Formal Assurance for Railway Interlockings: Verifying Z3 SAT Proofs with Tseitin in Rocq

Harry Bryant*
harry.bryant@swansea.ac.uk
Swansea University, Dept. of Computer Science,
Swansea, United Kingdom

Andrew Lawrence
andrew.lawrence@siemens.com
Siemens Mobility
Chippenham, United Kingdom

Monika Seisenberger
m.seisenberger@swansea.ac.uk
Swansea University, Dept. of Computer Science,
Swansea, United Kingdom

Anton Setzer
a.g.setzer@swansea.ac.uk
Swansea University, Dept. of Computer Science,
Swansea, United Kingdom

Abstract

We present a method for verifying in the theorem prover Rocq SAT proofs of railway interlockings generated by Z3 using the RUP format and Tseitin Transformations. We outline the checking procedures for RUP and the supporting Tseitin Transformation rules. This includes steps to prove their correctness and the extraction of a fully verified checker. We prove Tseitin steps correct by turning them into tautologies using a novel technique. The verification process is crucial in the context of railway interlockings, where ensuring logical soundness and a high level of safety are essential.

CCS Concepts: • **Software and its engineering → Formal language definitions**; **Formal software verification**; • **Mathematics of computing → Solvers**; • **Theory of computation → Program verification**; **Logic and verification**; **Verification by model checking**.

Keywords: Railway Verification, Z3, RUP proofs, Reverse Unit Propagation, Tseitin transformation, Rocq, SAT solving

ACM Reference Format:
Harry Bryant, Andrew Lawrence, Monika Seisenberger, and Anton Setzer. 2026. Formal Assurance for Railway Interlockings: Verifying Z3 SAT Proofs with Tseitin in Rocq. In *Proceedings of Rocq for Programming Languages 2026 (RocqPL '26)*. ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Railway systems are safety-critical systems, where software failures can have catastrophic consequences, including loss of life, environmental harm, and economic disruption. To ensure safety and regulatory compliance, rigorous validation against formalised safety requirements is essential. While traditionally reliant on extensive manual testing, formal verification tools are increasingly used to detect issues earlier in development (see e.g. [4, 20, 22]). One tool is the Ladder Logic Verifier [10], which targets ladder logic [14, 21] interlockings. It uses Z3, a leading SMT solver, to prove the unreachability of unsafe states by demonstrating the unsatisfiability of negated safety properties. However, due to the internal complexity of Z3, independent validation of its results is crucial. To support high-assurance certification (e.g., SIL4), we present a formally verified proof checker for Z3's Reverse Unit Propagation (RUP) format, introduced in 2022. This format replaces resolution-based proofs relying on CNF representations supported by the Tseitin Transformation. Our verified checker, extracted from Rocq, validates Z3's SAT-based unsatisfiability proofs on results produced by the Ladder Logic Verifier, ensuring that interlockings only proceed to industrial testing after both verification and proof validation are complete (Fig. 1). Prior work on interacting with SAT/SMT solvers in Rocq includes SMTCoq [2, 15], involving a certified proof checker for the (former) Alethe format, and Coq-Hammer [12], mainly using proof reconstruction, based on hints obtained from automated tools. Our approach is different as it uses program extraction from the correctness proof in Rocq, and works well with Z3. Other current work on proof certification, in Isabelle and Lean, involves work by Fleury [16] and Lammich [24] for SAT and certification for CVC5[5–7] for SMT. A detailed comparison with the above and other related work will be given in an extended version of this paper.

