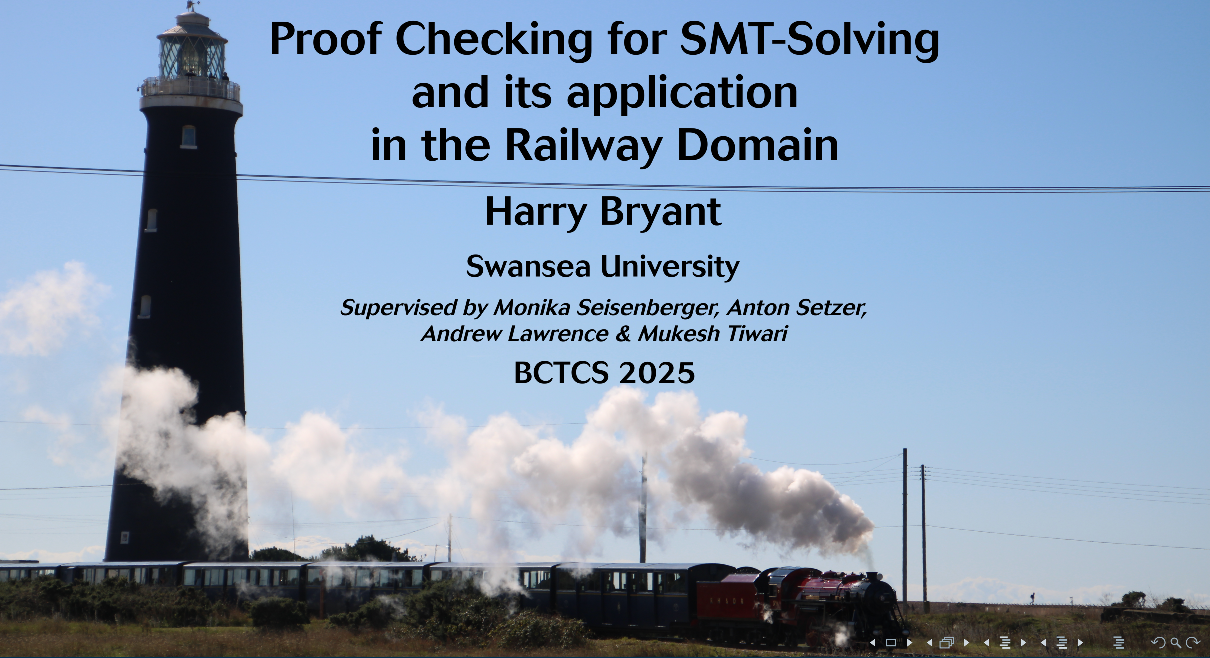
Proof Checking for SMT-Solving and its application in the Railway Domain

