# Techniques for Verified Propositional SMT Proof Checking

Harry Bryant[1], Andrew Lawrence[2], Monika Seisenberger[1], and Anton Setzer[1]

[1] Swansea University, Dept. of Computer Science, Swansea, United Kingdom
{harry.bryant,m.seisenberger,a.g.setzer}@swansea.ac.uk
[2] Siemens Mobility, Research & Development, Chippenham, United Kingdom
andrew.lawrence@siemens.com

**Abstract**

Safety-critical applications such as railway interlockings require strong guarantees of correctness, making the use of verification and therefore verified proof checking essential. In this paper we focus on techniques for checking Z3 proof logs within Rocq. We build on a formally proven RUP checker and extend it with a parser and preprocessing pipeline that reconstructs Z3's proof steps into a format compatible with the verified checker. Combining these components, we obtain a verified proof checker capable of validating complete propositional Z3 proof logs. We describe the OCaml parser and its preprocessing stages and discuss the limitations of the approach. Moreover, we show how the methodology is designed to scale. The techniques developed provide a foundation for extending verified proof checking to additional SMT-LIB theories, such as integer arithmetic and beyond.

**Introduction.** Railway interlockings are safety-critical systems where software faults can lead to severe consequences. Achieving the assurance needed for SIL4 requires rigorous validation against formally specified safety requirements. While industrial workflows have traditionally relied on extensive manual testing, formal verification tools are increasingly used to detect design errors earlier (e.g. [17, 19, 3]). The Ladder Logic Verifier [9] is one such tool, analysing ladder logic SAT-based interlockings [13, 18] using the Z3 SMT solver [12] to prove unsafe states unreachable by showing that negated safety properties are unsatisfiable. Because Z3 is vast and complex, its unsatisfiability results must be checked independently in high-assurance settings. We addressed this by developing a checker for Z3 proof logs in the Reverse Unit Propagation (RUP) format introduced in 2022. This work builds on the fully verified RUP checker, implemented in Rocq, with additional performance improvements presented by us in [8]. We present the supplementary parser and preprocessing pipeline that reconstructs Z3's SAT-based proof steps into a format compatible with the verified checker. While the checker is fully verified and extractable to OCaml, the parser and preprocessing remains unverified. However, integration with the Ladder Logic Verifier shows that the pipeline enables end-to-end validation of Z3's SAT-based proofs before industrial testing. Our long-term aim is to extend these techniques for additional SMT-LIB theories, such as integer arithmetic.

**Related Work** on proof checking in Rocq includes SMTCoq [2, 14], which checks an earlier form of Alethe proofs, and CoqHammer [11], which reconstructs proofs rather than checking solver outputs. In Isabelle, there is Sledgehammer for proof reconstruction [20], Fleury's verified IsaSAT [15] with integrated checking, and Lammich's SAT-checking [21] which provides both verified integrated and post-hoc checking of LRAT proofs. The latter has a verified DIMACS CNF parser, whereas our (unverified) parser accepts arbitrarily nested propositional formulas. In the future, we will verify more of the parsing stages for correctness, and produce quantitative data against these existing tools to identify optimisations. [1]

**Proof Checker.** The Z3 theorem prover [12] checks the satisfiability of the negated safety property $\neg\varphi$. If unsatisfiable, it produces a proof log for independent validation [26]. We have

[1] Regarding SMT, extensive infrastructure [5, 22, 20] is available for cvc5 [4] with Ethos [10] being the most advanced checker supporting CPC proofs [1, 6] A full account of related work is deferred to a future paper.

developed a checker for the SAT-based Ladder Logic Verifier [9] which is based on Z3. Z3's SAT proof logs use four rule types: Assumption, RUP, Tseitin, and Deletion – with RUP [16] as the core inference rule and Tseitin steps providing CNF encodings via satisfiability-preserving patterns. Our verified checker for these rules, together with its proof architecture, was presented by us in [8]. We prove soundness by logical entailment [24]: any model of the assumptions models the derived conclusions. Two key lemmas establish soundness: (1) If the proof log is correct, all derived clauses follow logically from the assumptions. (2) connects this to unsatisfiability:

(1) IsTrue(ZProofCheck $p$) $\rightarrow$ EntailsListZCl(ZProof2Assumption $p$, ZProof2ConclusionOpt $p$).
(2) ZProofCheckUnsat $p$ = true $\rightarrow$ UnsatZCl(ZProof2Assumption $p$).

Together, these ensure whenever the checker declares the Z3 proof log correct, the original assumptions are unsatisfiable. This verified checker was then extracted to OCaml for execution.

