SMTCoq: Mixing automatic and interactive proof technologies

Chantal Keller

Abstract SMTCoq is a plugin for the Coq interactive theorem prover to work in conjunction with automated theorem provers based on Boolean Satisfiability (SAT) and Satisfiability Modulo Theories (SMT), in an efficient and expressive way. As such, it allows one to formally establish, in a proof assistant, mathematical results relying on large combinatorial properties that require automatic Boolean reasoning. To this end, the huge SAT proofs coming from such problems can be safely checked in Coq and combined with standard mathematical Coq developments in a generic and modular way, for instance to obtain a formal proof of the Erdős Discrepancy Conjecture.

To achieve this objective with the same degree of safety as Coq itself, SMTCoq communicates with SAT and SMT solvers that, in addition to a yes/no answer, can output traces of their internal proof search. The heart of SMTCoq is thus a certified, efficient and modular checker for such traces expressed in a format that can encompass most aspects of SMT reasoning. Preprocessors - that need not be certified - for proof traces coming from the state-of-the-art SMT solvers CVC4 and veriT and SAT solvers zChaff and Glucose are implemented. Coq can thus work in conjunction with widely used provers.

From a proof assistant perspective, SMTCoq also provides a mechanism to let Coq users enjoy automation provided by external provers.

Chantal Keller
LRI, Univ. Paris-Sud, CNRS UMR 8623, Université Paris-Saclay
Bât 650 Ada Lovelace
Université Paris Sud
91405 Orsay Cedex France
e-mail: Chantal.Keller@lri.fr

1 Introduction

Mechanization of mathematical reasoning can be seen as starting from two somewhat opposite applications [Mac95].

On the one hand, *interactive theorem provers* (also known as *proof assistants*) aim at checking (even complex) mathematical proofs with great confidence. Theorems and proofs should be stated and written interactively by mathematicians, with the help of the system to deduce facts, discharge automatically trivial sub-goals, and check the actual proof. To achieve confidence, these systems rely on a *kernel* that is a piece of code, as small as possible, implementing a proof checker for a well-defined logic [H+96]. Among the most successful current interactive theorem provers, one can cite the HOL family [Gor00] (HOL4, HOL Light, Isabelle/HOL), the type-theoretical family (Agda [Nor09], Coq [HH14], Lean [dMKA+15], Matita [ARCT11], ...) and many other systems such as PVS [ORS92], Mizar [BBG+15] or Nuprl [ACE+00]. Interactive theorem provers often come with high-level *tactics* that translate the interaction with the user into low-level proofs that are checked by the kernel. These tactics offer the possibility of having safe automation by performing complex reasoning while relying on the kernel. Most of the time, such tactics are dedicated decision procedures [GM05, Bes06]: they can automatically solve problems that belong to a recognized fragment of a logic.

On the other hand, *automated theorem provers* aim at finding proofs fully automatically. Theorems should be stated in a logic accepted by the system which may, in return, prove it, give a counter-example, or fail, if the problem falls into an undecidable fragment or the proof search exceeds some heuristic limit. While the algorithms are shown to be correct on paper, actual implementations involve fast automatic proof searches and may thus contain bugs [BB09]. They are very powerful tools in proving automatically generated goals, for instance in the context of proof of programs [FP13, SHK+16]. More recently they have been used to settle combinatorial problems such as the Erdős Discrepancy Conjecture (for discrepancy up to 3) [KL15] or the Boolean Pythagorean Triples problem [HKM16]. Currently, the most two successful approaches rely on satisfiability [BHvMW09], in particular with Conflict-Driven Clause Learning SAT [SLM09] and SMT (Satisfiability Modulo Theories) provers (zChaff [FMM07], CVC4 [BCD+11], Z3 [dMB08], veriT [BODF09], ...), and saturation-based resolution and superposition [BG98] with first-order provers (SPASS [WDF+09], Vampire [RV02], E [Sch13], ...).

The idea to deploy both human interaction and expressive automation in a single tool started in the '90s. One can in particular cite NQTHM and its successor ACL2 [KM96], which implements an interactive prover on top of a powerful automated prover. Confidence was achieved by implementing ACL2 in ACL2 itself, with the possibility of establishing properties on the system.

Subsequently, the independent success of both interactive and automated provers led again to the need for reconciling both worlds. A first approach, called the *autarkic* approach, close to what has been done for ACL2, was to implement and prove correct automatic provers inside proof assistants [Les11, Hur05]. This approach for-

mally establishes the correctness of the underlying algorithms, but has the major drawback of fixing an implementation that would be very difficult to enhance without re-doing most of the proof work. More recently, the *skeptical* approach makes use of external solvers that, in addition to a yes/no answer, can output a *certificate*, that is to say the arguments underlying the proof they found, that allows proof reconstruction in the proof assistants [BW10, PB10, BKPU16, AFG+11]. This approach actually scales since it allows the use of state-of-the-art external solvers, which may evolve independently. Moreover, proof checking is faster and easier rather than proof search.

The skeptical approach has had a major success with the Isabelle/HOL *sledgehammer* tactic [PB10], that employs multiple external solvers in parallel and reconstructs an Isabelle/HOL proof script based in particular on autarkic solvers. It allows users of this interactive prover to sketch the interesting part of mathematical proofs (induction, intermediate lemmas) and leave the remaining automatic. This idea has been recently ported to the Coq proof assistant [BKPU16].

In this chapter, we present SMTCoq[1] [AFG+11, EKK+16, EMT+17], a Coq plugin to interact with SAT and SMT external solvers via certificates. The objective is to provide a generic and efficient proof checker for automated provers, with the same degree of confidence as Coq itself. This tool can be used to take advantage of automation in Coq (Sec. 4.3). Care has also been taken to ensure efficiency of proof checking (Sec. 3), which allows for the certification of big certificates that arise when proving large combinatorial problems (Sec. 4.1 and 4.2). We start by explaining the kind of problems that SMTCoq handles (Sec. 2).