Harry Bryant

Swansea University

Supervised by Monika Seisenberger, Anton Setzer, Andrew Lawrence & Mukesh Tiwari

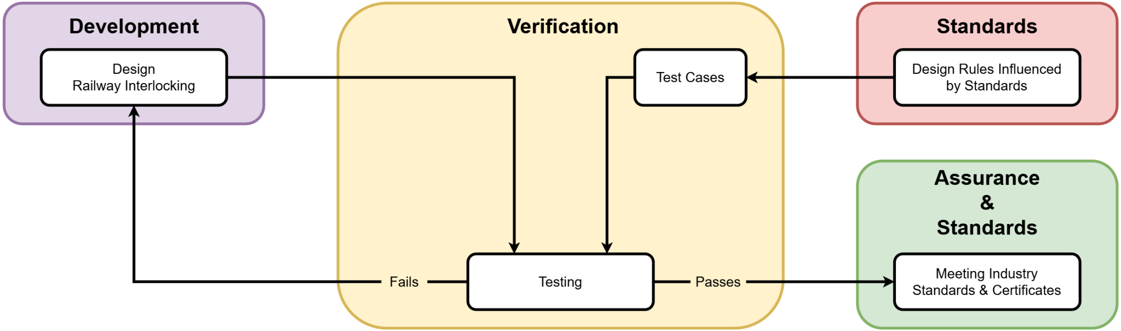
BCTCS 2025



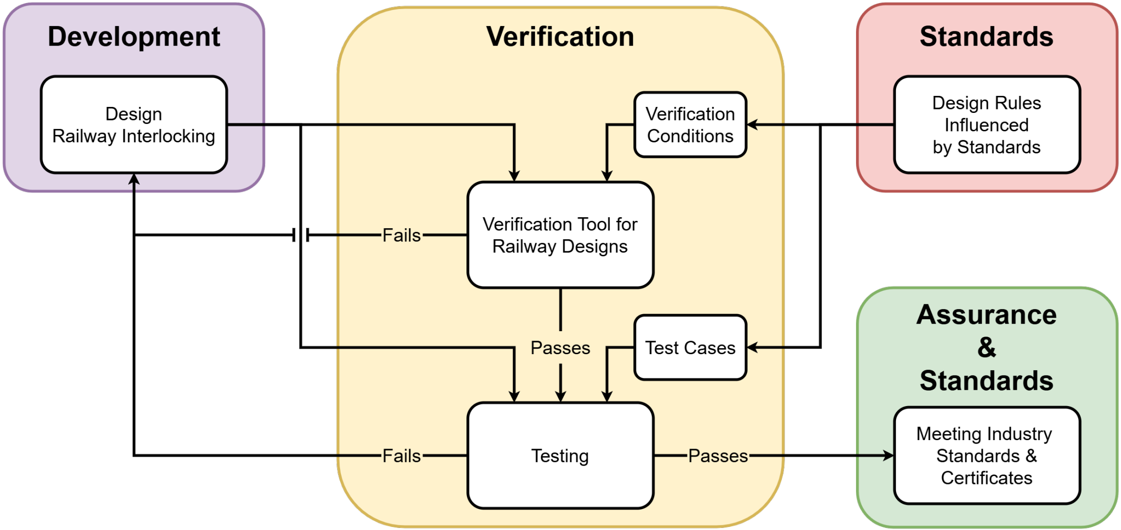
Overview

1. Railway Verification and Proof Checkers
2. The Z3 Proof Rules
3. Proving the Rules to be Correct
4. Z3 Proof Checker Prototype
5. Future Work

