LLM-Assisted Formal Verification of Bit-vector Invertibility Conditions in Rocq

Ha Truong[1], Jordin Palmeri[1], Liam Secrist[1], Jacob Schuckman[1], Arjun Viswanathan[1], and Yoni Zohar[2]

[1] Union College, Schenectady, NY, USA
[2] Bar-Ilan University, Ramat Gan, Israel

Abstract

Invertibility conditions are properties used in Satisfiability Modulo Theories (SMT) to reason about bit-vector formulas. Previous work has verified 125 of 160 invertibility conditions for arbitrary bit-width. Some of these were proven in the Rocq interactive theorem prover (ITP) and others were verified using SMT solvers. We prove the remaining 35 properties in Rocq with the help of Google Gemini and Claude Code. This paper presents our efforts in completing this set of proofs.

1 Introduction

Reasoning about machine integers has important applications in hardware and software verification [9, 28]. Satisfiability Modulo Theories (SMT) solvers [5] can be used to perform such reasoning. However, SMT-LIB 2 [4] — the input/output standard for most SMT solvers — only allows properties over bit-vectors (used to model machine integers) to be expressed for specific numeric bit-widths. As a result, such properties cannot be verified with enough generality. For example, the commutativity of addition of bit-vectors cannot be verified for bit-vectors of all bit-widths.

Our focus is on a specific set of properties called *invertibility conditions* [20] — used incidentally, by SMT solvers for reasoning over formulas containing quantified and non-quantified bit-vectors in cvc5 [3] and bitwuzla [18] (see [20, 19] for applications of invertibility conditions). As an illustrative example, consider the formula $x + s = t$, where x, s and t are bit-vectors that have the same bit-width. x is invertible for this formula, and its inverse or solution is $t - s$. Since such a solution exists for all values of x, s, and t, the condition under which x is invertible is simply $\top$. This is expressed as the *invertibility equivalence* $\top \Leftrightarrow \exists x.\ x + s = t$. This formula is valid in the theory of bit-vectors for any bit-width. Niemetz et al. [20] generate this and 179 other invertibility conditions, most of which are more constrained than $\top$. For example, $s \div (s \div t) = t \Leftrightarrow \exists x.\ s \div x = t$ expresses the fact that x is invertible in $s \div x = t$ precisely when $s \div (s \div t) = t$ holds. These 180 invertibility conditions were verified using SMT solvers by Niemetz et al. [20] but only up to bit-width 65. Subsequent work [22, 23, 7] verified a large subset of these equivalences for arbitrary bit-width using SMT solvers via a conversion of the formulas to quantified non-linear integer arithmetic and uninterpreted functions. However, this approach was not fully successful in verifying all equivalences for arbitrary bit-width.

This led to using the interactive theorem prover (ITP) Rocq [27] to verify the remaining properties. Rocq's expressive type system provides dependent types, which can be used to specify properties over bit-vectors for any bit-width. Moreover, proving these properties using an ITP gives us more confidence in their validity with respect to their SMT-based verification. Previous work [11, 12] used a bit-vector library for SMTCoq [10] to prove 19 equivalences in Rocq, 12 of which were previously unverified for arbitrary bit-width.

Symbol	SMT-LIB Syntax	Sort
$=, \neq$	=, distinct	$\sigma_{[n]} \times \sigma_{[n]} \to \mathsf{Bool}$
$<_u, >_u, \leq_u, \geq_u$	bvult, bvugt, bvule, bvuge	$\sigma_{[n]} \times \sigma_{[n]} \to \mathsf{Bool}$
$\sim, -$	bvnot, bvneg	$\sigma_{[n]} \to \sigma_{[n]}$
$\&, \mid, \ll, \gg, \gg_a$	bvand, bvor, bvshl, bvlshr, bvashr	$\sigma_{[n]} \times \sigma_{[n]} \to \sigma_{[n]}$
$+$	bvadd	$\sigma_{[n]} \times \sigma_{[n]} \to \sigma_{[n]}$
$<_s, >_s, \leq_s, \geq_s$	bvslt, bvsgt, bvsle, bvsge	$\sigma_{[n]} \times \sigma_{[n]} \to \mathsf{Bool}$
$\cdot$, mod, $\div$	bvmul, bvurem, bvudiv	$\sigma_{[n]} \times \sigma_{[n]} \to \sigma_{[n]}$
$\circ$	concat	$\sigma_{[n]} \times \sigma_{[m]} \to \sigma_{[n+m]}$
$[u:l]$	extract	$\sigma_{[n]} \to \sigma_{[u-l+1]}$

Table 1: The signaturess Σ_0, $\Sigma_{0.5}$ and Σ_1 with SMT-LIB 2 syntax.

