

PBLean: Pseudo-Boolean Proof Certificates for Lean 4

Stefan Szeider

Algorithms and Complexity Group, TU Wien, Vienna, Austria

Abstract

We present PBLean, a method for importing VERIPB pseudo-Boolean (PB) proof certificates into Lean 4. The key to our approach is *reflection*: a Boolean checker function. We prove the soundness of this function fully in Lean and execute it as compiled native code. Our method scales to proofs with tens of thousands of steps that would exhaust memory under explicit proof-term construction. Our checker parses all VERIPB kernel rule types and verifies unsatisfiability by a combination of cutting-plane derivations, proof-by-contradiction subproofs, and redundancy/dominance for symmetry breaking; soundness is established in Lean for the rules used along the UNSAT path. In contrast to external verified checkers that produce verdicts, our integration yields Lean theorems. These can serve as composable lemmas in larger formal developments. To obtain theorems about the original combinatorial problems rather than about PB constraints alone, we support verified encodings. Since both the constraint translation and its correctness proof are formalized in Lean, this closes the trust gap between solver output and problem semantics. We demonstrate the approach on various combinatorial problems.

Keywords

Pseudo-Boolean reasoning, Proof certificates, Lean 4, Cutting planes, Reflection

1. Introduction

Pseudo-Boolean (PB) solvers handle linear constraints over Boolean variables. They have become prominent tools for combinatorial optimization and verification [1]. Since these solvers carry out sophisticated derivations, independent verification of their results is particularly desirable. However, PB reasoning involves cutting-planes proof steps that cannot be captured efficiently by resolution-based formats.

Haken's classical result [2] shows that the pigeonhole principle requires exponential-size resolution proofs. Polynomial-size cutting-plane proofs, in contrast, exist [3]. Beyond pigeonhole, cutting planes also efficiently capture XOR/parity constraints [4] and certified symmetry breaking [5]. These are reasoning methods that lack efficient resolution-based proof logging. This motivated the VERIPB proof format [6, 5], which is based on cutting planes. This format is now used by solvers such as ROUNDINGSAT for certified PB solving.

The LRAT format [7] enables SAT solvers to produce clausal certificates that can be checked in theorem provers. Lean's FromLRAT module [8] imports such certificates and outputs kernel-checked theorems. Heule and Scheucher [9] established the empty hexagon theorem using SAT solving with verified certificate checking. Subercazeaux et al. [10] subsequently formalized this encoding in Lean and verified the reduction from geometry to SAT. For pseudo-Boolean reasoning, a similar integration has been missing.

Why not compile PB reasoning into clausal proofs and reuse LRAT? Bryant et al. [11] showed that this is possible via BDD compilation. Their approach handles classical hard cases such as pigeonhole and parity efficiently. However, the compilation requires coordinating a PB solver with a BDD-based clausal proof generator, and whether it preserves efficiency for all PB reasoning remains open. Direct VERIPB import avoids this detour.

The CAKEPB project [12] provides a verified checker extracted from HOL4 via CakeML. It produces a standalone binary that outputs verdicts. The main advantage of CAKEPB is its small trusted computing

17th Pragmatics of SAT International Workshop 2026 (PoS'26)

✉ stefan.szeider@tuwien.ac.at (S. Szeider)

🌐 <https://www.ac.tuwien.ac.at/people/szeider/> (S. Szeider)

🆔 0000-0001-8994-1656 (S. Szeider)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

base enabled by verified compilation. However, the verified result (a binary verdict) cannot serve as a lemma in larger formal developments.

A proof certificate establishes that certain PB constraints are unsatisfiable. But it says nothing about whether those constraints correctly model the intended problem. An encoding bug can produce unsatisfiable constraints even when solutions exist. We address this gap with *verified encodings*: both the constraint translation and its correctness proof are entirely in Lean. This ensures that certificate import yields theorems about the original problem, not merely about PB constraints.

We present PBLEAN, a VERIPB integration for Lean 4. It parses kernel-format proofs and verifies them via *reflection*. Instead of building Lean proof terms for each derivation step, we define a Boolean checker function and prove its soundness once. Lean’s `native_decide` then runs the checker as compiled code. This follows the pattern of Lean’s `bv_decide` tactic [13]. This approach scales to instances that would exhaust memory under explicit proof construction. Our implementation covers all VERIPB kernel rule types, including proof-by-contradiction subproofs and redundance/dominance for symmetry breaking. Soundness is established in Lean for the rules used in unsatisfiability certificates, with the dominance rule checked in its satisfiability-preserving (red-style) fragment. We apply the verified encoding workflow to independence numbers of Paley graphs and other combinatorial theorems to obtain end-to-end verified results.

2. Background

A pseudo-Boolean (PB) constraint has the form $\sum_{i=1}^n a_i \ell_i \geq d$, where $a_i, d \in \mathbb{N}$ are non-negative coefficients, and each ℓ_i is a literal (a variable x_i or its negation $\bar{x}_i$). Under an assignment σ , a literal evaluates to 1 if satisfied and 0 otherwise. The constraint is satisfied when the weighted sum reaches the degree d . For example, $3x_1 + 5x_2 \geq 4$ is satisfied by $\sigma(x_1) = \sigma(x_2) = 1$ (sum = 8 $\geq$ 4) but not by $\sigma(x_1) = 1, \sigma(x_2) = 0$ (sum = 3 < 4). Clauses are a special case: $x_1 \vee x_2 \vee \bar{x}_3$ corresponds to $x_1 + x_2 + \bar{x}_3 \geq 1$.