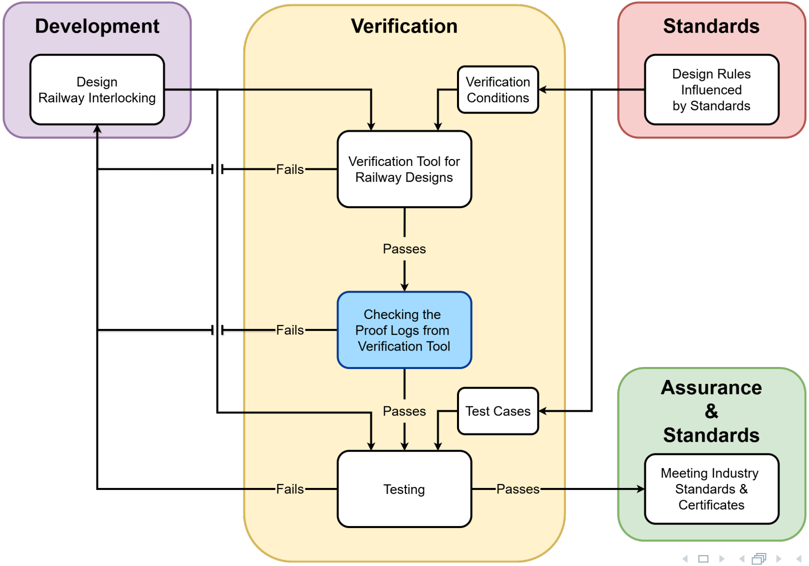
Railway Verification Overview: Testing



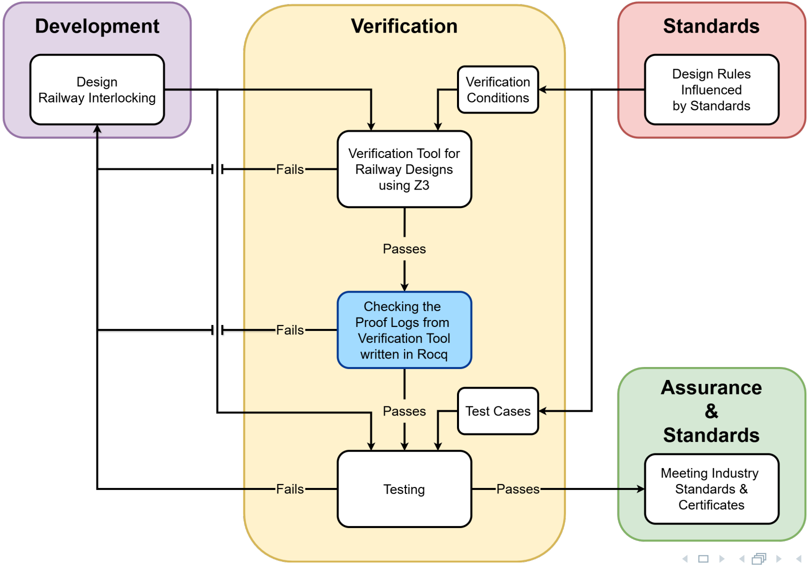
Railway Verification Overview: Verification & Testing



Railway Verification Overview: Verification, Checking & Testing



Railway Verification Overview: Z3 + Rocq



Z3 Theorem Prover

Z3 is a high-performance theorem prover developed by Microsoft Research

- Includes solvers for both SAT and SMT problems
- Offers efficient solving algorithms and supports various input formats and programming languages
- Z3 provides precise and efficient results
  - A Model is produced when Satisfiable
  - A Proof is produced when Unsatisfiable

PhD Project Goal

The Problem:

- SMT solvers such as Z3 are tools often applied to safety critical systems
- However, these may have flaws or optimisations that produce incorrect results
- We require more than a True or False result, we want a justified and verified result with a certificate

PhD Project Goal

The Problem:

- SMT solvers such as Z3 are tools often applied to safety critical systems
- However, these may have flaws or optimisations that produce incorrect results
- We require more than a True or False result, we want a justified and verified result with a certificate

Our Goal:

- Build a verified proof checker to independently verify Z3's results

PhD Project Goal

The Problem:

- SMT solvers such as Z3 are tools often applied to safety critical systems
- However, these may have flaws or optimisations that produce incorrect results
- We require more than a True or False result, we want a justified and verified result with a certificate

Our Goal:

- Build a verified proof checker to independently verify Z3's results

Why It Matters:

- This increases confidence in using SMT solvers in our Industrial Partner's tools

The need for a Bespoke Proof Checker

Why it is needed:

- Currently there is no Proof Checker for the current Z3 format
- Checker should be “simple” in comparison to the SMT Solvers
- Checker should obey Standard Industrial Validation Methods

Our Approach:

- Option 1: Write checker by hand
  - Risk of introducing errors
- Option 2: Formalise in a Theorem Prover, Prove it, then Extract the code
  - Provides added Safety because the Checker is verified

Section 2

The Z3 Proof Rules

Comparing the Z3 Proof Formats

Reverse Unit Propagation (RUP)

RUP Inference

A clause $C = \{x_1, x_2, \ldots, x_k\}$ is a RUP Inference from a formula F if:

The unit clauses $\{\neg x_1\}, \{\neg x_2\}, \ldots, \{\neg x_k\}$, when added to F, make the formula refutable via Unit-Clause Propagation (UCP).

Reverse Unit Propagation (RUP)

RUP Inference

A clause $C = \{x_1, x_2, \ldots, x_k\}$ is a RUP Inference from a formula F if:
The unit clauses $\{\neg x_1\}, \{\neg x_2\}, \ldots, \{\neg x_k\}$, 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 $C = \{x_1, x_2, \ldots, x_k\}$ is a RUP Inference from a formula F if:

The unit clauses $\{\neg x_1\}, \{\neg x_2\}, \ldots, \{\neg x_k\}$, 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:

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

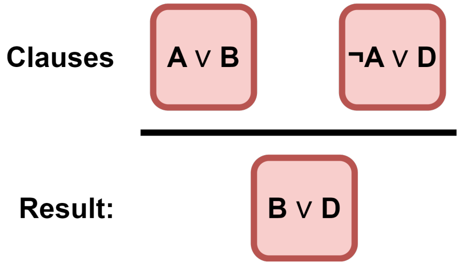
If a clause is a RUP Inference, its negation will lead to a contradiction via UCP.

