–355, doi:10.1007/978-3-662-48899-7_24.

[14] Robert Harper, Furio Honsell & Gordon D. Plotkin (1993): *A Framework for Defining Logics*. *J. ACM* 40(1), pp. 143–184, doi:10.1145/138027.138060.

[15] Marijn Heule, Warren A. Hunt Jr. & Nathan Wetzler (2013): *Verifying Refutations with Extended Resolution*. In Maria Paola Bonacina, editor: *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, *Lecture Notes in Computer Science* 7898, Springer, pp. 345–359, doi:10.1007/978-3-642-38574-2_24.

[16] Jean-Pierre Jouannaud & Zhong Shao, editors (2011): *Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*. *Lecture Notes in Computer Science* 7086, Springer, doi:10.1007/978-3-642-25379-9.

[17] Oliver Kullmann (1999): *On a Generalization of Extended Resolution*. *Discrete Applied Mathematics* 96-97, pp. 149–176, doi:10.1016/S0166-218X(99)00037-2.

[18] Pierre Letouzey (2002): *A New Extraction for Coq*. In Herman Geuvers & Freek Wiedijk, editors: *Types for Proofs and Programs, Second International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers*, *Lecture Notes in Computer Science* 2646, Springer, pp. 200–219, doi:10.1007/3-540-39185-1_12.

[19] Yogesh S. Mahajan, Zhaohui Fu & Sharad Malik (2004): *Zchaff2004: An Efficient SAT Solver*. In Holger H. Hoos & David G. Mitchell, editors: *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, *Lecture Notes in Computer Science* 3542, Springer, pp. 360–375, doi:10.1007/11527695_27.

[20] Lawrence C. Paulson & Jasmin Christian Blanchette (2010): *Three years of experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers*. In Geoff Sutcliffe, Stephan Schulz & Eugenia Ternovska, editors: *The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011*, *EPiC Series* 2, EasyChair, pp. 1–11. Available at `http://www.easychair.org/publications/?page=820355915`.

[21] Aaron Stump, Duckki Oe, Andrew Reynolds, Liana Hadarean & Cesare Tinelli (2013): *SMT proof checking using a logical framework*. *Formal Methods in System Design* 42(1), pp. 91–118, doi:10.1007/s10703-012-0163-3.

[22] G. Tseitin (1970): *On the Complexity of Proofs in Propositional Logics*. In: *Seminars in Mathematics*, 8, pp. 466–483.

[23] T. Weber (2008): *SAT-based Finite Model Generation for Higher-Order Logic*. Ph.D. thesis, Institut für Informatik, Technische Universität München, Germany. Available at `http://www.cl.cam.ac.uk/~tw333/publications/weber08satbased.html`.

[24] Nathan Wetzler, Marijn Heule & Warren A. Hunt Jr. (2013): *Mechanical Verification of SAT Refutations with Extended Resolution*. In Sandrine Blazy, Christine Paulin-Mohring & David Pichardie, editors: *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, *Lecture Notes in Computer Science* 7998, Springer, pp. 229–244, doi:10.1007/978-3-642-39634-2_18.