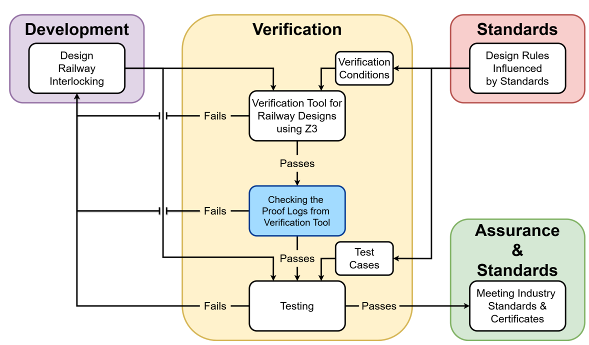

Figure 1. Proposed railway interlocking design methodology: The interlocking and safety properties are translated into SMTLIB[8] for analysis with Z3. If Z3 returns unsatisfiability, our proof checker validates the corresponding proof log. Only then does industrial testing proceed.

2 Z3 SAT Proof Logs of Unsatisfiability

The Z3 SMT solver [13] verifies by checking the satisfiability of the negated safety property $\neg\varphi$, combined with the system model [23]. If unsatisfiable, φ holds, and Z3 produces a proof log [32] for independent validation, otherwise, a counterexample is

RocqPL '26, Rennes, France

ACM ISBN 978-1-4503-XXXX-X/2026/01
https://doi.org/XXXXXXX.XXXXXXX

returned [1, 27, 30]. Both answers are limited by the resource constraints provided. The SAT-based Ladder Logic Verifier [10] uses Z3 for inductive verification [23] and bounded model checking [3, 11]. Z3's SAT proof logs consist of four rule types: Assumption, RUP [17, 18] (clause derivation), Tseitin (CNF encoding) [26, 31], and Deletion (efficiency). A checker for proof logs has been developed and proven correct in Rocq, and extracted to OCaml.

2.1 Reverse Unit Propagation Proof Steps

Reverse Unit Propagation (RUP) is the foundation of the new Z3 proof format. RUP was according to Oe and Stump [28] introduced by Van Gelder [17, 18] in 2008, based on Goldberg and Novikov [19]. Resolution is still the foundation; however, RUP allows efficient checking while addressing the issue that resolution proofs can be too long to store feasibly. The RUP rule expresses that a clause is derivable if, when adding its negation to the assumptions, a contradiction is derived by unit propagation, without storing full resolution proofs. RUP operates on formulas in conjunctive normal form (CNF), where each clause is a disjunction of literals. Z3 translates non-CNF formulas using the Tseitin transformation [26, 31].

To check a RUP inference, the negation of the derived clause is added to the current set of clauses, and unit propagation is applied to derive a contradiction. The clause set is split into unit clauses (length 1) and non-unit clauses (length ≥ 2). We repeatedly select a unit clause and resolve it against the others: if a clause contains the same literal, it is removed; if it contains the negated literal, we remove that literal from the clause. If this process produces an empty clause, the inference is valid. If no contradiction is found after all unit clauses are processed, the inference is not valid.

2.2 Tseitin Transformation Proof Steps

The Z3 proof format includes Tseitin [26, 31] steps, which introduce auxiliary clauses to convert non-CNF formulas into CNF. This transformation translates conjunction, disjunction, implication, and negation into clauses, preserving satisfiability. For example, the formula $a \land b$ is related to its sub-formulas a and b by adding the three clauses $[\neg Pa, \neg Pb, P]$, $[Pa, \neg P]$, $[Pb, \neg P]$, where P, Pa, Pb are new propositional variables representing $a \land b$, a, b respectively. If we replace P, Pa, Pb by the formulas they represent, these clauses become tautologies. Z3 implements eight patterns for these transformations, covering the introduction and elimination rules.

We validate Tseitin steps by proving in Rocq that all eight transformation patterns are tautologies. Consequently, any Tseitin step that matches one of these patterns preserves the satisfiability of the derived clauses. Z3 supports `And` and `Or` expressions with an arbitrary number of conjuncts or disjuncts. When applying `And` elimination or `Or introduction`, a specific element is selected by its index, and we need to check that this index falls within the main formula. In some cases, Z3 produces Reduced Tseitin steps by omitting literals where their negation has been derived in prior proof steps, resulting in clauses c that are subsets of the original tautology d. A Reduced Tseitin Step deriving c using tautology d and already derived clauses f is valid, if (1) d is a tautology, i.e. $\forall m\ m \models d$; (2) $c \subset d$; (3) $\forall l \in (d \setminus c).[\neg l] \in f$.

