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

Harry Bryant, Andrew Lawrence,
Monika Seisenberger & Anton Setzer
Swansea University

Types 2025

Contributors

Harry Bryant (PhD student)
(Heavy weight lifting + Slides design)

Andrew Lawrence (Siemens Mobility)
(Money + Industrial expertise)

Monika Seisenberger (PhD supervisor)
(Money + Lead)

Anton Setzer (PhD co-supervisor)
(Dependent type missionary + Presenter)

Motivation

- **Joint work with Siemens Mobility:** Verification of railway interlocking systems
- **High safety requirements:** Safety-critical infrastructure
- **Why Z3?** Widely used SMT solver,
  Required by Siemens because industrial tool (Microsoft Research; liability issues).
- **Challenge:** SMT solvers can produce incorrect results
- **Community response:** SAT conferences now require proof checkers.
- **Safety Critical Systems** requires much higher level of correctness of proof checker than mathematics.
- Main problem **correctness of actual implementation of proof checker** rather than theoretical algorithm.

Key Insight

For safety-critical systems, we need **verified checkers**.

Verification doesn't replace Testing . . .

- Would you fly plane which has been **fully verified in Agda**

Verification doesn't replace Testing . . .

- Would you fly plane which has been **fully verified in Agda**
- but **never been flown?**

Verification doesn’t replace Testing . . .

- Would you fly plane which has been **fully verified in Agda**
- but **never been flown?**

Verification doesn't replace Testing . . .

- Would you fly plane which has been **fully verified in Agda**
- but **never been flown?**

- Example of small toy railway interlocking system developed by Anton

Verification doesn't replace Testing . . .

- Would you fly plane which has been **fully verified in Agda**
- but **never been flown?**

- Example of small toy railway interlocking system developed by Anton
- fully verified but trains started to disappear.

Verification doesn't replace Testing . . .

- Would you fly plane which has been **fully verified in Agda**
- but **never been flown?**

- Example of small toy railway interlocking system developed by Anton
- fully verified but trains started to disappear.
- Disappearing trains happened in real world interlocking systems (US)

. . . but can reduce cost

- **Reduces cost of testing** (finding errors earlier)
- Find some errors thorough testing won't find.

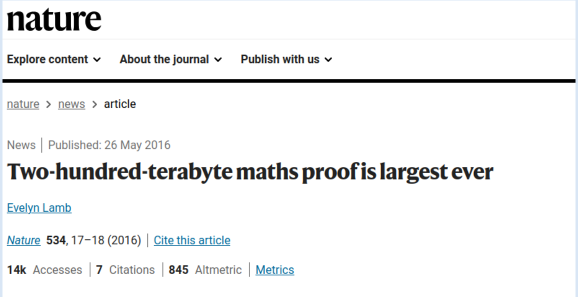
Industrial Use of Formal Methods

- **Big progress in use of formal methods**
  Now high level discussions about limits of SMT solving and Rocq prover possible.
- **Tool chain** in railway verification [BCL+23].
- Need industrial tools **licensed or under control of Siemens**.

SMT Proofs can be very long

Heule-Kullmann-Markek's "largest proof in the world"

200 TB maths proof [Lam16, HKM16].
Used already DRAT format (based on RUP) [FHB+24]
Generated on a supercomputer in Texas
In this form never made it to Swansea.
Compressed proof: 68 GB


nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature > news > article

News | Published: 26 May 2016

Two-hundred-terabyte maths proof is largest ever

Evelyn Lamb

Nature **534**, 17–18 (2016) | Cite this article

14k Accesses | **7** Citations | **845** Altmetric | Metrics

SMT Proofs can be very long

Why It Matters?

