Verifying Z3 RUP Proofs with the Interactive Theorem Provers Coq/Rocq and Agda

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

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

Ensuring the correctness of safety-critical systems, such as in railway control systems, is as important as ever. To achieve this, machine-assisted theorem proving is increasingly used in the railway domain. Tools such as Z3 [12] are employed to formally prove that such systems meet the required standards (see e.g. the papers by our group [1, 7]). However, any of these solvers may have flaws or implement optimisations that produce incorrect results. To increase trust, proof checking offers an independent check of the Z3 output. We are developing a verified proof checker for Z3 using its new Reverse Unit Propagation (RUP) format. As a first step, we have focused on propositional formulae in conjunctive normal form (CNF) [17]. The new RUP proof format was introduced to the Z3 theorem prover in September 2022, replacing resolution [2]. Proof checking for other proof formats for SAT and SMT solving has been performed in, e.g., [3, 5, 4, 8, 26, 11, 27, 13, 9].

The notion of a RUP proof was introduced by van Gelder [15, 14] in 2008. It addresses the issue that resolution proofs can be too lengthy to store feasibly while still allowing efficient checking. The underlying concept is proof verification by Goldberg and Novikov [16], where unit propagation checks unsatisfiability without storing full resolution proofs. Van Gelder refined this into RUP, requiring each derived clause to cause a contradiction when added, making proofs more compact and efficient.

RUP takes logical statements written in CNF, where each clause is a disjunction of literals $\{x_1, \ldots, x_n\}$. Negation of a literal x_i simply switches from x_i to $\neg x_i$, or vice versa. A formula is a conjunction of clauses. Z3 deals with formulae not in CNF by translating them using the Tseitin transformation [25, 22, 20].

The goal of a SAT or SMT solver is to decide whether clauses Γ are unsatisfiable, which means Γ entails falsity. Intermediate steps of a proof of unsatisfiability derive from Γ a set of clauses Δ, such that the conjunction of the clauses in Γ entails all clauses in Δ, with the ultimate proof including the empty clause in Δ.

A RUP inference of a clause $C = \{x_1, , \ldots, x_n\}$ from assumptions (clauses) Γ is correct, if from $\Gamma' := \Gamma, \{\neg x_1\}, \ldots, \{\neg x_n\}$ we can derive the empty clause $\{\}$ using unit-clause propagation only. A RUP proof from an initial clause set Γ_0 is a sequence of clauses C_i, for $i \geqslant 1$, such that for all i C_i is a RUP inference from Γ_{i-1}, where $\Gamma_j = \Gamma_{j-1} \cup \{C_j\}$, for $j \geqslant 1$. If some C_j is the empty clause $\{\}$, the sequence is called a RUP refutation [15]. Checking a RUP inference is done as follows: Divide Γ' into non-unit clauses Γ_{nunit} (clauses of length $\geqslant 2$) and unit clauses Γ_{unit} (clauses of length 1). If an empty clause is found, then we have derived falsity, and hence Γ was already unsatisfiable. Then, we repeatedly do the following, as long as $\Gamma_{\text{unit}} \neq \varnothing$: pick

*https://github.com/HarryBryant99, ORCID: 0009-0008-9926-8678
†https://www.swansea.ac.uk/staff/m.seisenberger/, ORCID: 0000-0002-2226-386X
‡https://csetzer.github.io/, https://www.swansea.ac.uk/staff/a.g.setzer/, ORCID: 0000-0001-5322-6060

one unit clause $\{x\} \in \Gamma_{\text{unit}}$ and remove it from Γ_{unit}. Next, we carry out unit clause resolution with all clauses c in $\Gamma_{\text{unit}} \cup \Gamma_{\text{nunit}}$. If c contains x, then it is implied by $\{x\}$ and is therefore removed. Otherwise, if c contains $\neg x$, then a unit resolution of c with $\{x\}$ derives $c' := c \backslash \{\neg x\}$. We replace c by c'; if it has length $\geqslant 2$, it will be in Γ_{nunit}, and if it has length 1, in Γ_{unit}. If it has length 0, then we have derived $\{\}$, so the RUP inference is verified, and we exit the loop. If c does not contain x or $\neg x$, it is kept. Once we have applied unit resolution with $\{x\}$ to all the clauses in $\Gamma_{\text{unit}} \cup \Gamma_{\text{nunit}}$, we repeat the process. After each step, the literal x and its negation do not occur anymore in $\Gamma_{\text{unit}} \cup \Gamma_{\text{nunit}}$, and no new literals have been created, so eventually the loop terminates because Γ_{unit} is empty. If we have not derived $\{\}$ by then, then $\{\}$ is not derivable by unit clause propagation, thus the verification of the RUP inference fails.