3 Verifying the Z3 Proof Rules to be Correct

To demonstrate the soundness of our checker, we verify it within Rocq by proving that entailment is preserved throughout the proof log. Therefore, if we have a proof log of unsatisfiability, it will be confirmed as unsatisfiable. We show that if all steps are valid, the initial assumptions entail the derived conclusions. The function `ZProofCheck` validates each proof log step based on its type.

- RUP steps: Check whether unit propagation on the negated clause derives the empty clause ($\emptyset$).
- Tseitin steps: Verify that the clause matches one of the eight predefined patterns, which are tautologies by construction.
- Assumption steps: No additional checks are required, as these represent initial clauses from the SMT-LIB script.

`ZProof2Assumption` constructs the list of clauses marked as assumptions from the original Z3 SMT-LIB script. Conclusions are computed by `ZProof2ConclusionOpt`, which aggregates all clauses up to the current step. Assumptions are added to both lists as they trivially entail themselves. RUP steps add only to conclusions, since unit propagation ensures correctness by deriving a contradiction. Two lemmas guarantee soundness: if the RUP check succeeds, the assumptions involve $\emptyset$. Then if assumptions entail conclusions before a valid RUP step, they also entail the extended set. Similarly, Tseitin and Reduced Tseitin steps add their resulting clause to the conclusions, as all eight patterns are proven tautologies and satisfiability is preserved. Therefore, if the proof log is valid, assumptions entail all conclusions.

Lemma 3.1 (Correctness of Z3ProofCheck)**.**
$\forall (p : ZProof),\ IsTrue(ZProofCheck(p)) \rightarrow$
$EntailsListZCl\big(ZProof2Assumption(p),\ ZProof2ConclusionOpt(p)\big).$

4 Extracting a Z3 SAT Checker

The verified Z3 SAT checker is extracted from Rocq [29] using Coq's extraction mechanism [25] to OCaml. We have implemented a parser to read Z3 proof logs, mapping steps to the corresponding Rocq datatypes. Preprocessing handles Tseitin steps by matching them to one of eight patterns or identifying reduced steps where antecedents and resulting clauses differ. The complete OCaml checker and RUP script are available on GitHub [9]. Proofs are processed sequentially, validating each step, and reporting the first failure if encountered. We evaluated it on an industrial interlocking with 75,000 propositional variables and 12,000 ladder logic rungs. Proof logs were generated using Inductive Verification (IV) and Bounded Model Checking (BMC). IV logs contained approximately 29,000 steps, while BMC logs exceeded 500,000, reflecting its higher computational cost. All logs were successfully validated, with IV completing significantly faster, averaging 45 seconds across 6 logs, compared to BMC's average of 13 hours 14 minutes across 13 logs.[1]

5 Conclusion

A fully verified SAT proof checker for railway interlockings has been developed in Rocq and extracted for testing. We plan to investigate parallelism to process multiple sections of a proof log simultaneously. Another improvement is tracking unit clauses throughout verification, accelerating both RUP checks and subset validation for Reduced Tseitin steps. Our technique used to show the correctness of the Tseitin transformations provides a template for adding additional Z3 proof rules. The goal is to support all SMTLIB-compatible formalisations, enabling verification of complex standards, like ETCS, involving higher-order logic and numeric types.

[1]Tests ran on a machine with 128 64-core processors at 2194.443 MHz.

References

[1] Sahel Alouneh, Sa'Ed Abed, Mohammad Alshayeji, and Raed Mesleh. 2019. A comprehensive study and analysis on SAT-solvers: advances, usages and achievements. *Artificial Intelligence Review* 52 (12 2019), 2575–2601. doi:10.1007/s10462-018-9628-0

[2] Michael Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. 2011. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In *Certified Programs and Proofs*, Jean-Pierre Jouannaud and Zhong Shao (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 135–150.