2 The Satisfiability and Satisfiability Modulo Theories problems

SAT solvers are automated provers to decide the satisfiability of (quantifier-free) Boolean formulas. They rely on an efficient exploration of the possible models of such formulas. By nature, they are powerful tools to solve combinatorial problems [KL15, HKM16].

On top of them, SMT solvers (standing for *Satisfiability Modulo Theory* solvers) automatically determine whether first-order formulas living in a combination of theories are satisfiable. Theories often include equality, arithmetic over various domains (integers, rationals, reals), and representation of data-structures (algebraic data-structures, arrays, bit vectors, ...). SMT solvers try to satisfy SMT formulas by the interaction of a SAT solver with theory reasoners (see [NOT06] for a detailed explanation), with the possibility to instantiate quantified hypotheses [dMB07, Bar16], making the logic very expressive.

[1] SMTCoq is available at https://smtcoq.github.io.

2.1 Examples

Let us illustrate the kind of problems SAT and SMT can be used for on the combinatorial example of the Erdős Discrepancy Conjecture.

Conjecture 1 (Erdős Discrepancy Conjecture). For any infinite sequence $\langle x_1, x_2, \dots \rangle$ of ± 1 integers and any integer C, there exist integers k and d such that

$$\left| \sum_{i=1}^{k} x_{i \times d} \right| > C$$

To prove this conjecture for a particular C_0, one has to find a length l of sequences such that the formula

$$\forall \langle x_1, x_2, \dots, x_l \rangle, \forall k, \forall d, \left| \sum_{i=1}^{k} x_{i \times d} \right| \leqslant C_0$$

is unsatisfiable. Konev and Lisitsa proposed a non trivial encoding of this problem into SAT, allowing to prove the conjecture up to $C = 3$ with modern SAT solvers [KL15]. More naively, the problem can be easily encoded into SMT using the theory of Linear Integer Arithmetic.

Example 1. For $l = 6$, the formula can be encoded in SMT by the conjunction of:

- $(x_1 = -1 \vee x_1 = 1) \wedge \dots \wedge (x_6 = -1 \vee x_6 = 1)$ (domain of the sequence)
- $(-C \leqslant x_1 + x_2 \leqslant C) \wedge \dots \wedge (-C \leqslant x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leqslant C)$ (sums for $d = 1$)
- $(-C \leqslant x_2 + x_4 \leqslant C) \wedge (-C \leqslant x_2 + x_4 + x_6 \leqslant C)$ (sums for $d = 2$)
- $(-C \leqslant x_3 + x_6 \leqslant C)$ (sum for $d = 3$)

Most SMT solvers supporting integer arithmetic are able to prove the conjecture for $C_0 = 1$ by choosing the encoding for $l = 12$.

Theorem 1. *Any sequence of length at least* 12 *has discrepancy at least* 2.

The proof of such a theorem relies on (a) an encoding of the original problem into a SMT formula, encoding which must be automatically generated for $C \geqslant 2$ since the formula becomes very large and (b) an automatic proof from a SAT or SMT solver. To increase confidence, one can formally establish such theorems in a proof assistant by (a) proving the correctness of the encoding and its generator and (b) proving the correctness of the SMT answer, using the autarkic or skeptical approach.

SMTCoq is, in particular, a way to formally establish (b) for the Erdős Discrepancy Conjecture, based on the skeptical approach. In comparison to similar work in checking these proofs in Coq [CS17], there was no need to implement and prove correct a dedicated checker: SMTCoq is generic and efficient enough to encompass such proofs.

The expressivity of SMTCoq makes it possible to formally and efficiently check SAT and SMT proofs coming from any kind of problem. Indeed, SAT and in particular SMT are very expressive and can encode problems coming from multiple areas of mathematics and computer science. We illustrate this aspect by two examples coming from program testing, where one has to generate inputs satisfying the preconditions of a program (*Example 2*), and program proving, where one has to establish properties for all the possible runs of a program (*Example 3*).

Example 2. The problem of automatically generating sorted integer arrays of a given length is a satisfiability problem in the combination of the theories of arrays and integer arithmetic. For instance, for length 4, it can be formulated as such: find a value for the variable `a` (belonging to the sort of arrays) such that:

- `length a = 4`
- $\texttt{a[0]} \leqslant \texttt{a[1]} \wedge \texttt{a[1]} \leqslant \texttt{a[2]} \wedge \texttt{a[2]} \leqslant \texttt{a[3]}$

Example 3. To prove the correctness of the mergesort algorithm on arrays, one should be able to establish[2] that if:

- $\forall\ \texttt{k}_1 \leqslant \texttt{k}_2 < \texttt{k},\ \texttt{a[k}_1\texttt{]} \leqslant \texttt{a[k}_2\texttt{]}$
- $\forall\ \texttt{k}_1 < \texttt{k},\ \texttt{a[k}_1\texttt{]} \leqslant \texttt{x}$
- `a[k] = x`

then:

- $\forall\ \texttt{l}_1 \leqslant \texttt{l}_2 \leqslant \texttt{k},\ \texttt{a[l}_1\texttt{]} \leqslant \texttt{a[l}_2\texttt{]}$

where `a` is an array and $\texttt{k}_1$, $\texttt{k}_2$, `k` and `x` are integers. The validity of this formula can be encoded as the unsatisfiabilty of the negation of the conclusion under the same hypotheses, which corresponds to the following SMT problem: check that the conjunction of

1. $\forall\ \texttt{k}_1 \leqslant \texttt{k}_2 < \texttt{k},\ \texttt{a[k}_1\texttt{]} \leqslant \texttt{a[k}_2\texttt{]}$
2. $\forall\ \texttt{k}_1 < \texttt{k},\ \texttt{a[k}_1\texttt{]} \leqslant \texttt{x}$
3. `a[k] = x`
4. $\texttt{l}_1 \leqslant \texttt{l}_2 \leqslant \texttt{k} \wedge \texttt{a[l}_1\texttt{]} > \texttt{a[l}_2\texttt{]}$

is unsatisfiable, that is to say that there is no concrete instance for the variables `a`, `k`, $\texttt{l}_1$ and $\texttt{l}_2$ that satisfy the four formulas.