**Parser.** To interface Z3 SAT proof logs with the verified datatypes, we implement an OCaml lexer–parser–preprocessor pipeline using `ocamllex` and `ocamlyacc`/Menhir [23]. The lexer recognises the Z3 rule keywords, punctuation, and atom names, which are interned to integer identifiers for constant-time comparison. The parser constructs a raw AST of clauses and translates them into the inductive datatypes used by the verified Rocq checker: formulas become `z3_Formula`, lists are folded into `listZ3_Formula` and each proof item is mapped to a canonical `zProofItem`. Assumption, RUP, and Deletion steps translate directly whilst Tseitin steps require additional processing. Z3 may emit full Tseitin clauses, which match one of the eight satisfiability-preserving patterns proved correct in Rocq. For efficiency, it produces reduced Tseitin clauses, where literals are omitted once their negation has already been derived. The `tseitin_sorter` analyses each `(f_list, g_list)` pair by attempting to match `f_list` against these verified patterns (implication, negation, and `And`/`Or` introduction and elimination). If a pattern matches `f_list`, the step is a reduced Tseitin step whenever `g_list` is a subset of the corresponding `f_list` tautology and every element omitted from the tautology appears negated among the conclusions. If `g_list` is identical to the `f_list` tautology, it is a standard Tseitin step. The `step_processor` then maps the result into the checker's representation. This ensures that all proof items delivered to the verified Rocq checker are consistent with the proven inference rules. We have validated the pipeline on examples confirming that Z3 SAT-based proof logs are faithfully reconstructed. Furthermore, the parsing and classification mechanism is readily adaptable to additional SMT-LIB theories - such as integers, arrays, and bit-vectors - as defined by the standard [25], enabling future development beyond purely propositional reasoning. A longer-term goal is to obtain a fully verified proof checker extracted to C++, allowing integration with industrial toolchains that require native C++ implementations. Our current OCaml parser and preprocessing pipeline act as a reference model. The plan is to reimplement these components in C++ while using the verified Rocq core as the shared foundation. To support this, we are currently exploring Rocq-to-C++ extraction methods, such as Crane [7].

**Limitations and Conclusion.** Our approach verifies the checking algorithm in Rocq, ensuring that whenever the checker accepts a proof item, the encoded assumptions imply the encoded conclusions. Thus, provided the proof object supplies the correct assumptions and conclusions, the checker guarantees sound inference. What lies outside the verification boundary is whether the assumptions and conclusions extracted from a Z3 proof are exactly those intended by Z3, this concerns the parser and preprocessing stages rather than the checker itself. The checker does not depend on any particular parsing strategy, only on receiving a well-formed proof object whose assumptions and conclusions match the original proof (for unsatisfiable proofs, only the assumptions matter). Establishing this correspondence is beyond our formal development, but it can be tested in practice. Future work aims to reduce the trusted computing base by moving more preprocessing into Rocq, leaving only raw proof parsing unverified.

# References


[1] Leni Aniva, Haniel Barbosa, Clark Barrett, Hanna Lachnitt, Daniel Larraz, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Hans-Jörg Schurr, Cesare Tinelli, Amalee Wilson, and Yoni Zohar. cvc5 at the SMT competition 2024. In *Proceedings of the SMT Competition 2024*, pages 25–36, Iowa City, USA, 2024. CEUR-WS.org. https://doi.org/10.5281/zenodo.11581520.

[2] Michael 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 Jean-Pierre Jouannaud and Zhong Shao, editors, *Certified Programs and Proofs*, pages 135–150, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[3] Madhusree Banerjee, Victor Cai, Sunitha Lakshmanappa, Andrew Lawrence, Markus Roggenbach, Monika Seisenberger, and Thomas Werner. A Tool-Chain for the Verification of Geographic Scheme Data. In Birgit Milius, Simon Collart-Dutilleul, and Thierry Lecomte, editors, *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*, pages 211–224, Cham, 2023. Springer Nature Switzerland.

[4] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 415–442, Cham, 2022. Springer International Publishing.

[5] Haniel Barbosa, Andrew Reynolds, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Arjun Viswanathan, Scott Viteri, Yoni Zohar, Cesare Tinelli, and Clark Barrett. Flexible proof production in an industrial-strength SMT solver. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning*, pages 15–35, Cham, 2022. Springer International Publishing.

[6] Nikolaj Bjørner, Pascal Fontaine, Aaron Stump, and the SMT-COMP Organizing Team. SMT-COMP 2025 results. https://smt-comp.github.io/2025/results/, 2025. Accessed 21 January 2026.

[7] Bloomberg. Crane: A C++ extraction tool for Rocq. https://github.com/bloomberg/crane. Accessed: 2026-01-19.

[8] Harry Bryant, Andrew Lawrence, Monika Seisenberger, and Anton Setzer. Formal assurance for railway interlockings: Verifying Z3 SAT proofs with Tseitin in Rocq. In *RocqPL 2026: Workshop on Rocq for Programming Languages, colocated with POPL 2026*, 2026. https://popl26.sigplan.org/details/rocqpl-2026-papers/3/Formal-Assurance-for-Railway-Interlockings-Verifying-Z3-SAT-Proofs-with-Tseitin-in-Rocq.

