Proof Generation in CDSAT

Maria Paola Bonacina
Dipartimento di Informatica, Università degli Studi di Verona, Verona, Italy
mariapaola.bonacina@univr.it

Proofs of unsatisfiability of a negated conjecture, or, equivalently, proofs of validity of the original conjecture, are an essential output of automated reasoning methods. The transformation, exchange, and standardization of proofs is a key factor for the interoperability of different automated reasoning systems. In theorem proving *proof reconstruction* is the task of extracting a proof from the final state of a derivation after generating the empty clause. While for several theorem proving methods and theorem provers it is a standard task, it is never trivial. For example, in parallel theorem proving with distributed search (see [6] for a recent survey), multiple parallel processes perform inferences and search for a proof. A parallel theorem proving method has *distributed proof reconstruction*, if the process that generates the empty clause can reconstruct the proof from the final state of its database, even if all processes contributed to the proof [4].

In propositional satisfiability (SAT) solving, the *conflict-driven clause learning* (CDCL) procedure generates proofs by *resolution*, because it uses resolution to explain conflicts [28, 26]. SAT solvers apply pre-processing steps and simplification techniques that also need to be accounted for in proofs. Furthermore, proofs generated by SAT solvers are so huge that their definition, generation, and manipulation, involving various proof formats, is an important research topic (e.g., [14]).

Satisfiability modulo theories (SMT) solving represents a middle ground between first-order theorem proving and SAT solving. Initially, *model generation* was emphasized over proof generation in SMT, because the focus was on fragments of first-order theories where satisfiability is decidable, in contrast with first-order logic where satisfiability is not even semidecidable. Over time, SMT solvers have been applied more and more to unsatisfiable inputs, including inputs with quantifiers that may fall outside decidable fragments. SMT solvers have become more similar to theorem provers, and proof generation is crucial also in SMT. Since most SMT solvers are built on top of the CDCL procedure, their proofs are proofs by resolution (with the same caveat as above) with proofs of theory lemmas plugged in as leaves or *black-box sub-proofs* [17, 2, 12, 23, 1].

CDSAT (*Conflict-Driven SATisfiability*) is a paradigm for SMT that innovates SMT solving in several ways [8, 9, 10, 11]. To begin with, CDSAT solves *SATisfiability problems Modulo theories and Assignments* (SMA), which means that the input problem may contain assignments to first-order terms (e.g., $x \leftarrow 3$). The solver has to determine whether there exists a model that satisfies the input formula and also *endorses* the input first-order assignments. A model *endorses* an assignment if it interprets identically left hand side and right hand side of the assignment. For uniformity, CDSAT views also formulæ as assignments to Boolean terms (e.g., $(\neg A \vee B) \leftarrow \mathsf{true}$), and seeks a model that endorses all input assignments. There is a subtle technical difference between, say, $x \leftarrow 3$ and $(x \simeq 3) \leftarrow \mathsf{true}$, since in the latter 3 is a constant symbol of the input language, whereas in the former 3 is a *value*, whose denotation requires a *theory extension*. The generalization of SMT to SMA is relevant to approaching *optimization* problems by solving iteratively SMA problems, where input first-order assignments are used to exclude sub-optimal solutions and induce a convergence towards an optimal one [16].

As the name says, CDSAT is a *conflict-driven* method. In general, a procedure is *conflict-driven* if it proposes a candidate model represented by a series of assignments, and performs non-trivial inferences

Chantal Keller and Mathias Fleury (Eds.): Seventh Workshop
on Proof eXchange for Theorem Proving (PxTP)
EPTCS 336, 2021, pp. 1–4, doi:10.4204/EPTCS.336.1

only to explain a conflict between the current candidate model and the formulæ to be satisfied. Since in CDSAT also formulæ are assignments, the separation between candidate model and formulæ disappears. The state of the computation is simply a sequence of assignments Γ, called a *trail*, which also contains the input assignments. A *conflict* is a subset of Γ that is unsatisfiable.