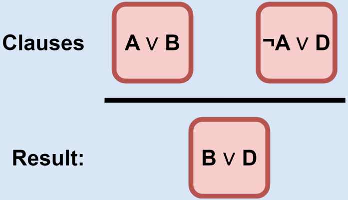
Proof size is a real challenge:

- Old resolution proofs not feasible.
- Verifying them requires significant computational resources.
- Proof checking more complex – proof checker requires verification.
- **Industrial proofs:** smaller size but still big and much higher requirement on correctness assurance.
- Need for efficient, trustworthy tools for handling large-scale proofs

Proofs in Z3

Old Z3 Proof Format

Based on full resolution.
Easy to verify.
Size problem.



New Z3 Proof Format based on RUP

- One of several more compressed proof proof formats for SAT/SMT solving.
- Introduced September 2022 for Z3 proofs [Bjø22].
- Based on Conflict Driven Clause Learning.
- More difficult to check and verify correctness of proof checker.

Our Solution

Project Plan

- **Prototype proof checker in Agda** including correctness proof.
- Write proof checker in Rocq.
- **Prove correctness in Rocq**.
- **Correctness doesn't require** creating **tree proofs** (resource consuming).
  - Optional creation of tree proofs for additional confidence
- **Extract verifier** from Rocq as **efficient C-program**.
- **Verifier is extensible** to addition of additional SMT features.
- **Two-/Three-level proof pattern** for proof of correctness of proof calculi in dependent type theory
- Use it for **integrating Z3 proofs into Agda** (Work in progress).

1. Introduction
2. Background
3. Correctness
4. Two vs Three Level Approach
5. Conclusion

1 Introduction

2 Background

3 Correctness

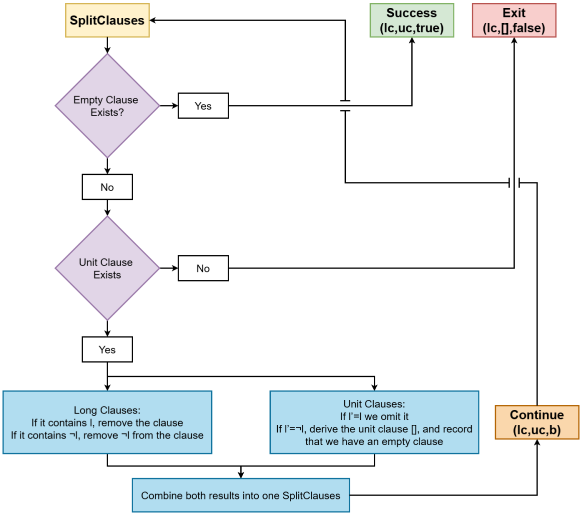
4 Two vs Three Level Approach

5 Conclusion

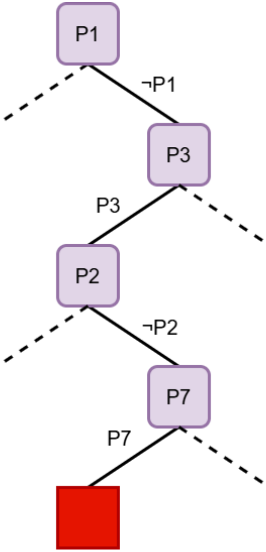
SAT Solving - General Setup

- We work on SAT-solving part of SMT solving.
- Basis propositional variables (which may denote longer SMT formulas).
- **Literals:** $l_1, l_2, \ldots$ Positive or negative propositional variables
- **Clauses:** Disjunctions of literals written $c = [l_1, l_2, l_3]$
- **Sequents:** Conjunctions of clauses.
- Split sequents using **Clause Splitting:**
  - Long clauses ($length \geq 2$)
  - Unit clauses ($length = 1$)
  - Empty clause (contradiction)
- Represented as: `SplitClauses = LongClauses × UnitClauses × Bool`
- **Bool flag:** `true` means empty clause exists (successful proof).