RUP Refutation

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

Connection to Resolution

Standard Resolution:



Reverse Unit Propagation:

- Resolution is replaced by RUP
- $\{B \vee D\}$ is a Valid RUP Inference derived from the clauses $\{A \vee B\}$ and $\{\neg A \vee D\}$
- Clauses used to find the Inference are not stored
- Easier to verify via Unit Propagation than finding the Clauses
- From $\{B \vee D\}$, derive clauses $\{\neg B\}$ and $\{\neg D\}$ to be added to the formula

Validating RUP Inferences

Checking a RUP Inference:

$$\{A \lor B\}$$
$$\land\{\neg A \lor D\}$$
$$\land \neg B$$
$$\land \neg D$$

Unit-Clause Propagation Applied

Propagating Leads to a Contradiction:

$$A$$

$$\land \neg A$$

$$\land \neg B$$

$$\land \neg D$$

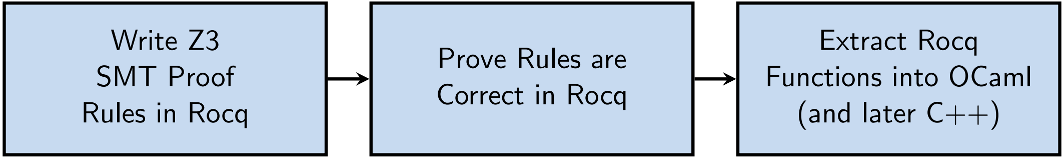
$\emptyset$ is derived, therefore, a valid Inference

Section 3

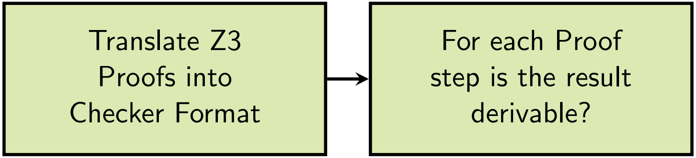
Proving the Rules to be Correct

Proof Checker Development Plan

Stage 1

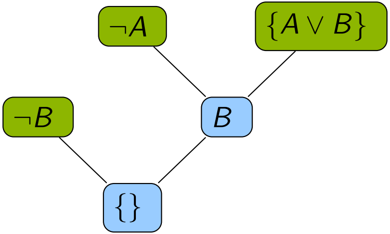


Stage 2



TreeProofs for Unit Propagation

- Unit Propagation applies a series of Unit Resolutions to derive a contradiction
- A Unit Resolution Proof can be represented as a tree



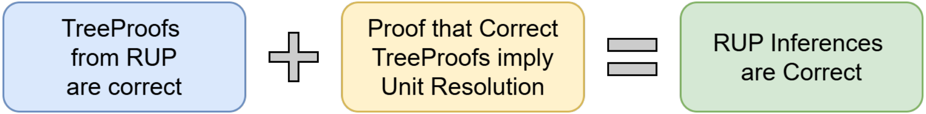
Proving RUP in Rocq

RUP relies on Unit-Clause Propagation, which applies Unit Resolution:
Therefore, for every step in Unit-Clause Propagation, we can create a TreeProof

Valid RUP Inference:

1. Unit-Clause Propagation returns $\{\}$?
2. For all TreeProofs produced in doing so, are they correct?

Application to Unit Resolution:



Acquiring a complete proof rather than generating TreeProofs is ongoing.

Proving Unit Resolution in Rocq

Goal:

Prove that if a Unit Resolution Proof is Correct, then it can be modelled as a Unit Resolution in Rocq, and then it will be Entailed

```
Inductive unitres : formula -> clause -> Prop
    :=
| subsumption : forall (c c2 : clause) (f :
    formula),
   In c f ->
   subset c c2 ->
   unitres f c2
| resolution : forall (c : clause) (l : literal )
    (f : formula),
   unitres f c ->
    is_literal_in_clause l c ->
   unitres f (cons (opposite l) []) ->
   unitres f ( remove_literal_from_clause l c).
```

```
Definition entails (f : formula) (c : clause) :
    Prop :=
 ( forall (m : model),
   Models_formula m f -> Models_clause m c).
```