2.2 SAT and SMT proof evidence

SMTCoq considers SMT solvers as black boxes that input a SMT problem and output evidence of the satisfiability or unsatisfiability of the problem (or nothing or a

[2] This corresponds to proving the invariant of the merge loop stating that the array is sorted up to a certain point.

partial evidence if it was not able to solve it). The input format has been standardized in the SMT-LIB project [BdMR+10] and is thus common to most state-of-the-art SMT solvers. However, the output format currently differs from one system to another.

If the problem is satisfiable, most provers return as evidence an instance of the variables that satisfies it, called a *model*.

Example 4. There exists a sequence of length 11 of discrepancy 1:

$$\langle 1,-1,-1,1,-1,1,1,-1,-1,1,1\rangle$$

So the SMT encoding of the Erdős Discrepancy Conjecture for $C_0 = 1$ and $l = 11$ is satisfiable and a possible model is:

$$\{x_1 \mapsto 1\,; x_2 \mapsto -1\,; x_3 \mapsto -1\,; x_4 \mapsto 1\,; x_5 \mapsto -1\,; x_6 \mapsto 1\,;$$

$$x_7 \mapsto 1\,; x_8 \mapsto -1\,; x_9 \mapsto -1\,; x_{10} \mapsto 1\,; x_{11} \mapsto 1\}$$

Example 5. The problem of *Example 2* is satisfiable and a possible model is:

$$\{\mathtt{a} \mapsto \boxed{\text{-2}}\boxed{3}\boxed{17}\boxed{42}\}$$

However, if the problem is unsatisfiable, while generic formats have been proposed [Stu09, BFT11], the evidence output by various SMT solvers may differ a lot, particularly in terms of granularity of the proof. To interact with various solvers at small cost, SMTCoq is based on a certificate format inspired by [BFT11] that can represent most existing SMT reasoning, and is also modular to be easily extensible with new theories or proofs with a different level of details[3]. This will make SMTCoq easy to extend at small cost, as detailed in Sec. 3.1.

The idea of this format is to combine independent steps. A *step* can be any deduction that transforms a (possibly empty) set of clauses into a clause that is implied: a step must preserve satisfiability of clauses. A *clause* consists of a disjunction of literals, where a *literal* can be any formula (positive literal) or its negation (negative literal)[4].

Example 6. The problem of *Example 1* consists of 14 clauses:

- 6 clauses of two positive literals each (*e.g.* a positive literal is $x_4 = -1$) for the domain of the sequence;
- 5 clauses of one positive literal each for the sums for $d = 1$;
- 2 clauses of one positive literal each for the sums for $d = 2$;
- 1 clause of one positive literal for the sums for $d = 2$.

The four formulas corresponding to the problem of *Example 3* are four clauses with a single positive literal (which is the formula itself).

[3] This format has been designed together with the veriT [BODF09] proof production engine.

[4] This definition of a clause is more general than the usual one: a literal can be any formula, even containing logical connectives.

The first clause deduced in *Example 7*:
$\neg$ (a[l_1] > a[l_2]) $\vee$ $\neg$ (a[l_1] = a[l_2])
contains two negative literals.

A certificate then combines steps to deduce, in the end, the empty clause from the initial problem. Since the empty clause is unsatisfiable, and each step must preserve satisfiability, it implies that the initial problem is indeed unsatisfiable.

Example 7. The problem of *Example 3* is unsatisfiable and a possible certificate starts with the following steps[5]:

step	deduced clause	premises	justification
5	$\neg$ (a[l_1] > a[l_2]) $\vee$ $\neg$ (a[l_1] = a[l_2])	-	LIA
6	$\neg$ (l_1 = l_2) $\vee$ (a[l_1] = a[l_2])	-	congruence
7	$l_1 \leqslant l_2 \leqslant$ k	4	$\wedge$ projection
8	a[l_1] > a[l_2]	4	$\wedge$ projection
9	$\neg$ (l_1 = l_2)	5, 6, 8	resolution
...	...	...	...

This piece of certificate reads as follows. The first four steps (which are not written) consist of taking the four input clauses (in *Example 3*) as known clauses. Step 5 deduces a new clause from no premise (hence the clause must be a tautology) in the theory of Linear Integer Arithmetic. Step 6 again produces a tautology by congruence of equality with respect to array lookup. Steps 7 and 8 project the $\wedge$ from the fourth input clause, respectively on the left-hand-side and on the right-hand-side. Finally, step 9 produces a new clause from steps 5, 6 and 8 by resolution (meaning that literals appearing both positively and negatively can be simplified out).

Example 8. Similarly, a certificate[6] for the SMT problem corresponding to **Theorem 1** is a proof of the empty clause obtained by combining, using resolution, the initial problem with tautologies in Linear Integer Arithmetic such as:

$$(1 \leqslant x_1 \wedge 1 \leqslant x_2) \Rightarrow x_1 + x_2 > 1$$

The main idea for modularity is that steps need only agree on the representation of formulas, but otherwise can be completely independent from each other. In particular, they independently deal with the various theories: as illustrated in the example, propositional reasoning is represented by resolution and connective steps [Tse70]; equality reasoning, by congruence (and transitivity) steps [BCP11], ...etc. Moreover, they can have a different granularity: resolution is very fine-grained but nothing prevents a step from representing a full SAT solving step.