Unit-Clause Propagation – Inexpensive Reductions



Conflict Driven Clause Learning



- $\neg P1 \wedge P3 \wedge \neg P2 \wedge P7 \rightarrow$ conflict.
- Deeper analysis optimises it, e.g. $(P3 \wedge P7) \rightarrow$ conflict
- Therefore, add conflict clause $[\neg P3, \neg P7]$
- Backtrack to decision level $P3$ and choose $\neg P3$

Reverse Unit Propagation (RUP)

RUP Inference

A clause $C = [l_1, l_2, \ldots, l_k]$ is a RUP Inference from a formula F if:

The unit clauses $[\neg l_1], [\neg l_2], \ldots, [\neg l_k]$, when added to F, make the formula refutable via Unit-Clause Propagation (UCP).

Reverse Unit Propagation (RUP)

RUP Inference

A clause $C = [l_1, l_2, \ldots, l_k]$ is a RUP Inference from a formula F if:

The unit clauses $[\neg l_1], [\neg l_2], \ldots, [\neg l_k]$, when added to F,
make the formula refutable via Unit-Clause Propagation (UCP).

RUP Proof

A sequence of clauses $C_1, C_2, \ldots,$ where each C_i is a RUP Inference from the formula:

$$F_j = F_{j-1} \cup \{C_j\}, \quad j \geq 1.$$

If a clause is a RUP Inference, its negation will lead to a contradiction via UCP.

Reverse Unit Propagation (RUP)

RUP Inference

A clause $C = [l_1, l_2, \ldots, l_k]$ is a RUP Inference from a formula F if:
The unit clauses $[\neg l_1], [\neg l_2], \ldots, [\neg l_k]$, when added to F,
make the formula refutable via Unit-Clause Propagation (UCP).

RUP Proof

A sequence of clauses $C_1, C_2, \ldots,$ where each C_i is a RUP Inference from the formula:

$$F_j = F_{j-1} \cup \{C_j\}, \quad j \geq 1.$$

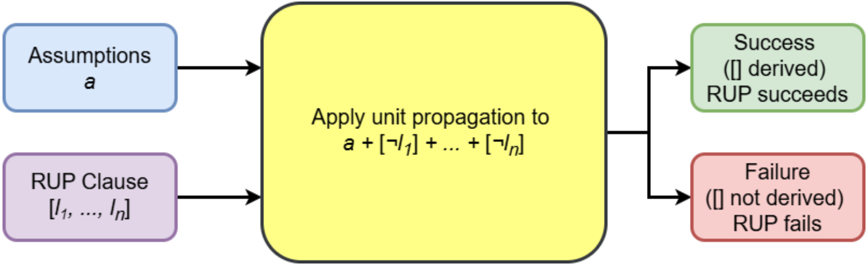
If a clause is a RUP Inference, its negation will lead to a contradiction via UCP.

RUP Refutation

A RUP Proof in which some clause $C_j = []$. This indicates that F_0 is unsatisfiable.

RUP Checker

For each RUP Inference, apply the RUP Checker to the list of assumptions *a*:



1 Introduction

2 Background

3 Correctness

4 Two vs Three Level Approach

5 Conclusion

Proof of Correctness

Lemma: $A \vdash_{\mathrm{UnitResolution}} \mathrm{UnitProp}(A)$.

Proof: Use unit resolution to derive from
$[l_1, \ldots, l_{n-1}, \neg l]$ and $[l]$
$[l_1, \ldots, l_{n-1}]$.

Lemma: If $A \vdash_{\mathrm{UnitResolution}} B$ then $A \models B$.

Proof of Correctness

Lemma: If $\texttt{RUPChecker}(A, [l_1, ..., l_n])\ \texttt{=}\ \texttt{true}$, then:

$$A + [\neg l_1] + \cdots + [\neg l_n] \models []$$

- $\texttt{RUPChecker}(A, [l_1, ..., l_n])\ \texttt{=}\ \texttt{true}$
- $\Rightarrow A + [\neg l_1] + \cdots + [\neg l_n] \vdash_{\text{UnitResolution}} []$
- $\Rightarrow A + [\neg l_1] + \cdots + [\neg l_n] \models []$.