In this work, we consider 160 of the 180 invertibility conditions that do not include multiple bit-widths. We verified all invertibility conditions from this subset that were previously unverified — either using Rocq or SMT solvers — for arbitrary bit-width. We proved some of these manually, others using Google Gemini [26] as an assistant, and yet others were proved using Claude Code [2] with minimal human supervision.

To summarize, our main contribution is the completion of the formal verification of all 160 invertibility equivalences. As part of this contribution, we also extended a Rocq library for bit-vectors, and demonstrated how modern large language model (LLM) based AI assistants, like Gemini and Claude Code can help in formal verification tasks.

The rest of the paper is structured as follows. In Section 2, we introduce the formal preliminaries necessary for a technical presentation of our work; in particualr, in Section 2.3, we present invertibility conditions and summarize previous attempts at formalizing them; Section 3 presents our contributions in completing these prior formalizations; Section 4 discusses how we approached these proofs, specifically, how we involved Gemini and Claude Code to help us with proofs, and Section 5 presents our conclusions and future avenues of research.

2 Background

2.1 Theory of Bit-Vectors

As in Ekici et al. [11, 12] (that this work is an extension of), we use many-sorted first-order logic with equality [13] and the signature Σ_{BV} of the SMT-LIB 2 theory of fixed-width bit-vectors which defines a unique sort for each positive integer n, denoted as $\sigma_{[n]}$. Constant, function and predicate symbols of Σ_{BV} as well as Σ_{BV}-formulas and their semantics are as specified by the SMT-LIB standard. We denote equality by $=$, and use $x \neq y$ as an abbreviation for $\neg(x = y)$. We also use the (overloaded) constant 0 to represent the bit-vectors composed of all 0-bits.

Table 1 contains the operators from Σ_{BV} for which invertibility conditions were defined. We define Σ_1 to be the signature that contains only these symbols. Σ_0 is the sub-signature obtained by only taking all operators until the first horizontal line and $\Sigma_{0.5}$ is the sub-signature obtained by only taking all operators until the second horizontal line.

2.2 Rocq

The Rocq proof assistant is based on the calculus of inductive constructions (CIC) [24]. It represents formulas as types and reduces the task of proving formulas to finding a term of the corresponding type. Its type system is more expressive than that of SMT-LIB 2. Of specific interest for this work is the ability to express formulas over *dependent types* — types that can depend on values — in Rocq. Dependent types allow types to depend on values. For-instance, a bit-vector type in Rocq can be defined as a list of Booleans — `bitvector`, say — and this type can be further parameterized by a natural number using a dependent type. This allows one to talk about bit-vectors of length n (`bitvector` n) for natural number n. Moreover, formulas can be constructed by quantifying over n.

2.3 Previous Work

We recall the following terms presented in [20, 12]. An *invertibility condition* for a variable x in a Σ_{BV}-literal $\ell[x, s, t]$ is a formula $IC[s, t]$ such that $\forall s.\forall t.\ IC[s, t] \Leftrightarrow \exists x.\ \ell[x, s, t]$ is valid in the theory of fixed-width bit-vectors, for all bit-widths. The *invertibility equivalence* associated with the literal $\ell[x, s, t]$ and its invertibility condition $IC[s, t]$ is the formula $IC[s, t] \Leftrightarrow \exists x.\ \ell[x, s, t]$. Niemetz et al. [20] generate 180 invertibility conditions and use the corresponding equivalences in a procedure for solving quantified bit-vector formulas. We consider only the invertibility conditions in which all bit-vectors have the same length. This corresponds to the invertibility conditions whose signatures are restricted by $\Sigma_{0.5}$ and there are 160 of these. The SMT solvers that use these conditions rely on the validity of the equivalences for all positive values of n. The following describes the different attempts at verifying these 160 properties ordered with respect to publication date.