Notice that results of unsatisfiability are the main use of SMTCoq: as illustrated in *Example 3*, by contradiction, a formula is valid (i.e. always true) if and only if

[5] It corresponds to the certificate given by veriT, stable version of 2016.

[6] The certificate given by veriT, stable version of 2016, contains 178 steps.

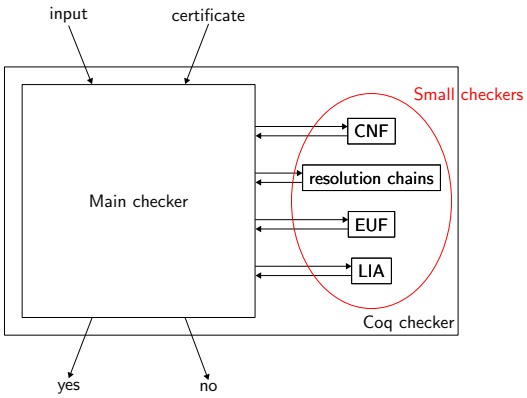

Fig. 1 Architecture of the SMTCoq checker

its negation is unsatisfiable. The remaining of the chapter thus focuses on this part. Nonetheless, checking the satisfiability given a model is much simpler[7].

3 A certified, efficient and modular checker for SMT

3.1 A modular checker

The choice of the certificate format naturally induces a modular checker based on the architecture given in **Fig. 1**.

To each kind of step corresponds what we call a *small checker*, whose task is to check this particular kind of step independently of the other possible steps. The role of the *main checker* is simply to dispatch each step to the corresponding small checkers, and check in the end that the empty clause has been deduced.

Small and main checkers operate over a state $\mathscr{S}$. This state initially contains the problem whose unsatisfiability is to be verified (Steps 1 to 4 in *Example 7*), and is, throughout the process, augmented with the clauses that are deduced by the small checkers (Steps 8 and beyond in *Example 7*). The data-structure used for states will be explained in the next subsection. One crucial aspect is that states can be embedded into Coq terms by a Coq function $[\![\bullet]\!]_\rho$ `:` $\mathscr{S} \to$ `bool` that interprets the state as the conjunction of the interpretation of each formula, and for each formula, interprets each syntactic connective and operator by its Coq counterpart (we refer the reader to [AFG+11] for a detailed explanation of this interpretation function). As standard, the valuation ρ is a mapping of the variables to Coq terms. The property $\forall \rho$, $[\![$ `s` $]\!]_\rho$ `= false` thus means that a state `s` $\in \mathscr{S}$ is unsatisfiable.

[7] In Coq, one simply needs to assign the variables using the model and compute that the formula reduces to the true formula.

As explained in the previous section, a small checker takes as input a (possibly empty) set of clauses and returns a new clause that is implied, in the sense of satisfiability. Concretely, a small checker is thus given by:

- a function `sc : 𝒮 → step → 𝒮` that, given a state and a step, returns the state augmented with the deduced clause;
- a proof that this function preserves satisfiability:

```
sc_ok : ∀ (s:𝒮) (p:step),
        ∀ ρ, ⟦ s ⟧ρ = true ⇒ ⟦ sc s p ⟧ρ = true
```

Adding a new small checker consists of providing such a function and its proof of correctness, independently of existing small checkers, making SMTCoq extensions to new checkers easy.

As the figure suggests, various small checkers have already been implemented for major SMT theories: the initial development of SMTCoq [AFG+11] implemented small checkers for propositional reasoning (via CNF computation and resolution), Equality of Uninterpreted Functions, and Linear Integer Arithmetic, and implementations of small checkers for the theories of bit vectors and arrays have been recently added [EKK+16], confirming the modularity of SMTCoq.

3.2 An efficient checker

For the skeptical approach to be practical, certificate checking must be far cheaper than proof search. This is theoretically the case for most concrete SAT and SMT problems, and SMTCoq has been designed to be as efficient as possible while being implemented and proved correct inside Coq.

The SMTCoq checker has been designed to run in a branch of Coq, called native-coq[8] [BDG11], that in particular lifts in Coq native data-structures such as machine integers and mutable arrays (with history), while preserving soundness. SMTCoq makes intensive use of these data-structures to be efficient.

As an example, formulas are hash-consed using mutable arrays instead of being represented by a standard recursive algebraic datatype: each sub-formula is stored in a cell of the array, and is referred to by its index in the array (which is a machine integer). Literals are encoded as follows: the positive literal associated to the formula at index i is represented by $2 \times i$, and the negative literal, by $2 \times i + 1$. This encoding enjoys the following aspects:

- it is efficiently represented in memory, since it has maximal sharing;
- computations that appear often in SMT checking are really fast: for instance, checking if a literal l is the negation of a literal m is computed by the bitwise operations $l \oplus m = 1$.

[8] The native-coq branch of Coq is progressively being integrated into the main version.

Example 9. The fourth formula of *Example 3* is the formula at index 2 in the following array:

index	0	1	2
sub-formula	$\texttt{l}_1 \leqslant \texttt{l}_2 \leqslant \texttt{k}$	$\texttt{a[l}_1\texttt{]} > \texttt{a[l}_2\texttt{]}$	$\texttt{0} \wedge \texttt{2}$

It corresponds to $\texttt{0} \wedge \texttt{2}$ since $0 = 2 \times 0$ is the positive literal associated to the sub-formula at index $\texttt{0}$ and $2 = 2 \times 1$ is the positive literal associated to the sub-formula at index $\texttt{1}$.