[3] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, Cambridge, MA 02142.

[4] Madhusree Banerjee, Victor Cai, Sunitha Lakshmanappa, Andrew Lawrence, Markus Roggenbach, Monika Seisenberger, and Thomas Werner. 2023. A Tool-Chain for the Verification of Geographic Scheme Data. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*, Birgit Milius, Simon Collart-Dutilleul, and Thierry Lecomte (Eds.). Springer Nature Switzerland, Cham, 211–224. doi:10.1007/978-3-031-43366-5_13

[5] 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. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, Dana Fisman and Grigore Rosu (Eds.). Springer International Publishing, Cham, 415–442.

[6] Haniel Barbosa, Jasmin Christian Blanchette, Mathias Fleury, Pascal Fontaine, and Hans-Jörg Schurr. 2019. Better SMT Proofs for Easier Reconstruction. In *AITP 2019 - 4th Conference on Artificial Intelligence and Theorem Proving*. University of Innsbruck / EasyChair, Obergurgl, Austria, 1–16. https://hal.science/hal-02381819

[7] 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. 2022. Flexible Proof Production in an Industrial-Strength SMT Solver. In *Automated Reasoning*, Jasmin Blanchette, Laura Kovács, and Dirk Pattinson (Eds.). Springer International Publishing, Cham, 15–35.

[8] Clark Barrett, Aaron Stump, and Cesare Tinelli. 2010. *The SMT-LIB Standard: Version 2.0*. https://smt-lib.org.

[9] Harry Bryant, Andrew Lawrence, Monika Seisenberger, and Anton Setzer. 2025. Verification of Z3 RUP Proofs in Coq-Rocq and Agda. https://github.com/HarryBryant99/Verification-of-Z3-RUP-Proofs-in-Coq-Rocq-and-Agda. Accessed: 2025-06-10.

[10] Simon Chadwick, Phillip James, Markus Roggenbach, and Tom Werner. 2018. Formal Methods for Industrial Interlocking Verification. In *2018 International Conference on Intelligent Rail Transportation (ICIRT)*. IEEE, Piscataway, NJ, USA, 1–5. doi:10.1109/ICIRT.2018.8641579

[11] Edmund Clarke, Orna Grumberg, and Doron Peled. 2001. *Model Checking*. MIT Press, Cambridge, MA 02142, USA.

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

[13] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Hungary) *(TACAS'08/ETAPS'08)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer-Verlag, Berlin, Heidelberg, 337–340. https://dl.acm.org/doi/abs/10.5555/1792734.1792766

[14] Nicolas DeGuglielmo, Saurav Basnet, and Douglas Dow. 2020. Introduce Ladder Logic and Programmable Logic Controller (PLC). In *Proceedings of the 2020 IEEE Applied Systems and Engineering Environment for Emerging Nations Conference (ASEE-NE)*. IEEE, Piscataway, NJ, USA, 1–5. doi:10.1109/ASEENE51624.2020.9292646

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

[16] Mathias Fleury and Peter Lammich. 2023. A More Pragmatic CDCL for IsaSAT and Targetting LLVM (Short Paper). In *Automated Deduction – CADE 29*, Brigitte Pientka and Cesare Tinelli (Eds.). Springer Nature Switzerland, Cham, 207–219.

[17] Allen van Gelder. 2008. Verifying RUP Proofs of Propositional Unsatisfiability. https://users.soe.ucsc.edu/~avg/ProofChecker/Documents/proofs-isaim08-trans.pdf Slides of a talk given at ISAIM'08.

[18] Allen van Gelder. 2008. Verifying RUP Proofs of Propositional Unsatisfiability: Have Your Cake and Eat It Too. https://users.soe.ucsc.edu/~avg/ProofChecker/Documents/proofs-isaim08-long.pdf In Proceedings of 10th International Symposium on Artificial Intelligence and Mathematics (ISAIM'08).