1. Niemetz et al. [20] verified all 160 equivalences for $1 \leq n \leq 65$ using SMT solvers.
2. Niemetz et al. [22, 23] verified 110 of the 160 equivalences for arbitrary n by translating the equivalences to formulas in the theories of non-linear integer arithmetic and uninterpreted functions. This approach still failed on 50 of the equivalences.
3. Ekici et al. [11, 12] formalized these equivalences in the Rocq ITP using dependent types and proved 19 equivalences, 12 of which were from the 50 unproved equivalences (completing the verification of all IEs in Σ_0).
4. Berger et al. [7] extended the framework of [22, 23] and verified 3 more equivalences.

In this work, we have completed all missing proofs, by proving the 35 IEs that were previously unverified, for every bit-width.

2.4 Large Language Models

Large Language Models (LLMs) are large-scale deep-learning neural networks — typically built on the Transformer architecture [29] — trained on vast text datasets to predict subsequent tokens, enabling them to process and generate highly sophisticated human language [8]. However, LLMs are inherently unreliable because their probabilistic architecture optimizes for statistical likelihood rather than absolute truth [16], making "hallucinations" an inevitable limitation [31].

Despite this unreliability, as long as the LLM hasn't (1) altered the stated theorem being proved and (2) introduced any new axioms that extend Rocq's trusted computing base (TCB), we can be confident in the soundness of the proofs, thanks to the principle of proof irrelevance

$\ell[x]$	$=$	$\neq$	$<_u$	$>_u$	$\leq_u$	$\geq_u$	$<_s$	$>_s$	$\leq_s$	$\geq_s$
$-x \bowtie t$	✓✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\sim x \bowtie t$	✓✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$x \,\&\, s \bowtie t$	✓	✓	✓	✓	✓	✓	**✓**	**✓**	**✓**	**✓**
$x \mid s \bowtie t$	✓	✓	✓	✓	✓	✓	**✓**	**✓**	**✓**	**✓**
$x \ll s \bowtie t$	✓	✓	✓	✓	✓	✓	**✓**	**✓**	**✓**	**✓**
$s \ll x \bowtie t$	✓✓	✓	✓	✓	✓	✓	**✓**	✓	**✓**	✓
$x \gg s \bowtie t$	✓✓	✓	✓	✓	✓	✓	✓	**✓**	✓	**✓**
$s \gg x \bowtie t$	✓✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$x \gg_a s \bowtie t$	✓	✓	✓	✓	✓	✓	**✓**	✓	**✓**	✓
$s \gg_a x \bowtie t$	✓✓	✓	✓	✓	✓	✓	**✓**	**✓**	**✓**	✓
$x + s \bowtie t$	✓✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$x \cdot s \bowtie t$	✓	**✓**	✓	**✓**	✓	**✓**	**✓**	**✓**	**✓**	**✓**
$x \div s \bowtie t$	✓	✓	✓	✓	✓	**✓**	✓	✓	✓	✓
$s \div x \bowtie t$	✓	✓	✓	✓	✓	✓	✓	✓	✓	**✓**
$x \bmod s \bowtie t$	✓	✓	✓	✓	✓	✓	**✓**	✓	**✓**	✓
$s \bmod x \bowtie t$	**✓**	✓	✓	✓	✓	✓	✓	**✓**	✓	**✓**

Table 2: Proofs of arbitrary-width invertibility equivalences using different approaches.

in Rocq — the idea that any two proofs of the same proposition are propositionally equal. As such, LLMs and, by extension, multimodal conversational assistants like Gemini [26] and agentic coding assistants like Claude Code [2], are suitable for our project's goal of formal proof generation. Others have also noticed this synergy, resulting in a growing body of research exploring the potential of pairing LLMs with ITPs (see, e.g., [6, 30, 32]).

3 Results

3.1 Summary

Table 2 summarizes the verification of the 160 invertibility equivalences that we consider.[1] $\bowtie$ ranges over the given predicate symbols. ✓ means that the invertibility equivalence was successfully verified in Rocq [12, 11] in previous work; ✓ indicates that the corresponding equivalence was verified by one of the approaches that used SMT solvers [23, 22, 7]; **✓** is used for IEs that were proved using Rocq as part of this work.