Proof of Correctness

Lemma: One step entailment from conflict

$$A + [\neg l] \models c \quad \Rightarrow \quad A \models [l] \cup c$$

Lemma: Entailment from Conflict

$$A + [\neg l_1] + \cdots + [\neg l_n] \models [] \quad \Rightarrow \quad A \models [l_1, \ldots, l_n]$$

Theorem: Soundness of RUP Checker

$$\mathtt{RUPChecker}(A, c) = \mathtt{true} \Rightarrow A \models c$$

1 Introduction

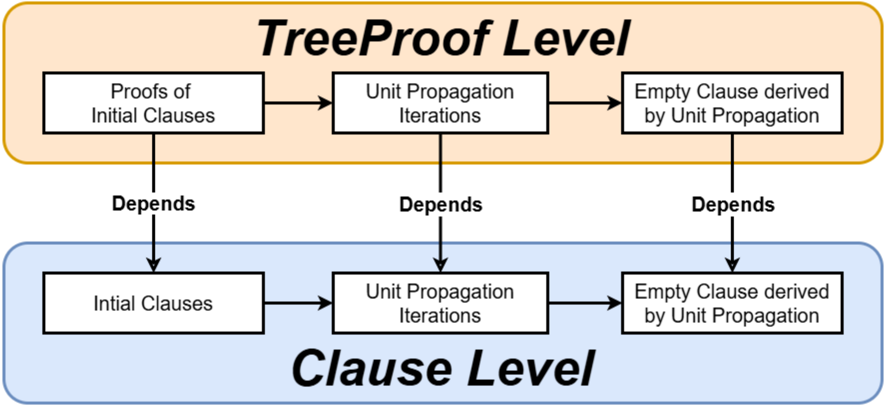
2 Background

3 Correctness

4 Two vs Three Level Approach

5 Conclusion

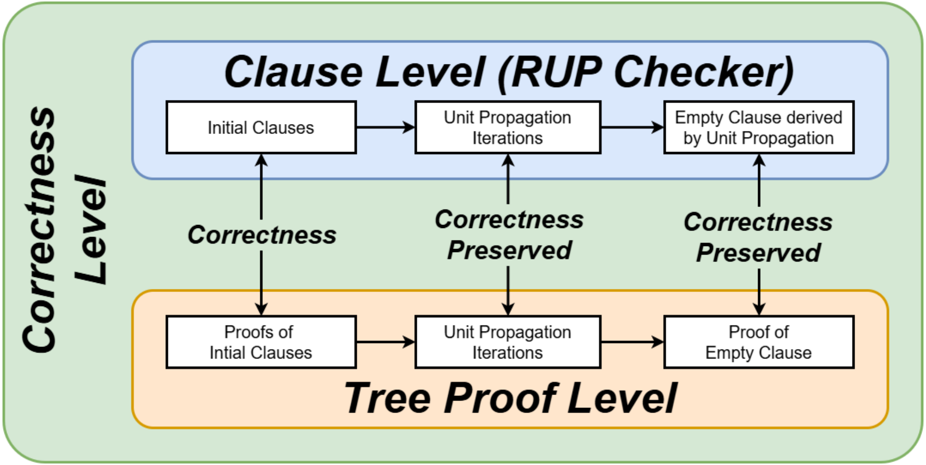
Agda: Two Level Approach for Correctness



Dealing with Resistance to Dependent Types

- Facing rebellion against dependent types
  (by Swansea's logic group)
- Therefore smuggling in dependent types where acceptable:
  - Treeproofs depending on clauses not acceptable.
  - Dependent correctness predicate is acceptable.