CDSAT is designed since the start for reasoning in a *union of theories*, with propositional logic as one of the theories. CDSAT lifts the conflict-driven style of CDCL from propositional logic to *conflict-driven reasoning in a union of theories*; and it reduces to CDCL if propositional logic is the sole theory. Prior to CDSAT, MCSAT (*Model-Constructing SATisfiability*) [15, 20, 27, 19, 3, 18] showed how to integrate CDCL with a conflict-driven theory satisfiability procedure (e.g., [22, 21, 13] and see [5] for a survey with more references). CDSAT generalizes MCSAT to generic unions of *disjoint* theories, meaning that their signatures do not share symbols other than equality on shared sorts. CDSAT resembles MCSAT, if there are only propositional logic and another theory with a conflict-driven satisfiability procedure.

For an input problem to be satisfiable in a union of theories, the theories need to agree on which shared terms are equal and on the cardinalities of shared sorts. Beginning with the pioneering work of Nelson and Oppen [25, 24], most approaches to reasoning in a union of theories are defined as *combination schemes* that combine theory satisfiability procedures (see [7] for a survey with more references). These schemes *separate* the original problem into sub-problems, one per theory in the union. The completeness of the combination scheme rests on a *combination lemma* that states which conditions the theories need to satisfy in order to agree on the cardinalities of shared sorts. The satisfiability of the original input in the union of theories is reduced to the satisfiability of every sub-problem in the respective theory, where every sub-problem is conjoined with an *arrangement*. An *arrangement* is a conjunction of equalities and disequalities between shared variables, or shared constants, depending on whether free variables or constants are used to represent shared terms. In a non-deterministic description the arrangement can be guessed. In practice, it is computed by the theory satisfiability procedures. The computation of the arrangement is the only activity where the theory satisfiability procedures cooperate, typically by exchanging equalities between shared variables.

In contrast with this traditional setting, CDSAT is defined as a *transition system* that orchestrates theory-specific *inference systems*, called *theory modules*. An inference system is a set of inference rules, and a theory module is an abstraction of a satisfiability procedure. Every module has its view of the trail, called *theory view*, which contains whatever the module can understand. A theory module can *expand* the trail with an assignment that is a *decision*, encapsulated in the `decide` transition rule of CDSAT, or the result of a *theory inference*, encapsulated in the `deduce` transition rule of CDSAT. Theory inferences are used for propagations, and conflict detection and explanation in the respective theory. The latter applies until the theory conflict surfaces on the trail as a Boolean conflict (e.g., $L \leftarrow \mathsf{true}$ and $\neg L \leftarrow \mathsf{true}$, or, equivalently, $L \leftarrow \mathsf{true}$ and $L \leftarrow \mathsf{false}$). Then the conflict-solving transition rules of CDSAT come into play. Since the Boolean conflict may descend from first-order assignments, the conflict-solving transition rules of CDSAT are designed to handle both Boolean and first-order assignments. It does not matter whether a theory satisfiability procedure is conflict-driven, because CDSAT is conflict-driven for all theories. At the very least, a theory satisfiability procedure can be abstracted into a *black-box theory module*, with an inference rule that detects unsatisfiability by invoking the procedure.

The completeness of CDSAT rests mainly on properties of the theory modules. Every theory module is required to be *complete*, meaning that it can expand its view of the trail if it is not satisfied by a model of its theory. One of the theories in the union needs to be the *leading theory*. A leading theory is aware of all the sorts in the union of theories, and its theory module is aware of all the constraints that the theories may have on the cardinalities of shared sorts. While for the leading theory module it suffices to be complete, any other module needs to be *leading-theory-complete*, meaning that it can expand its view

of the trail if it is not satisfied by a model of its theory that concurs with a model of the leading theory on cardinalities of shared sorts and equality of shared terms.

This description shows that while the traditional combination schemes combine decision procedures as *black-boxes*, CDSAT provides a tighter form of integration at the inference level. This has consequences on *proof generation*. Since the conflict-driven reasoning happens directly in the union of the theories and not only in propositional logic, resolution does not have a dominant role. CDSAT proofs can be rendered as resolution proofs, but this is not a necessary choice. Since the theory satisfiability procedures are not combined as black-boxes, theory sub-proofs are not necessarily black-boxes either. Since CDSAT solves SMA problems, also first-order assignments may appear in proofs. The theory inferences may introduce *new* (i.e., non-input) terms, in order to explain conflicts. Thus, such new terms may appear in proofs.