3.2 Extensions to the bit-vector Library

We use and extend the `BVList` library [12] — a Rocq library that models bit-vectors and operations over bit-vectors, and provides various proofs about bit-vectors. The library uses a 2-layered approach to model bit-vectors — underneath the dependently-typed definition of bit-vectors (`bitvector` in Section 2.2) is a simply-typed (or non-dependently-typed) definition.

[1] The extended `BVList` library and all files relevant to our usage of Claude Code for this project are publicly available at https://github.com/HaTruong05/invCondProofs/tree/lpar26.

Proofs are produced for simply-typed bit-vectors, and lifted up to proofs of dependently-typed bit-vectors using a functor. Bit-vectors in `BVList` are specified using the little-endian notation. This matches the internal representation of bit-vectors in SMT solvers. For a more elaborate explanation of the representation of bit-vectors in `BVList`, see Ekici et al. [12].

In this section we enumerate our extensions to the library that allowed us to prove the remaining invertibility equivalences.

Signed Comparison Operators

We added $>_s$, $\leq_s$, and $\geq_s$ to `BVList` (it already contained $<_s$). Similar to the formalization of the unsigned comparison operators (described in detail for $\leq_u$ and $\geq_u$ in Ekici et al. [12]), these operators are defined over multiple *layers* - at the top is the dependently typed definition; underneath it, a function ensures that the bit-vectors being compared have the same length; yet another function reverses both bit-vectors for easier comparison starting from the most-significant bits of the bit-vectors; finally, a function at the bottom-most layer does a bit-wise comparison of the two bit-vectors from the most to least-significant bits.

Conversions between Bit-Vectors and Integers

We added two conversion functions from bit-vectors to integers:

- `bv2int` - a conversion from unsigned bit-vectors to their corresponding integer values.
- `sbv2int` - a conversion from two's complement signed bit-vectors to their corresponding integer values.

We used `bv2int` in many of the invertibility equivalence proofs over multiplication that we did manually (without AI assistance). It helped to reason about multiplication over `Z` — the Rocq integer type — and use lemmas from the Rocq library over integer multiplication.

For the invertibility conditions over signed comparison predicates, we needed `sbv2int` — a conversion from bit-vectors to integers that would take the signed interpretation of the bit-vectors. This allowed us to also soundly leverage properties over Rocq's integer type for the equivalences that contained signed comparators.

Unsigned Division and Remainder

We added $\div$ and mod, using the definitions from Figure 1. Both definitions first check whether the size of the operands match. Both operators also follow SMT-LIB 2 semantics when the divisor is 0 (expressed in `BVList` using the `mk_list_false` function). Unlike previous arithmetic operators, which were defined in `BVList` in terms of a *bit-blasting semantics* where the bit-vector operator is expressed as a propositional operation that composes all the bit-level operations over the operands, $\div$ and mod were defined simply by converting the operands to natural numbers and using the corresponding division and remainder operators for natural numbers (`N.div` and `N.modulo`). We found that the more complex the bit-blasting semantics, the more natural it is to reason about the operator after having soundly converted the operands to natural number or integers. In fact, even though multiplication was defined in `BVList` by bit-blasting, most of our invertibility equivalence proofs used the `bv2int` function so that reasoning over the operators could be performed using their integer interpretations.

```
Definition udiv_list (a b : list bool) : list bool :=
if beq_list b (mk_list_false (length b)) then mk_list_true (length a)
else N2list (N.div (list2N a) (list2N b)) (length a).

Definition bv_udiv (a b : bitvector) : bitvector :=
if ((@size a) =? (@size b)) then udiv_list a b
else nil.

Definition urem_list (a b : list bool) : list bool :=
if beq_list b (mk_list_false (length b)) then a
else N2list (N.modulo (list2N a) (list2N b)) (length a).

Definition bv_urem (a b : bitvector) : bitvector :=
if ((@size a) =? (@size b)) then urem_list a b
else nil.
```

Figure 1: Definitions of $\div$ and mod.

4 Proof Techniques

In terms of how much AI-assistance we used, our proofs can be classified as manual (no AI assistance), supported by AI (where we used Gemini to fill in some gaps in our proofs) or produced by AI with minimal human supervision (using Claude Code).