[19] Eugene Goldberg and Yakov Novikov. 2003. Verification of Proofs of Unsatisfiability for CNF Formulas. In *2003 Design, Automation and Test in Europe Conference and Exhibition*. IEEE Computer Society, Los Alamitos, CA, USA, 886–891. doi:10.1109/DATE.2003.1253718

[20] Anne Haxthausen and Jan Peleska. 2013. Efficient Development and Verification of Safe Railway Control Software. In *Railways: Types, Design and Safety Issues*, Cacilie Reinhardt and Klaus Shroeder (Eds.). Nova Science Publishers, Hauppauge, NY, USA, 127–148.

[21] International Electrotechnical Commission. 2013. *IEC 61131-3:2013 - Programmable controllers – Part 3: Programming languages* (3.0 ed.). https://webstore.iec.ch/en/publication/4552 Accessed: 2025-06-10.

[22] Phillip James, Andy Lawrence, Faron Moller, Markus Roggenbach, Monika Seisenberger, Anton Setzer, Karim Kanso, and Simon Chadwick. 2014. Verification of Solid State Interlocking Programs. In *Software Engineering and Formal Methods*, Steve Counsell and Manuel Núñez (Eds.). Springer International Publishing, Cham, 253–268.

[23] Phillip James, Andy Lawrence, Faron Moller, Markus Roggenbach, Monika Seisenberger, Anton Setzer, Karim Kanso, Simon Chadwick, and Swansea Railway Verification Group. 2014. *Verification of Solid State Interlocking Programs*. Vol. 8368. Springer-Verlag London, London, United Kingdom. 253–268 pages.

[24] Peter Lammich. 2024. Fast and Verified UNSAT Certificate Checking. In *Automated Reasoning*, Christoph Benzmüller, Marijn J.H. Heule, and Renate A. Schmidt (Eds.). Springer Nature Switzerland, Cham, 439–457.

[25] Pierre Letouzey. 2008. Extraction in Coq, an Overview. In *Logic and Theory of Algorithms: 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008 Proceedings 4*, Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe (Eds.). Springer-Verlag, Heidelberg, Germany, 359–369. doi:10.1007/978-3-540-69407-6_39

[26] Marius Minea. 2024. Conjunctive Normal Form: Tseitin Transform. https://people.cs.umass.edu/~marius/class/h250/lec2.pdf H250: Honors Colloquium - Introduction to Computation.

[27] Julien Murzi and Lionel Shapiro. 2015. Validity and Truth-Preservation. In *Unifying the Philosophy of Truth*, Theodora Achourioti, Henri Galinon, and José Martinez (Eds.). Springer, 233 Spring Street, New York, NY 10013, USA, 431–459.

[28] Duckki Oe and Aaron Stump. 2011. Combining a logical framework with an RUP checker for SMT proofs. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-80.html In informal proceedings of SMT 2011. Technical Report No UCB/EECS-2011-80, EECS Department, University of California, Berkeley.

[29] Rocq Development Team. 2025. Rocq. https://rocq-prover.org/. Accessed: 2025-05-09.

[30] Markus Roggenbach, Antonio Cerone, Bernd-Holger Schlingloff, Gerardo Schneider, and Siraj Ahmed Shaikh. 2021. *Formal Methods for Software Engineering Languages, Methods, Application Domains* (1 ed.). Springer International Publishing, Heidelberg, Germany. 192–207 pages.

[31] Grigori S. Tseitin. 1983. On the Complexity of Derivation in Propositional Calculus. In *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, Jörg H. Siekmann and Graham Wrightson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 466–483. doi:10.1007/978-3-642-81955-1_28

[32] Z3 Development Team. 2025. Inference logs and proofs. https://microsoft.github.io/z3guide/programming/Proof%20Logs/ https://microsoft.github.io/z3guide/programming/Proof%20Logs/.

Received 31 October 2025