Another place where native data-structures are crucial is in the presentation of states. As detailed above, states start with the initial problem and are "augmented" with new clauses that are deduced. Simply keeping all the clauses is infeasible in practice, since a certificate may produce thousands of them. Thus, a state is a mutable array, whose length is (at least) the number of clauses that are alive at the same time: once all clauses that are implied from a clause c have been deduced, clause c is not useful anymore.

Example 10. In *Examples 3 and 7*, the fourth clause will not be useful anymore after Steps 7 and 8, and can thus be removed from the state.

It is necessary to know in advance, before certificate checking, how many clauses are alive at the same time and in which cell to allocate each clause (in order to overwrite clauses that will not be used anymore). This is done by another nice property of the skeptical method: before checking, certificates can be transformed as needed, and the whole process remains sound even if certificate transformations are not proved correct. Indeed, if certificates are transformed in an unsound way, the checker will not be able to reconstruct a proof.

This principle is widely applied in SMTCoq: upstream from the checker presented in this section, many preprocessors have been implemented (without the need to certify them), that in particular allocate clauses, but also clean certificates from unused steps, ... etc. The checker of **Fig. 1** is thus used in the context of **Fig. 2** (left), where the preprocessors need not be certified.

3.3 Modular link with state-of-the-art SMT solvers

In addition to efficiency, the preprocessing technique allows the use of the Coq checker with any SMT solver without more certification, even if there is no standard for SMT proof witnesses (Sec. 2): it is sufficient to encode proof witnesses into the SMTCoq certificate format, and this encoding does not need to be proved (see **Fig. 2** (right)). Thus, handling a new solver is simply writing an uncertified encoder, and the SMTCoq format is generic enough to welcome state-of-the-art solver's formats.

Encoders for the SAT solvers zChaff and Glucose, as well as the SMT solvers veriT and CVC4 are currently implemented, allowing an efficient check of the answers of all these solvers with the same certified checker.

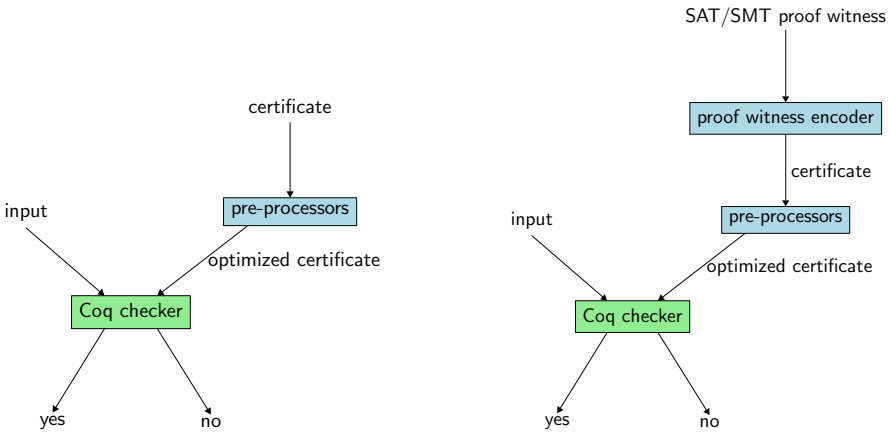

Fig. 2 A certificate can be arbitrarily transformed before being validated by the Coq checker

4 Applications

4.1 Certified validation

The direct application of this checker is to certify answers coming from SAT and SMT solvers: given a SAT or SMT problem and a proof witness provided by a supported solver, the checker can be used to check the unsatisfiability of the input problem, using the proof witness as a hint. This idea is detailed in **Fig. 3** (top left). Note that in this use case, the parser of the SAT/SMT problem must be trusted. If it was to replace the input problem with a trivially unsatisfiable problem, then the checker could easily answer "yes" but it would have certified nothing! In SMTCoq, the parser has not been certified, but it is a very small piece of code that straightforwardly transforms a string into the corresponding SAT/SMT abstract syntax tree (contrary to the encoders and preprocessors that can perform arbitrary transformations). Hence, in this application, if the checker answers "yes", we can be sure that the original problem is unsatisfiable: the checker is correct. Note that, however, if the checker answers "no", we know nothing: the answer coming from the solver may be invalid or incomplete, or the checker may fail to check the proof since it is not shown to be complete. However, it has been tested against a very large benchmark of problems (coming from the SAT and SMT competitions) to make sure that it is complete in practice.

To handle this application, SMTCoq offers two possibilities: the checker can be called from Coq via a dedicated command, or extracted to the OCaml programming language to be used without the need to install Coq. Thanks to the use of native-coq, both methods are really efficient: applications to the benchmarks of the SAT and SMT competitions showed that proof search by the solvers is the bottleneck, but not proof checking by SMTCoq [AFG+11].