At each step, all formulae in $\Gamma_{\text{unit}} \cup \Gamma_{\text{nunit}}$ are derivable from Γ using unit clause resolution; therefore, they are entailed. If the procedure succeeds, then Γ' entails falsity and is therefore unsatisfiable. Thus, by classical logic (we have tertium non datur for the Boolean variables) it follows that Γ entails $\{x_1, \ldots, x_n\}$. We leave the full proof of completeness to a future article.

We have formalised the logic in Rocq [23] (formally called Coq [10]) and written a procedure that checks the proofs from Z3 and ensures that all RUP inferences are correct. For examples using more complex logical formulae or involving other data structures, such as integers supported by Z3, our checker verifies their correctness as well. However, verification of these rules is left for future work. For CNF formulae, Rocq creates only assumptions, deletions, subsumptions, and RUP inference rules. For RUP inferences, our system currently creates proofs which derive falsity from the assumptions and negated unit clauses using unit resolution. We have a proof in Rocq that if this proof is correct, the assumptions plus the negated unit clauses entail falsity and, therefore, the assumptions entail the formula in question. If all the generated proofs are correct, we have a proof in Rocq that the Z3 proof is correct. Therefore, if Z3 returns unsatisfiable, the assumptions are shown to be unsatisfiable.

Functions to check RUP inferences can be extracted from Rocq [23] into executable code using Rocq's extraction mechanism [21], typically to OCaml or Haskell. Extraction to other languages is possible, for example, C using the Codegen package [24]. Extraction to C supports basic types like numbers and lists, but complex types need extra handling or may not be supported. This is problematic for dependent types or higher-order functions lacking C equivalents.

Currently, proofs of correctness for RUP rely on generating all intermediate resolution proofs for each RUP inference. In fact, generating these proofs may be desirable when working with critical systems. Although having a proof that the checker is correct provides a high level of trust, there remains a remote possibility that an inconsistency in Rocq was used. Genuine bugs are occasionally detected in theorem provers. Therefore, having independently verifiable proof logs would allow for an even higher level of trust. The additional generated intermediate resolution proofs make it easier and therefore more trustworthy to verify the RUP proofs.

As a prototype, we are working on the verification of RUP inferences and Z3 proofs in Agda (see the GitHub repository [6]). Induction-recursion, as supported by Agda, is very beneficial in this project: we define proofs inductively while recursively deriving their conclusion. As a first step, we created a resolution proof of falsity from the assumptions and negated literals, provided that the RUP inference is correct. Therefore, these assumptions and negated literals are unsatisfiable. Although it is not of direct use for the industrial application – Agda does not allow compiling into C – the verification in Agda will, once completed, enable the integration of Z3 proofs into Agda. This advances our effort to incorporate automated theorem proving into Agda for verifying railway interlocking systems, a collaboration between Setzer and Kanso [19, 18].

References

[1] 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. doi:10.1007/978-3-031-43366-5_13.

[2] Nikolaj Bjørner. Proofs for SMT, 11 October 2022. Slides, Dagstuhl, October 11 2022. URL: https://z3prover.github.io/slides/proofs.html#/.

[3] Sascha Böhme. Proof reconstruction for Z3 in Isabelle/HOL, 2009. Slides of talk given at 7th International Workshop on Satisfiability Modulo Theories (SMT '09). URL: https://wwwbroy.in.tum.de/~boehmes/proofrec-talk.pdf.

[4] Sascha Böhme. Proof reconstruction for Z3 in Isabelle/HOL. In *Workshop on Proof Exchange for Theorem Proving (PxTP)*, 2009. URL: https://www21.in.tum.de/~boehmes/proofrec.pdf.