The powerful abstractions that characterize CDSAT leads to proof generation approaches also based on abstraction. The CDSAT transition system can be made *proof-carrying*, by equipping the transition rules with the capability to generate *abstract proof terms*. During proof reconstruction these proof terms can be translated into different proof formats, including resolution proofs. The resulting proofs can be dispatched to proof checkers or proof assistants, or otherwise manipulated and integrated. Alternatively, CDSAT can adopt the LCF style for proofs, which avoids building proof objects in memory altogether. In LCF style, the prover or solver (e.g., a CDSAT based solver) is built on top of a *trusted kernel* of primitive operations. When the reasoner detects unsatisfiability, the refutation is correct by construction, because otherwise a type error would arise.

Acknowledgements The author thanks Stéphane Graham-Lengrand for their discussions.

References

[1] Haniel Barbosa, Jasmin C. Blanchette, Mathias Fleury & Pascal Fontaine (2020): *Scalable Fine-Grained Proofs for Formula Processing*. *J. of Autom. Reason.* 64(3), pp. 485–550, doi:10.1007/s10817-018-09502-y.

[2] Nikolaj Bjørner & Leonardo de Moura (2008): *Proofs and refutations, and Z3*. In: *IWIL-7*, *CEUR* 418, pp. 123–132.

[3] François Bobot, Stéphane Graham-Lengrand, Bruno Marre & Guillaume Bury (2018): *Centralizing equality reasoning in MCSAT*. In: *SMT-16*.

[4] Maria Paola Bonacina (1996): *On the reconstruction of proofs in distributed theorem proving: a modified Clause-Diffusion method*. *J. of Symb. Comput.* 21(4–6), pp. 507–522, doi:10.1006/jsco.1996.0028.

[5] Maria Paola Bonacina (2018): *On conflict-driven reasoning*. In: *AFM-6*, *Kalpa Publications* 5, EasyChair, pp. 31–49, doi:10.29007/spwm.

[6] Maria Paola Bonacina (2018): *Parallel theorem proving*. In: *Handbook of Parallel Constraint Reasoning*, chapter 6, Springer, pp. 179–235, doi:10.1007/978-3-319-63516-3_6.

[7] Maria Paola Bonacina, Pascal Fontaine, Christophe Ringeissen & Cesare Tinelli (2019): *Theory combination: beyond equality sharing*. In: *Description Logic, Theory Combination, and All That: Essays Dedicated to Franz Baader*, *LNAI* 11560, Springer, pp. 57–89, doi:10.1007/978-3-030-22102-7_3.

[8] Maria Paola Bonacina, Stéphane Graham-Lengrand & Natarajan Shankar (2017): *Satisfiability modulo theories and assignments*. In: *CADE-26*, *LNAI* 10395, Springer, pp. 42–59, doi:10.1007/978-3-319-63046-5_4.

[9] Maria Paola Bonacina, Stéphane Graham-Lengrand & Natarajan Shankar (2018): *Proofs in conflict-driven theory combination*. In: *CPP-7*, ACM, pp. 186–200, doi:10.1145/3167096.

[10] Maria Paola Bonacina, Stéphane Graham-Lengrand & Natarajan Shankar (2020): *Conflict-driven satisfiability for theory combination: transition system and completeness*. *J. of Autom. Reason.* 64(3), pp. 579–609, doi:10.1007/s10817-018-09510-y.

[11] Maria Paola Bonacina, Stéphane Graham-Lengrand & Natarajan Shankar (2021): *Conflict-Driven Satisfiability for Theory Combination: Lemmas, Modules, and Proofs*. *J. of Autom. Reason.* Submitted, pp. 1–54. http://profs.sci.univr.it/~bonacina/cdsat.html.

[12] Maria Paola Bonacina & Moa Johansson (2015): *Interpolation systems for ground proofs in automated deduction: a survey*. *J. of Autom. Reason.* 54(4), pp. 353–390, doi:10.1007/s10817-015-9325-5.