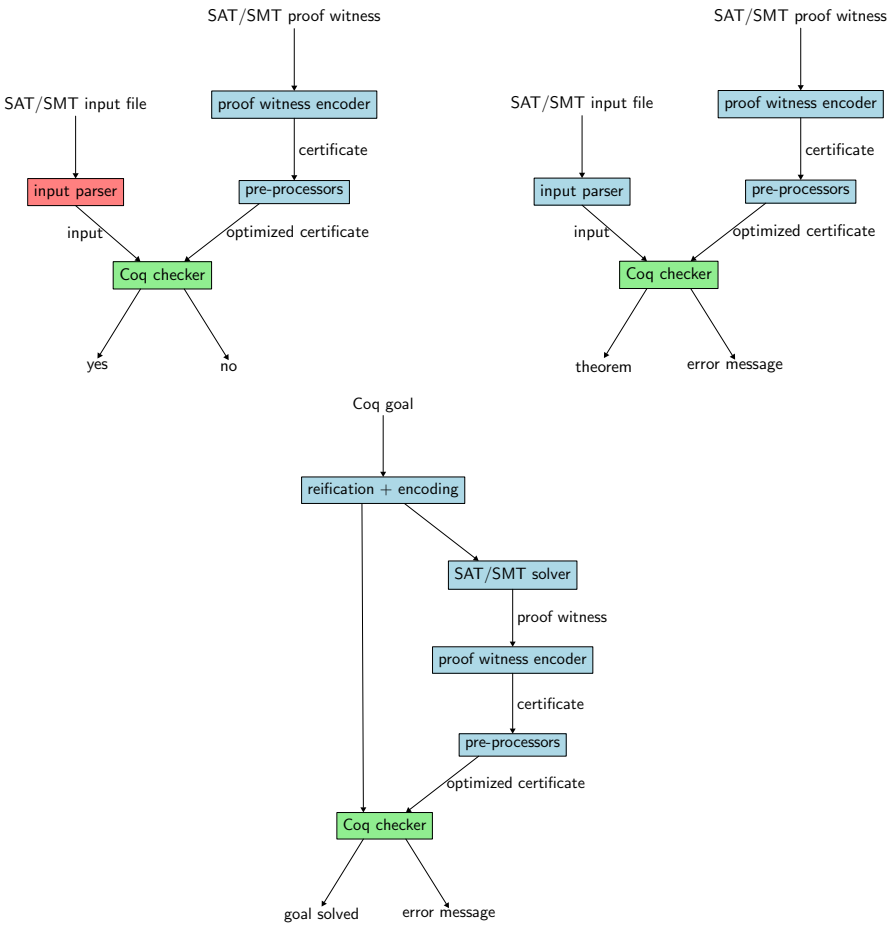

Fig. 3 Applications of the SMTCoq checker

4.2 Theorem import

More generally, the checker can be used to safely import new theorems into Coq (**Fig. 3** (top right)). Given a problem and a proof witness provided by a supported solver, a Coq command generates a new theorem, proved by an application of the correctness of the checker (if the checker fails, then the theorem is not proved and not added to Coq). This theorem can then be used to deduce facts inside Coq. Contrary to the previous subsection, the input parser does not need to be trusted anymore; if it changes the statement of the problem, then a useless theorem may be imported in Coq, but it will not compromise soundness.

As illustrated in Sec. 2, this mechanism has been used to check mathematical proofs containing very large combinatorial results, such as the proof of the Erdős

Discrepancy, by importing the combinatorial part from SAT and proving the remaining as standard in Coq. An ongoing work applies this methodology to other combinatorial proofs such as the Boolean Pythagorean Triples problem [HKM16]. Moreover, SMTCoq is generic enough to import theorems based on SMT solving from many domains.

4.3 Automatic tactics

Finally, as discussed in the introduction, SMTCoq can also be used to automatically solve Coq goals by discharging them to a SAT or a SMT solver and checking the answer (**Fig. 3** (bottom)). The input problem given to the solver comes from a concrete goal that the user wants to prove: the goal is provable if and only if its negation is unsatisfiable. Then, the same process as before is used. Hopefully, the chosen solver returns a proof witness that can be verified by the SMTCoq checker, and if so, the correctness of this latter allows to conclude. Notice that, if the goal is not provable, then its negation is satisfiable and the SAT/SMT solver may return a model that can be used to give a counter-example to the user.

SMTCoq comes with such tactics for most supported provers, which are actually able to solve goals that belong to the combination of theories supported by the provers. Ongoing work [EMT+17] consists of improving the expressivity of these tactics, in particular by encoding goals that are not directly supported by the logic of SMT.

5 Conclusion and perspectives

We have presented SMTCoq, a plug-in for the Coq proof assistant to work in conjunction with external SAT/SMT solvers. SMTCoq has been designed to be modular at many levels (handling new theories and new provers), making it extensible at small cost, and already comes with support for state-of-the-art SAT and SMT solvers. It is distributed as a Coq plug-in that users can enjoy, and is still under active development and expansion. It can be used for various applications ranging from formal proofs of combinatorial problems to day-to-day automation in Coq.

Recently, mathematicians have more frequently used programs and automated provers to establish new results. In addition to the combinatorial problems already presented, two major successes are the proofs of the four-color theorem [Gon07] and of the Kepler conjecture [HAB+15]. SMTCoq is today a generic way to certify some of these proofs, and we argue that, as a Coq plugin, it may become a way to revisit and discover mathematical knowledge by structuring them in a new way [GAA+13].

The certificate approach has been designed first for proof exchange and proof checking. However, we believe that it is more general and will become a standard way of designing reliable proofs in mathematics and computer science.

As presented in the introduction, SMTCoq belongs to a long-term goal to take advantage of mechanized mathematical reasoning which is both automatic and extremely reliable. In this direction, having a single tool is unrealistic: many different proof assistants and automated provers have been designed in the last decades because they all have strong and weak points (regarding automation and reliability, but also expressivity, degree of expertise needed to master them, ... etc.). We rather advocate interoperability between different proof systems, and strongly argue that it relies on *universality* of proofs and mathematical libraries [Mil13, KR16, Sai15]. Proof systems should be able to output evidence of their reasoning, in such a way that it can be combined with proofs coming from other systems, while having the flexibility to have various granularity and underlying logic.

With respect to computer science, many successful tools [FP13, SHK+16] have been designed to prove the correctness of software based on the autarkic approach, and are used in critical industries such as avionics or cryptography [JLB+15, DFK+17]. To reach a larger audience, and even be applicable to most software, the skeptical approach offers a lighter technique that separates the software design from its verification: instead of certifying (possibly complex) algorithms, we certify checkers for their answers. Ongoing works are applying this method to other domains than proof checking, with an objective to generalize certificates rather than having to design a checker for each application. Recent cryptographic technologies can be applied in this direction [PHGR13, FKL16].