Rocq: Three Level Approach for Correctness



Example Three Level Approach (Rocq)

```
Definition SplitClauses : Type := (list Clause * list Literal * bool).

Definition SplitTreeProofs : Type := (list TreeProof * list TreeProof * option TreeProof).

Definition CorrectSplit (al : Assumption)(c : SplitClauses)
          (t : SplitTreeProofs) : Prop :=
  match c with
  | (clauses,literals,b) =>
      match t with
      | (ct,lt,bt) => (CorrectProofList al clauses ct) /\
                      (CorrectLiteralProof al literals lt) /\
                      (CorrectOptionTreeProof' al b bt)
      end
  end.
```

Example ProofStep (Three Level Approach)

```
Definition propagationStep
  (clauses : list Clause)
  (literals : list Literal)
  (l : Literal) : SplitClauses :=
  combineSplitClausesSplitLits (processAndSplitClausesWithLit clauses l)
    (processListLitsWithLit literals l).

Definition propagationStepProofs (clauses : list Clause)
  (literals : list Literal) (l : Literal) (proofs_c proofs_l : list TreeProof)
  (tp : TreeProof) : SplitTreeProofs :=
  combineSplitTreeProofs (process_and_extract_treeproofs clauses l proofs_c tp)
    (remove_treeproof literals proofs_l l tp).
```

Example ProofStep (Three Level Approach)

```
Lemma propagationStepCorrect :
  forall (al : Assumption)
         (clauses : list Clause)
         (literals : list Literal)
         (l : Literal)
         (proofs_c proofs_l : list TreeProof)
         (tp : TreeProof),
    CorrectProofList al clauses proofs_c ->
    CorrectLiteralProof al literals proofs_l ->
    CorrectProof al [l] tp ->
    CorrectSplit al
      (propagationStep clauses literals l)
      (propagationStepProofs clauses literals l proofs_c proofs_l tp).
```

Theorems in Agda and Rocq

Theorem in Agda:

```
rupCorrect : (f : Formula)(rp : Clause) → (atom (checkOneRup f rp))
                → EntailsCl f rp
```

Theorem in Rocq:

```
(* Main Theorem *)
Lemma RUP_Checker_correct :
  forall (a : Assumption)(c : Clause),
    RUP_Checker a c = true -> entails a c.
```

1 Introduction

2 Background

3 Correctness

4 Two vs Three Level Approach

5 Conclusion

Conclusion

- Addressing RUP format of proofs.
- **Theorem RUPChecker**$(A, c) =$ **true** $\Rightarrow$ **A** $\models$ **c.**
- Proofs in Agda and Rocq.
- **Two-** and **three level approach to proving correctness**
  More general proof pattern.
- No need to generate tree proofs from RUP proof (resource consuming) and then check them.
  - But option to compute tree proofs of [] if RUPChecker(a,c) returns true for extra confidence.
- Proof checker works well on railway examples.
  - smaller proofs: 150,000 lines, roughly 30,000 steps, 3 mins.
  - larger proofs: 4,750,000 lines, roughly 500,000 steps, 7.5 hrs.
- Should allow to integrate output from SAT solvers into Agda and Rocq proofs (Important for Agda!).
  - Combination of interactive and interactive theorem proving.
  - Modelling interactively, verification conditions using SMT solving.

Limitations

- Need verified extraction of programs from Rocq.
- Need to explore use of trusted core of Rocq.

Thank You for Listening

Bibliography I

[BCL+23] 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. `http://doi.org/10.1007/978-3-031-43366-5_13`.

[Bjø22] Nikolaj Bjørner. Proofs for SMT, 11 October 2022. Slides, Dagstuhl, October 11 2022, available from `https://z3prover.github.io/slides/proofs.html#/`.

[FHB+24] Nick Feng, Alan J. Hu, Sam Bayless, Syed M. Iqbal, Patrick Trentin, Mike Whalen, Lee Pike, and John Backes. Drat proofs of unsatisfiability for sat modulo monotonic theories, 2024.

Bibliography II

[HKM16] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer, page 228–245. Springer International Publishing, 2016. `http://doi.org/10.1007/978-3-319-40970-2_15`.

[Lam16] Evelyn Lamb. Two-hundred-terabyte maths proof is largest ever. Nature, 534(7605):17–18, Jun 2016. `https://doi.org/10.1038/nature.2016.19990`.

[WHH13] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. Mechanical verification of SAT refutations with extended resolution. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, Interactive Theorem Proving, pages 229–244, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[WHH14] 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, pages 422–429, 07 2014.