[5] Sascha Böhme and Tjark Weber. Fast LCF-Style Proof Reconstruction for Z3. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving*, pages 179–194, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:https://doi.org/10.1007/978-3-642-14052-5_14.

[6] Harry Bryant, Andrew Lawrence, Monika Seisenberger, and Anton Setzer. Verification of Z3 RUP Proofs in Coq-Rocq and Agda. https://github.com/HarryBryant99/Verification-of-Z3-RUP-Proofs-in-Coq-Rocq-and-Agda, 2025. Accessed: 2025-05-09.

[7] Simon Chadwick, Phillip James, Markus Roggenbach, and Thomas Werner. Formal Methods for Industrial Interlocking Verification. In *ICIRT*, 2018. doi:10.1109/ICIRT.2018.8641579.

[8] Ran Chen, Cyril Cohen, Jean-Jacques Lévy, Stephan Merz, and Laurent Théry. Formal Proofs of Tarjan's Strongly Connected Components Algorithm in Why3, Coq and Isabelle. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2019.13, doi:10.4230/LIPIcs.ITP.2019.13.

[9] Alessio Coltellacci, Gilles Dowek, and Stephan Merz. Reconstruction of SMT proofs with Lambdapi. In Giles Reger and Yoni Zohar, editors, *CEUR Workshop Proceedings*, volume 3725, pages 13–23, Montréal, Canada, July 2024. URL: https://inria.hal.science/hal-04861898.

[10] Rocq community. About the Rocq Prover, Accessed 10 March 2025. URL: https://rocq-prover.org/about.

[11] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient Certified RAT Verification. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 220–236, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-63046-5_14.

[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. URL: https://dl.acm.org/doi/abs/10.5555/1792734.1792766.

[13] Mathias Fleury and Hans-Jörg Schurr. Reconstructing veriT proofs in Isabelle/HOL. *Electronic Proceedings in Theoretical Computer Science*, 301:36–50, August 2019. doi:10.4204/eptcs.301.6.

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

[15] 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). URL: https://users.soe.ucsc.edu/~avg/ProofChecker/Documents/proofs-isaim08-long.pdf.

[16] Eugene Goldberg and Yakov Novikov. Verification of Proofs of Unsatisfiability for CNF Formulas. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 886–891, March 2003. doi:10.1109/DATE.2003.1253718.

[17] Alan G. Hamilton. *Logic for Mathematicians*. Cambridge University Press, Cambridge, rev. ed. edition, 1988.

[18] Karim Kanso. *Agda as a Platform for the Development of Verified Railway Interlocking Systems*. PhD thesis, Dept. of Computer Science, Swansea University, Swansea, UK, August 2012. Git repository available at https://github.com/kazkansouh/agda. URL: https://csetzer.github.io/articlesFromOthers/kanso/karimKansoPhDThesisAgdaAsAPlatformForVerifiedRailways.pdf.

[19] Karim Kanso and Anton Setzer. A light-weight integration of automated and interactive theorem proving. *Mathematical Structures in Computer Science*, 26(1):129–153, 2016. doi:10.1017/S0960129514000140.

[20] Elias Kuiter, Sebastian Krieter, Chico Sundermann, Thomas Thüm, and Gunter Saake. Tseitin or not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3551349.3556938.

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

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

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

[24] Akira Tanaka. Coq to C translation with partial evaluation. In *Proceedings of the 2021 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, PEPM 2021, pages 14–31, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3441296.3441394.

[25] Grigori S. Tseitin. On the complexity of derivation in propositional calculus. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg. doi:10.1007/978-3-642-81955-1_28.

[26] Nathan Wetzler, Marijn Heule, and Warren Hunt. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In *International Conference on Theory and Applications of Satisfiability Testing*, 07 2014. doi:10.1007/978-3-319-09284-3_31.

[27] Nathan David Wetzler. *Efficient, mechanically-verified validation of satisfiability solvers*. PhD thesis, Univeristy of Texas at Austin, Austin, 2015. URL: https://repositories.lib.utexas.edu/bitstream/handle/2152/30538/WETZLER-DISSERTATION-2015.pdf?sequence=1.