All our proofs followed the same general approach: the equivalence was proved over simply-typed bit-vectors by proving separately, the left-to-right and right-to-left implications that the equivalence is composed of. In keeping with previous work, this was the more challenging and interesting part of proofs. The corresponding proof over dependently-typed bit-vectors is straight-forward to derive from this. In fact, we derived these for all 35 of the proofs we added using Claude Code.

In the following, we elaborate on each of the 3 techniques we used for completing our proofs.

4.1 Manual Proofs

6 of the proofs were produced manually, without any LLM assistance. 3 of these were over the $<_s$ predicate and operators from Σ_0 and 3 of them were over the bit-vector multiplication operator. For proving properties over bit-vector multiplication, we found it useful to reason about them by interpreting them as integers and using Rocq's integer multiplication operation — `Z.mul`. This also allowed us to use many properties about `Z.mul` from the Rocq standard library. To maintain Rocq's TCB, we proved the translation from bit-vectors to integers sound for the multiplication operator. The following lemma played a central role in proving the `bv2int` translation sound:

$$\forall(x\ y\ :\ \texttt{bitvector}\ n),\ \texttt{bv2int}(x \cdot y)\ \%\ 2^n\ =\ (\texttt{bv2int}(x)\ *\ \texttt{bv2int}(y))\ \%\ 2^n$$

where `%` and `*` are the modulo and multiplication operators over Rocq's integer type (`Z`).

4.2 Proofs Supported by AI

Despite their ability to hallucinate in their responses, we decided to use AI assistants to help with our proofs since we had the assurance from Rocq that it would not accept an unsound

proof as long as we didn't add any axioms to it. We first employed Google Gemini to help us with our proofs. We used it interactively via its web interface — with Gemini 3 Pro as the chosen model [14] — for many different tasks: to translate whiteboard proofs into Rocq proofs; to fill in part of a larger proof; to come up with ideas to prove a particular lemma. While this was undoubtedly useful — it led to the completion of 11 proofs over the course of a 10-week term — it had its limitations. For the more "challenging" (we admit to using a subjective interpretation of the word) proofs, it needed a lot of interaction, and needed to be told when it was taking an obviously incorrect path. Often, we had to remind Gemini of problems that it had already seemed to have addressed. The issue that restricted its usage the most was the fact that the `BVList` library had tens of thousands of lines of proofs, and Gemini did not have access to these lines. We were concerned that exposing Gemini assistant to the entire library would trigger long-context vulnerabilities [25, 17], resulting in its inability to focus on the relevant information needed for each proof. As shown in [15], Gemini is not exempt from such vulnerabilities, and its performance may also degrade as more prompts are issued. Thus, we instead settled on the approach of asking Google Gemini what information it needs, searching for it ourselves, and passing it to the AI.

4.3 Proofs Produced by AI

To overcome the limitations encountered while using Gemini as a chatbot, we opted for Claude Code on the Pro subscription plan, running it directly in the command-line terminal. All the proof work we initiated with Claude Code uses Sonnet 4.6 [1]. Claude Code sometimes spawned subagents running on Haiku 4.5 [1] for auxiliary tasks. We initialized a Claude Code project within the directory containing the Rocq files `BVList.v`, `InvCond.v` and `DepInvCond.v`. `BVList.v` is the Rocq file containing `BVList` library, `InvCond.v` contains proofs of all invertibility equivalences over simply-typed bit-vectors proved using lemmas from `BVList`, and `DepInvCond.v` contains all the invertibility equivalence proofs over dependently-typed bit-vectors. Initializing the Claude project (1) gives access to all files within that directory to the AI (2) creates another directory storing logs of all conversations with Claude Code initiated in the project directory in JSONL format.

We put the necessary instructions and context for all proof work in a markdown file named `CLAUDE.md`. Doing this gave Claude some context, so we didn't have to specify instructions to Claude for every new invertibility equivalence. Instead, we simply gave it the following prompt: "Prove `<invertibility equivalence>`. Follow the workflow in CLAUDE.md."

Claude Code succeeded in (1) defining $\div$ and mod , (2) proving 18 invertibility equivalences, taking on average 2 hours per proof, and (3) deriving the proofs of all 35 equivalences over dependently-typed bit-vectors from the corresponding proofs over simply-typed bit-vectors in under 40 minutes.