Proving Unit Resolution in Rocq

Proving each Unit Resolution will remove a literal from a clause while preserving satisfiability:

TreeProof Correctness ⇒ Unit Resolution

```
Lemma treeproof_implies_unitres :
  forall (t : TreeProof) (ass : Assumption),
  correct ass t = true ->
  unitres ass (conclusion ass t).
Proof.
```

Unit Resolution ⇒ Entailment

```
Lemma URes_implies_Entailment :
  forall (f : formula) (c : clause),
  unitres f c ->
  entails f c.
Proof.
```

Proving Unit Resolution in Rocq

Entailment of Falsity (Single Literal)

```
Lemma entailsFalsity1 :
  forall (A : formula) (c : clause) (x :
        literal ),
  entails ([[opposite x]] ++ A) [] ->
  entails A [x].
Proof.
```

Entailment of Falsity (Multiple Literals)

```
Lemma entailsFalsity2 :
  forall (A : formula) (c : clause)
         (x : literal ) (xs : list literal ),
  entails ([[opposite x]] ++ A) xs ->
  entails A (x :: xs).
Proof.
```

General Entailment of Falsity

```
Lemma entailsFalsity :
  forall (A : formula) (xs : list literal ),
  entails (negate_clause xs ++ A) [] ->
  entails A xs.
Proof.
```

Section 4

Z3 Proof Checker Prototype

Proof Checker Status

Overview:

- A Proof-of-Concept SAT proof checker built and extracted in Rocq
- Designed for integration with an industrial verification tool

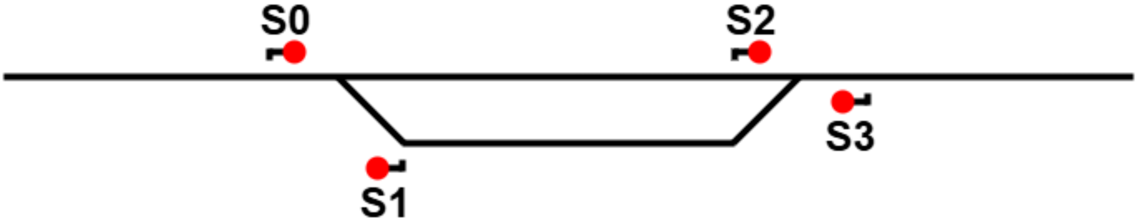
Focus:

- RUP is the new basis for the checker
- Other rules, such as the Tseitin Transformation, have been implemented

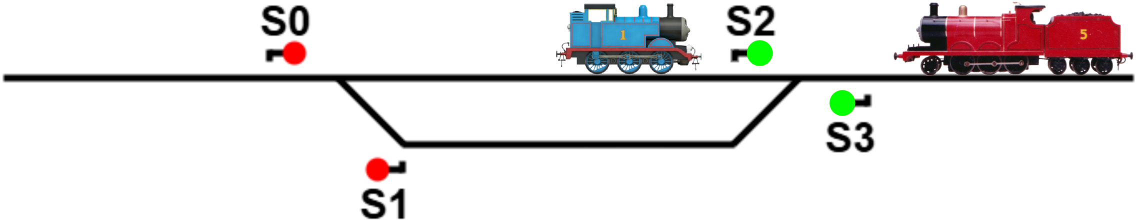
Current Status:

- A full SAT Proof Checker is now operational
- Tested on small examples to verify correctness
- Verification of SAT Proof Checker is nearing completion

Checking a Railway Example



Checking a Railway Example



We do not want opposing signals being green simultaneously

Checking a Railway Example

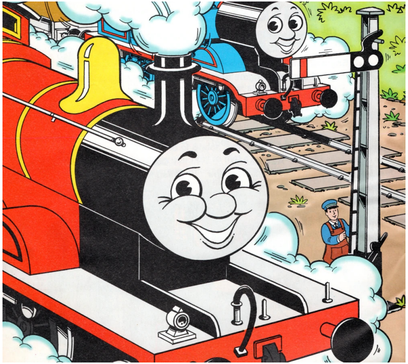
Checking a Railway Example

- **Signal Conditions**:
  - Signals depend on track segments and points
  - S0 & S1, and S2 & S3 cannot both be green
- **Train Movement Conditions**:
  - Trains enter tracks if signals are green
  - No two trains on the same track simultaneously
- **Track Occupation**:
  - Track Segments are occupied if:
    - There is a train in the segment before
    - The corresponding signal is green
- **Contradiction Creation**: Assumes S0 & S1, S2 & S3 are green
- **Satisfiability Check**: Unsatisfiable, signals are safe