Recent works show that the interoperability between systems could be designed in a correct-by-construction approach rather than relying on certificates [BGS18, BM17]. A way of understanding it is that a certificate checker can be turned into the kernel of a certificate producer [BM17], in the same sense as a proof assistant. Ongoing work consists of transforming SMTCoq into a kernel for an SMT solver that could experiment with many proof search strategies without compromising soundness[9]. Further work will lead to an understanding as to how this technique may apply to generic *a posteriori* certification of software discussed in the previous paragraph.

Acknowledgments:

This work would be impossible without the help from past, present and future contributors of SMTCoq. Past and present contributors are Mikaël Armand, Clark Barett, Valentin Blot, Burak Ekici, Germain Faure, Benjamin Grégoire, Guy Katz, Tianyi Liang, Alain Mebsout, Andrew Reynolds, Laurent Théry, Cesare Tinelli and Benjamin Werner. Contributors to the certification of the Erdős Discrepancy Conjecture include Maxime Dénès and Pierre-Yves Strub.

The author thanks Véronique Benzaken and Évelyne Contejean for providing good feedback in the choice of the examples.

[9] This research is supported by Labex DigiCosme (project ANR11LABEX0045DIGICOSME) operated by ANR as part of the program « Investissement d'Avenir» Idex ParisSaclay (ANR11IDEX000302).

The author finally thanks the editors for their invitation to contribute to this book, and the reviewers and editors for their valuable feedback.

References

[ACE+00] Stuart F. Allen, Robert L. Constable, Richard Eaton, Christoph Kreitz, and Lori Lorigo. The nuprl open logical environment. In David A. McAllester, editor, *Automated Deduction - CADE-17, 17th International Conference on Automated Deduction, Pittsburgh, PA, USA, June 17-20, 2000, Proceedings*, volume 1831 of *Lecture Notes in Computer Science*, pages 170–176. Springer, 2000.

[AFG+11] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In Jouannaud and Shao [JS11], pages 135–150.

[ARCT11] Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. The matita interactive theorem prover. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 64–69. Springer, 2011.

[Bar16] Haniel Barbosa. Efficient instantiation techniques in SMT (work in progress). In Pascal Fontaine, Stephan Schulz, and Josef Urban, editors, *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning co-located with International Joint Conference on Automated Reasoning (IJCAR 2016), Coimbra, Portugal, July 2nd, 2016.*, volume 1635 of *CEUR Workshop Proceedings*, pages 1–10. CEUR-WS.org, 2016.

[BB09] B. Brummayer and A. Biere. Fuzzing and Delta-Debugging SMT Solvers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, pages 1–5. ACM, 2009.

[BBG+15] Grzegorz Bancerek, Czeslaw Bylinski, Adam Grabowski, Artur Kornilowicz, Roman Matuszewski, Adam Naumowicz, Karol Pak, and Josef Urban. Mizar: State-of-the-art and beyond. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, volume 9150 of *Lecture Notes in Computer Science*, pages 261–279. Springer, 2015.

[BCD+11] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[BCP11] Frédéric Besson, Pierre-Emmanuel Cornilleau, and David Pichardie. Modular SMT proofs for fast reflexive checking inside coq. In Jouannaud and Shao [JS11], pages 151–166.

[BDG11] Mathieu Boespflug, Maxime Dénès, and Benjamin Grégoire. Full Reduction at Full Throttle. In Jouannaud and Shao [JS11], pages 362–377.

[BdMR+10] Clark Barrett, Leonardo Mendonça de Moura, Silvio Ranise, Aaron Stump, and Cesare Tinelli. The SMT-LIB initiative and the rise of SMT - (HVC 2010 award talk). In Sharon Barner, Ian G. Harris, Daniel Kroening, and Orna Raz, editors, *Hardware and Software: Verification and Testing - 6th International Haifa Verification Conference, HVC 2010, Haifa, Israel, October 4-7, 2010. Revised Selected Papers*, volume 6504 of *Lecture Notes in Computer Science*, page 3. Springer, 2010.

[Bes06] F. Besson. Fast Reflexive Arithmetic Tactics the Linear Case and Beyond. In Thorsten Altenkirch and Conor McBride, editors, *TYPES*, volume 4502 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2006.

[BFT11] F. Besson, P. Fontaine, and L. Théry. 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*, pages 15–26, 2011.

[BG98] Leo Bachmair and Harald Ganzinger. Equational reasoning in saturation-based theorem proving. *Automated deduction—a basis for applications*, 1:353–397, 1998.

[BGS18] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. Proofs in conflict-driven theory combination. In June Andronick and Amy P. Felty, editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 186–200. ACM, 2018.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[BKPU16] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.

[BM17] Sylvain Boulmé and Alexandre Maréchal. Toward Certification for Free! working paper or preprint, July 2017.

[BODF09] Thomas Bouton, Diego Caminha Barbosa De Oliveira, David Déharbe, and Pascal Fontaine. verit: An open, trustable and efficient smt-solver. In Schmidt [Sch09], pages 151–156.

[BW10] S. Böhme and T. Weber. Fast LCF-Style Proof Reconstruction for Z3. In Matt Kaufmann and Lawrence C. Paulson, editors, *ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2010.

[CS17] Luís Cruz-Filipe and Peter Schneider-Kamp. Formally proving the boolean pythagorean triples conjecture. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 509–522. EasyChair, 2017.

[DFK+17] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. Implementing and proving the TLS 1.3 record layer. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 463–482. IEEE Computer Society, 2017.

[dMB07] Leonardo Mendonça de Moura and Nikolaj Bjørner. Efficient e-matching for SMT solvers. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2007.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[dMKA+15] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

[EKK+16] Burak Ekici, Guy Katz, Chantal Keller, Alain Mebsout, Andrew J. Reynolds, and Cesare Tinelli. Extending SMTCoq, a Certified Checker for SMT (Extended Abstract). In Jasmin Christian Blanchette and Cezary Kaliszyk, editors, *Proceedings*

First International Workshop on Hammers for Type Theories, HaTT@IJCAR 2016, Coimbra, Portugal, July 1, 2016., volume 210 of *EPTCS*, pages 21–29, 2016.