For all AI-generated proofs, we were able to prevent added unsoundness by ensuring that (1) the AI wasn't modifying the statement of the property being proved (2) it wasn't adding any unproved (admitted) lemmas. This was sufficient due to the soundness guarantees of Rocq's TCB. However, this TCB does not protect against unsoundness introduced by incorrectly defined operators which could propagate through to unsound proofs that could still be accepted by Rocq. As a consequence, we spent some time manually verifying the definitions of $\div$ and mod to ensure they were consistent with the corresponding SMT-LIB 2 definitions. These have also been presented in 1.

Niemetz et al. [21] found that 3 of the invertibility conditions presented in Niemetz et al. [22] were invalid for bit-vectors of length 1 (but valid for all other lengths). For length 1, a

specialized IC was given. We tried to prove the general version of the third one:

$$\forall s, t : \sigma_{[n]}.\ (s \geq_s 0 \Rightarrow s \geq_s t) \wedge (s <_s 0 \Rightarrow s \gg 1 \geq_s t) \Leftrightarrow (\exists x : \sigma_{[n]}.\ s \div x \geq_s t)$$

Claude was able to corroborate the error by giving us a counter-example for the equivalence for $n = 1$ and instead proving the equivalence for a refined invertibility condition:

$$(n = 0 \Rightarrow s \geq_s t) \wedge (n > 0 \Rightarrow (s \geq_s 0 \Rightarrow s \geq_s t) \wedge (s <_s 0 \Rightarrow (s \gg 1 \geq_s t)))$$

5 Conclusion and Future Work

From the invertibility equivalences produced by Niemetz et al. [20], we completed the verification for arbitrary bit-widths of all 160 invertibility equivalences that only contain bit-vectors of identical widths. We did this by verifying the 35 previously unverified equivalences in Rocq. We also verified 3 other equivalences in Rocq that were only verified by SMT solvers previously. We used the `BVList` library that models SMT-LIB 2 bit-vectors, and extended it with the predicates $>_s$, $\leq_s$ and $>_s$, and the bitvector operators $\div$ and mod. From our 35 Rocq proofs, 6 were produced without any AI assistance, 11 were produced using assistance from Google Gemini as a chatbot, and 18 were produced by Claude Code with minimal human supervision.

One natural way of extending our work would be to prove those of the 20 invertibility equivalences over $\circ$ (bit-vector concatenation) that remain unverified. This would require modifying the general structure of our equivalences. Another useful change to the library would be to ensure that all definitions of arithmetic operators are defined consistently — either using bit-blasting semantics, or using a conversion to natural numbers or integers. Currently, $+$, $-$, and $\cdot$ are defined using bit-blasting, while $\div$ and mod are defined by converting the operators to natural numbers. `BVList` can specify multiple definitions for the same operator and a proof of their equivalence allows a user to use whichever definition they prefer (in fact, the library already has multiple definitions for each shift operator). So whichever semantics we choose for all arithmetic operators, we can still keep the alternative definitions, since they might be useful for proving some properties.

Especially since the use of LLMs have accelerated our proving process, another direction of future work would be to prove all invertibility equivalences that have only been verified by SMT solvers. Since ITPs like Rocq maintain a reliable trusted computing base that our proofs don't violate, properties proved in them come with higher guarantees than those verified by SMT solvers.

References

[1] Anthropic. Choosing a model: Claude Sonnet and Claude Haiku frameworks. https://platform.claude.com/docs/en/about-claude/models, 2026. Accessed: 2026-06-17.

[2] Anthropic. Claude code: Overview and agentic architecture. https://code.claude.com/docs/en/overview, 2026. Accessed: 2026-06-17.

[3] 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.

[4] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[5] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, Frontiers in Artificial Intelligence and Applications, pages 1267–1329. IOS Press, 2021.

[6] Guillaume Baudart, Marc Lelarge, Tristan Stérin, and Jules Viennot. Putnam 2025 problems in rocq using opus 4.6 and rocq-mcp, 2026.