Checking a Railway Example

Checker Output:

```
Running Proof Checker for example: rhdr
Proof Check:
Checker result: True
All steps are valid.
```



Summary:

- Checker returns `true` for the list of steps
- Therefore, we have trust in Z3's response
- Therefore, opposing signals cannot both be green

Section 5

Future Work

The Next Steps

- Continue testing the SAT Checker on Industrial Scale Examples
- Formalise and Prove further Z3 Proof Rules for SMT Examples in Rocq
- How to deal with SMT decision procedures
- Enable us to perform proof checking on all our Industrial Partner's tools
- Perform Industrial Testing of the final checker

Summary

- The Proof Checker will independently verify formal Z3 proofs
- The Checker will be extracted from proven code to assure that it is also correct
- Proof Checking provides further assurance to the verification process
- This further increases trust in the Railway Interlockings

Thank You for Listening

References

[1] Clark W. Barrett (2008)
SMT Solvers: Theory and Practice

[2] Leonardo de Moura, Nikolaj Bjørner (2008)
Z3: An Efficient SMT Solver
Tools and Algorithms for the Construction and Analysis of Systems
Springer Berlin Heidelberg
https://doi.org/10.1007/978-3-540-78800-3

[3] Madhusree Banerjee, Victor Cai, Sunitha Lakshmanappa, Andrew Lawrence, Markus Roggenbach, Monika Seisenberger, Thomas Werner (2023)
A Tool-Chain for the Verification of Geographic Scheme Data
Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification
Springer Nature Switzerland
https://doi.org/10.1007/978-3-031-43366-5

References

[4] Simon Chadwick, Phillip James, Markus Roggenbach, Tom Werner (2018)
Formal Methods for Industrial Interlocking Verification
2018 International Conference on Intelligent Rail Transportation (ICIRT)
https://doi.org/10.1109/ICIRT.2018.8641579

[5] Simon Chadwick, Tom Werner and Siemens Mobility (2020)
Formal Methods From Theory towards Practice

[6] Jean-Christophe Filliâtre, Pierre Letouzey (2020)
Extraction of programs in OCaml and Haskell
https://coq.inria.fr/doc/V8.11.1/refman/addendum/extraction.html

[7] Yannick Forster, Matthieu Sozeau, Nicolas Tabareau (2024)
Verified Extraction from Coq to OCaml
Proceedings of the ACM on Programming Languages, 8, PLDI, 52–75
https://inria.hal.science/hal-04329663/file/main.pdf

References

[8] Allen Van Gelder (2008)
Verifying RUP Proofs of Propositional Unsatisfiability
International Symposium on Artificial Intelligence and Mathematics
https://api.semanticscholar.org/CorpusID:17252954

[9] 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
253 – 268.

[10] Phillip James (2010)
SAT- Based Model Checking and Its Applications to Train Control Systems
Swansea University
https://books.google.co.uk/books?id=rPWozQEACAAJ

[11] Chu-Min Li, Fan Xiao, Mao Luo, Felip Manyà, Zhipeng Lü, Yu Li (2020)
Clause vivification by unit propagation in CDCL SAT solvers
Artificial Intelligence, 279, 103197
https://www.sciencedirect.com/science/article/pii/S0004370219301961

References

[12] Pierre Letouzey (2008)
Extraction in Coq: An Overview
Logic and Theory of Algorithms, Proceedings of the 4th Conference on Computability in Europe (CiE 2008)
https://doi.org/10.1007/978-3-540-69407-6_39

[13] Marius Minea (2024)
Conjunctive Normal Form: Tseitin Transform
H250: Honors Colloquium – Introduction to Computation

[14] Vincent Derkinderen (2024)
Pruning Boolean d-DNNF Circuits Through Tseitin-Awareness
arXiv preprint arXiv:2407.17951
https://arxiv.org/pdf/2407.17951