[9] Simon Chadwick, Phillip James, Markus Roggenbach, and Tom Werner. Formal methods for industrial interlocking verification. In *2018 International Conference on Intelligent Rail Transportation (ICIRT)*, pages 1–5, Piscataway, NJ, USA, 2018. IEEE.

[10] cvc5 Development Team. Ethos: A flexible and efficient proof checker for SMT solvers. https://github.com/cvc5/ethos, 2023. Accessed: December 12, 2025.

[11] Lukasz Czajka and Cezary Kaliszyk. Hammer for Coq: Automation for Dependent Type Theory. In *International Conference on Interactive Theorem Proving*, pages 425–442, Cham, Switzerland, 2018. Springer, Springer.

[12] Leonardo De Moura and Nikolaj Bjørner. Z3: An Efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, page 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] Nicolas DeGuglielmo, Saurav Basnet, and Douglas Dow. Introduce ladder logic and programmable logic controller (PLC). In *Proceedings of the 2020 IEEE Applied Systems and Engineering Envi-*

*ronment for Emerging Nations Conference (ASEE-NE)*, pages 1–5, Piscataway, NJ, USA, 10 2020. IEEE.

[14] SMTCoq Developers. SMTCoq: Communication between Coq and SAT/SMT solvers. https://smtcoq.github.io/, 2025. Accessed: 2025-06-26.

[15] Mathias Fleury and Peter Lammich. A More pragmatic CDCL for IsaSAT and targetting LLVM (short paper). In Brigitte Pientka and Cesare Tinelli, editors, *Automated Deduction – CADE 29*, pages 207–219, Cham, 2023. Springer Nature Switzerland.

[16] Allen van Gelder. Verifying RUP Proofs of Propositional Unsatisfiability: Have Your Cake and Eat It Too, 2008. In Proceedings of 10th International Symposium on Artificial Intelligence and Mathematics (ISAIM'08).

[17] Anne Haxthausen and Jan Peleska. Efficient development and verification of safe railway control software. In Cacilie Reinhardt and Klaus Shroeder, editors, *Railways: Types, Design and Safety Issues*, pages 127–148. Nova Science Publishers, Hauppauge, NY, USA, 2013.

[18] International Electrotechnical Commission. *IEC 61131-3:2013 - Programmable controllers – Part 3: Programming languages*, 3.0 edition, 2 2013. Accessed: 2025-06-10.

[19] Phillip James, Andy Lawrence, Faron Moller, Markus Roggenbach, Monika Seisenberger, Anton Setzer, Karim Kanso, and Simon Chadwick. Verification of solid state interlocking programs. In Steve Counsell and Manuel Núñez, editors, *Software Engineering and Formal Methods*, pages 253–268, Cham, 2014. Springer International Publishing.

[20] Hanna Lachnitt, Mathias Fleury, Haniel Barbosa, Jibiana Jakpor, Bruno Andreotti, Andrew Reynolds, Hans-Jörg Schurr, Clark Barrett, and Cesare Tinelli. Improving the SMT proof reconstruction pipeline in Isabelle/HOL. In Yannick Forster and Chantal Keller, editors, *16th International Conference on Interactive Theorem Proving (ITP 2025)*, volume 352 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:22, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[21] Peter Lammich. Fast and verified UNSAT certificate checking. In Christoph Benzmüller, Marijn J.H. Heule, and Renate A. Schmidt, editors, *Automated Reasoning*, pages 439–457, Cham, 2024. Springer Nature Switzerland.

[22] Abdalrhman Mohamed, Tomaz Mascarenhas, Harun Khan, Haniel Barbosa, Andrew Reynolds, Yicheng Qian, Cesare Tinelli, and Clark Barrett. LEAN-SMT: An SMT tactic for discharging proof goals in Lean. In Ruzica Piskac and Zvonimir Rakamarić, editors, *Computer Aided Verification*, pages 197–212, Cham, 2025. Springer Nature Switzerland.

[23] OCaml Development Team. *OCaml Manual: Lexer and Parser Generators (ocamllex, ocamlyacc)*, 2025. Accessed: 2025-12-04.

[24] Pieter A. M. Seuren. Logic and entailment. In *The Logic of Language: Language From Within Volume II*, page 85=87. Oxford University Press, 10 2009.

[25] The SMT-LIB Initiative. SMT-LIB standard: Available theories. https://smt-lib.org/theories.shtml. Accessed: 2026-01-19.

[26] Z3 Development Team. Inference logs and proofs, Retrieved 16 June 2025. https://microsoft.github.io/z3guide/programming/Proof%20Logs/.