[7] Zvika Berger, Yoni Zohar, Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Bit-precise reasoning with parametric bit-vectors. In Jeremias Berg and Jakob Nordström, editors, *28th International Conference on Theory and Applications of Satisfiability Testing, SAT 2025, Glasgow, Scotland, August 12-15, 2025*, volume 341 of *LIPIcs*, pages 4:1–4:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[9] Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ansi-c programs. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 168–176, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[10] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. Smtcoq: A plug-in for integrating SMT solvers into coq. In *Proceedings of 29th International Conference on Computer Aided Verification (CAV 2017)*, volume 10427 of *Lecture Notes in Computer Science*, pages 126–133. Springer, 2017.

[11] Burak Ekici, Arjun Viswanathan, Yoni Zohar, Clark W. Barrett, and Cesare Tinelli. Verifying bit-vector invertibility conditions in coq (extended abstract). In Giselle Reis and Haniel Barbosa, editors, *Proceedings Sixth Workshop on Proof eXchange for Theorem Proving, PxTP 2019, Natal, Brazil, August 26, 2019*, volume 301 of *EPTCS*, pages 18–26, 2019.

[12] Burak Ekici, Arjun Viswanathan, Yoni Zohar, Cesare Tinelli, and Clark W. Barrett. Formal verification of bit-vector invertibility conditions in coq. In Uli Sattler and Martin Suda, editors, *Frontiers of Combining Systems - 14th International Symposium, FroCoS 2023, Prague, Czech Republic, September 20-22, 2023, Proceedings*, Lecture Notes in Computer Science, pages 41–59. Springer, 2023.

[13] Herbert B. Enderton. Chapter two - first-order logic. In Herbert B. Enderton, editor, *A Mathematical Introduction to Logic (Second Edition)*, pages 67 – 181. Academic Press, Boston, second edition edition, 2001.

[14] Google DeepMind. Gemini 3 pro model card. Model card, Google DeepMind, 2025. Accessed: 2026-06-17.

[15] Google DeepMind. Gemini 3 pro technical report and architecture model card. Technical report, Google DeepMind, 2025. Accessed: 2026-06-18.

[16] Alon Jacovi and Yoav Goldberg. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? *CoRR*, abs/2004.03685, 2020.

[17] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

[18] Aina Niemetz and Mathias Preiner. Bitwuzla. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22,*

2023, Proceedings, Part II, volume 13965 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2023.

[19] Aina Niemetz, Mathias Preiner, and Armin Biere. Propagation based local search for bit-precise reasoning. *Formal Methods Syst. Des.*, 51(3):608–636, 2017.

[20] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark Barrett, and Cesare Tinelli. Solving quantified bit-vectors using invertibility conditions. In *Proceedings of 30th International Conference on Computer Aided Verification (CAV 2018)*, pages 236–255, 2018.

[21] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. On solving quantified bit-vector constraints using invertibility conditions. *Formal Methods Syst. Des.*, 57(1):87–115, 2021.

[22] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Yoni Zohar, Clark W. Barrett, and Cesare Tinelli. Towards bit-width-independent proofs in SMT solvers. In *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2019.

[23] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Yoni Zohar, Clark W. Barrett, and Cesare Tinelli. Towards satisfiability modulo parametric bit-vectors. *J. Autom. Reason.*, 65(7):1001–1025, 2021.

[24] Christine Paulin-Mohring. Introduction to the Calculus of Inductive Constructions. In Bruno Woltzenlogel Paleo and David Delahaye, editors, *All about Proofs, Proofs for All*, volume 55 of *Studies in Logic (Mathematical logic and foundations)*. College Publications, January 2015.

[25] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context, 2023.

[26] Gemini Team. Gemini: A family of highly capable multimodal models, 2025.

[27] The Rocq Development Team. *The Rocq Proof Assistant*, 2026. Version 9.0, formerly known as Coq.

[28] David Trabish and Shachar Itzhaky. Enhancing symbolic execution with machine-checked safety proofs. In *Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP '26, page 294–308, New York, NY, USA, 2026. Association for Computing Machinery.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[30] Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models, 2022.

[31] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is inevitable: An innate limitation of large language models, 2025.

[32] Yedi Zhang, Yufan Cai, Xinyue Zuo, Xiaokun Luan, Kailong Wang, Zhe Hou, Yifan Zhang, Zhiyuan Wei, Meng Sun, Jun Sun, Jing Sun, and Jin Song Dong. Position: Trustworthy AI agents require the integration of large language models and formal methods. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 82441–82459. PMLR, 13–19 Jul 2025.