[13] Franz Brauße, Konstantin Korovin, Margarita Korovina & Norbert Müller (2019): *A CDCL-style calculus for solving non-linear constraints*. In: *FroCoS-12*, *LNAI* 11715, Springer, pp. 131–148, doi:10.1007/978-3-030-29007-8_8.

[14] Luís Cruz-Felipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann & Peter Schneider-Kamp (2017): *Efficient certified RAT verification*. In: *CADE-26*, *LNAI* 10395, Springer, pp. 220–236, doi:10.1007/978-3-319-63046-5_14.

[15] Leonardo de Moura & Dejan Jovanović (2013): *A model-constructing satisfiability calculus*. In: *VMCAI-14*, *LNCS* 7737, Springer, pp. 1–12, doi:10.1007/978-3-642-35873-9_1.

[16] Leonardo de Moura & Grant Olney Passmore (2013): *Exact global optimization on demand (Presentation only)*. In: *ADDCT-3*. Available at https://userpages.uni-koblenz.de/~sofronie/addct-2013/.

[17] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto & Alwen Tiu (2006): *Expressiveness+automation+soundness: towards combining SMT solvers and interactive proof assistants*. In: *TACAS-12*, *LNCS* 3920, Springer, pp. 167–181, doi:10.1007/11691372_11.

[18] Stéphane Graham-Lengrand, Dejan Jovanović & Bruno Dutertre (2020): *Solving bitvectors with MCSAT: explanations from bits and pieces*. In: *IJCAR-10*, *LNAI* 12166, Springer, pp. 103–121, doi:10.1007/978-3-030-51074-9_7.

[19] Dejan Jovanović (2017): *Solving nonlinear integer arithmetic with MCSAT*. In: *VMCAI-18*, *LNCS* 10145, Springer, pp. 330–346, doi:10.1007/978-3-319-52234-0_18.

[20] Dejan Jovanović, Clark Barrett & Leonardo de Moura (2013): *The design and implementation of the model-constructing satisfiability calculus*. In: *FMCAD-13*, ACM and IEEE.

[21] Dejan Jovanović & Leonardo de Moura (2013): *Cutting to the chase: solving linear integer arithmetic*. *J. of Autom. Reason.* 51, pp. 79–108, doi:10.1007/s10817-013-9281-x.

[22] Dejan Jovanović & Leonardo de Moura (2012): *Solving non-linear arithmetic*. In: *IJCAR-6*, *LNAI* 7364, Springer, pp. 339–354, doi:10.1007/978-3-642-31365-3_27.

[23] Guy Katz, Clark W. Barrett, Cesare Tinelli, Andrew Reynolds & Liana Hadarean (2016): *Lazy proofs for DPLL(T)-based SMT solvers*. In: *FMCAD-16*, ACM and IEEE, pp. 93–100, doi:10.1109/FMCAD.2016.7886666.

[24] Greg Nelson (1983): *Combining satisfiability procedures by equality sharing*. In: *Automatic Theorem Proving: After 25 Years*, AMS, pp. 201–211, doi:10.1090/conm/029/11.

[25] Greg Nelson & Derek C. Oppen (1979): *Simplification by Cooperating Decision Procedures*. *ACM Trans. on Prog. Lang. and Syst.* 1(2), pp. 245–257, doi:10.1145/357073.357079.

[26] Natarajan Shankar (2009): *Automated deduction for verification*. *ACM Comput. Surv.* 41(4), pp. 507–522, doi:10.1145/1592434.1592437.

[27] Aleksandar Zeljić, Christoph M. Wintersteiger & Philipp Rümmer (2016): *Deciding bit-vector formulas with* mcSAT. In: *SAT-19*, *LNCS* 9710, Springer, pp. 249–266, doi:10.1007/978-3-319-40970-2_16.

[28] Lintao Zhang & Sharad Malik (2003): *Validating SAT solvers using an independent resolution-based checker: practical implementations and other applications*. In: *DATE 2003*, IEEE, pp. 10880–10885, doi:10.5555/789083.1022835.