[EMT+17] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark W. Barrett. SMTCoq: A Plug-In for Integrating SMT Solvers into Coq. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 126–133. Springer, 2017.

[FKL16] Cédric Fournet, Chantal Keller, and Vincent Laporte. A Certified Compiler for Verifiable Computing. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 268–280. IEEE Computer Society, 2016.

[FMM07] Z. Fu, Y. Marhajan, and S. Malik. zChaff. *Research Web Page. Princeton University, USA,(March 2007) http://www.princeton.edu/ chaff/zchaff.html*, 2007.

[FP13] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 - where programs meet provers. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7792 of *Lecture Notes in Computer Science*, pages 125–128. Springer, 2013.

[GAA+13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A Machine-Checked Proof of the Odd Order Theorem. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, volume 7998 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2013.

[GM05] Benjamin Grégoire and Assia Mahboubi. Proving Equalities in a Commutative Ring Done Right in Coq. In Joe Hurd and Thomas F. Melham, editors, *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2005.

[Gon07] Georges Gonthier. The Four Colour Theorem: Engineering of a Formal Proof. In Deepak Kapur, editor, *Computer Mathematics, 8th Asian Symposium, ASCM 2007, Singapore, December 15-17, 2007. Revised and Invited Papers*, volume 5081 of *Lecture Notes in Computer Science*, page 333. Springer, 2007.

[Gor00] Mike Gordon. From LCF to HOL: a short history. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 169–186. The MIT Press, 2000.

[H+96] John Harrison et al. *Formalized mathematics*. Citeseer, 1996.

[HAB+15] Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason M. Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *CoRR*, abs/1501.02155, 2015.

[HH14] Gérard P. Huet and Hugo Herbelin. 30 years of research and development around coq. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 249–250. ACM, 2014.

[HKM16] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016.

[Hur05] J. Hurd. System Description: The Metis Proof Tactic. *Empirically Successful Automated Reasoning in Higher-Order Logic (ESHOL)*, pages 103–104, 2005.

[JLB+15] Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. A formally-verified C static analyzer. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 247–259. ACM, 2015.

[JS11] Jean-Pierre Jouannaud and Zhong Shao, editors. *Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*, volume 7086 of *Lecture Notes in Computer Science*. Springer, 2011.

[KL15] Boris Konev and Alexei Lisitsa. Computer-aided proof of erdős discrepancy properties. *Artif. Intell.*, 224:103–118, 2015.

[KM96] Matt Kaufmann and J Strother Moore. ACL2: An industrial strength version of Nqthm. In *Computer Assurance, 1996. COMPASS'96, Systems Integrity. Software Safety. Process Security. Proceedings of the Eleventh Annual Conference on*, pages 23–34. IEEE, 1996.

[KR16] Michael Kohlhase and Florian Rabe. QED reloaded: Towards a pluralistic formal library of mathematical knowledge. *J. Formalized Reasoning*, 9(1):201–234, 2016.

[Les11] Stéphane Lescuyer. *Formalizing and Implementing a Reflexive Tactic for Automated Deduction in Coq. (Formalisation et developpement d'une tactique reflexive pour la demonstration automatique en coq)*. PhD thesis, University of Paris-Sud, Orsay, France, 2011.

[Mac95] Donald MacKenzie. The automation of proof: a historical and sociological exploration. *IEEE Annals of the History of Computing*, 17(3):7–29, 1995.

[Mil13] Dale Miller. Foundational proof certificates: making proof universal and permanent. In Alberto Momigliano, Brigitte Pientka, and Randy Pollack, editors, *Proceedings of the Eighth ACM SIGPLAN International Workshop on Logical Frameworks & Meta-languages: Theory & Practice, LFMTP 2013, Boston, Massachusetts, USA, September 23, 2013*, pages 1–2. ACM, 2013.

[Nor09] Ulf Norell. Dependently typed programming in agda. In Andrew Kennedy and Amal Ahmed, editors, *Proceedings of TLDI'09: 2009 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, Savannah, GA, USA, January 24, 2009*, pages 1–2. ACM, 2009.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(). *J. ACM*, 53(6):937–977, 2006.

[ORS92] Sam Owre, John M. Rushby, and Natarajan Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer, 1992.

[PB10] Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011*, volume 2 of *EPiC Series in Computing*, pages 1–11. EasyChair, 2010.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252. IEEE Computer Society, 2013.

[RV02] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2-3):91–110, 2002.

[Sai15] Ronan Saillard. *Typechecking in the lambda-Pi-Calculus Modulo : Theory and Practice. (Vérification de typage pour le lambda-Pi-Calcul Modulo : théorie et pratique)*. PhD thesis, Mines ParisTech, France, 2015.

[Sch09] R. A. Schmidt, editor. *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*. Springer, 2009.

[Sch13] Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.

[SHK+16] Nikhil Swamy, Catalin Hritcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean Karim Zinzindohoue, and Santiago Zanella Béguelin. Dependent types and multi-monadic effects in F. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 256–270. ACM, 2016.

[SLM09] João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Biere et al. [BHvMW09], pages 131–153.

[Stu09] Aaron Stump. Proof checking technology for satisfiability modulo theories. *Electr. Notes Theor. Comput. Sci.*, 228:121–133, 2009.

[Tse70] G. Tseitin. On the Complexity of Proofs in Propositional Logics. In *Seminars in Mathematics*, volume 8, pages 466–483, 1970.

[WDF+09] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Schmidt [Sch09], pages 140–145.