Cutting-planes reasoning derives new constraints via: (1) addition of constraints, (2) scalar multiplication by $k \in \mathbb{N}^+$, (3) division by k with the degree rounded up, and (4) saturation, which caps each coefficient at the degree. These rules are strictly more general than resolution [3]. Here, resolution corresponds to addition with complementary literal cancellation. The power of cutting planes stems from the interplay of multiplication and division, which derives constraints unreachable by resolution alone [14].

The VERIPB proof format [6, 5] provides two modes: an augmented mode with high-level rules (symmetry breaking, redundance-based strengthening) and a kernel mode (version 3.0) with only primitive operations. The kernel format’s small rule set admits a soundness proof suitable for theorem prover integration. It supports: `pol` (cutting-planes operations in reverse Polish notation, including the weakening operator `w`), `rup` (reverse unit propagation with conflict hints), `pbc` (proof by contradiction with explicit subproofs), `red/dom` (redundance and dominance, adding equisatisfiable constraints via a substitution witness), `del/delc` (constraint deletion, sound by monotonicity), `sol/soli` (solution logging), and `conclusion` (UNSAT/SAT/BOUNDS verdict). Solvers such as ROUNDINGSAT produce augmented proofs; `veripb --elaborate` translates them to kernel format.

Lean 4 [15] provides a metaprogramming framework where tactics run in monads such as `MetaM`. Code in `MetaM` constructs `Expr` values (Lean’s internal representation of proof terms), and, upon type-checking, Lean’s kernel verifies their correctness. We use this framework in two ways. The explicit proof-term approach builds `Expr` values from VERIPB proofs in untrusted meta-code, as in `FromLRAT`; the kernel checks the resulting term, so soundness reduces to the kernel and the soundness lemmas of Section 3. The reflection approach instead defines a Boolean checker as a Lean function whose soundness is proved against its functional specification; the parser that turns the proof string into the data structures consumed by this function is part of that specification, so a parser bug can cause a sound rejection but cannot violate soundness.

3. Lean Formalization

For constraint representation, we use natural number arithmetic throughout. A *Literal* is `pos i` or `neg i`, and a *Term* is a pair $(a, \ell) \in \mathbb{N} \times \text{Literal}$; a *Constr* is a term list plus degree. Given a valuation $v : \mathbb{N} \rightarrow \text{Bool}$, we define $\text{evalLit}(v, \ell) \in \{0, 1\}$ (1 if satisfied) and $\text{evalSum}(v, T) = \sum_{(a, \ell) \in T} a \cdot \text{evalLit}(v, \ell)$ recursively to support structural induction; $\text{coeffSum}(T) = \sum_{(a, \ell) \in T} a$ is the sum of coefficients. A constraint (T, d) is satisfied when $d \leq \text{evalSum}(v, T)$.

Our 15 soundness lemmas are summarized in Table 1; every lemma is proved in Lean and verified by the kernel. Two details matter for matching VERIPB’s semantics: normalization must precede saturation and division, and the negation of $\sum a_i \ell_i \geq d$, under the precondition $d \leq \sum_i a_i$, is $\sum a_i \bar{\ell}_i \geq \sum_i a_i - d + 1$. The precondition keeps the resulting degree well-defined in $\mathbb{N}$ under truncating subtraction; if it fails ($d > \sum_i a_i$), the original constraint is already unsatisfiable and never arises as a negated target in our checker.

Table 1

Soundness lemmas for VERIPB kernel operations.

Operation	Lean Type
<i>Cutting planes</i>	
<code>add_sat</code>	$c_1.\text{sat } v \rightarrow c_2.\text{sat } v \rightarrow (c_1.\text{terms} ++ c_2.\text{terms}, c_1.\text{deg} + c_2.\text{deg}).\text{sat } v$
<code>mul_sat</code>	$c.\text{sat } v \rightarrow (\text{map } (\lambda(a, \ell). (k \cdot a, \ell)) c.\text{terms}, k \cdot c.\text{deg}).\text{sat } v$
<code>div_sat</code>	$0 < k \rightarrow c.\text{sat } v \rightarrow (\text{map } (\lambda(a, \ell). (\lceil a/k \rceil, \ell)) c.\text{terms}, \lceil c.\text{deg}/k \rceil).\text{sat } v$
<code>saturate_sat</code>	$c.\text{sat } v \rightarrow (\text{map } (\lambda(a, \ell). (\min(a, c.\text{deg}), \ell)) c.\text{terms}, c.\text{deg}).\text{sat } v$
<i>Normalization</i>	
<code>cancel_pair_sat</code>	Let $m = \min(a, b)$ with $m \leq d$. $(\dots (a, \ell) \dots (b, \bar{\ell}) \dots, d).\text{sat } v \rightarrow (\dots (a-m, \ell) \dots (b-m, \bar{\ell}) \dots, d-m).\text{sat } v$
<code>remove_zero_sat</code>	$(\text{pre} ++ (0, \ell)::\text{post}, d).\text{sat } v \rightarrow (\text{pre} ++ \text{post}, d).\text{sat } v$
<code>merge_terms_sat</code>	$(\dots (a, \ell) \dots (b, \ell) \dots, d).\text{sat } v \rightarrow (\dots (a+b, \ell) \dots, d).\text{sat } v$
<i>Axioms and contradiction</i>	
<code>lit_axiom_pos/neg</code>	$\forall v. ([(1, \ell)], 0).\text{sat } v$
<code>contra_unsat</code>	$\text{coeffSum}(c) < c.\text{deg} \rightarrow \neg c.\text{sat } v$
<i>Negation and proof by contradiction</i>	
<code>negate_sat_of_not_sat</code>	$c.\text{deg} \leq \text{coeffSum}(c) \rightarrow \neg c.\text{sat } v \rightarrow c.\text{negate}.\text{sat } v$
<code>propagation_forces</code>	$(T, d).\text{sat } v \rightarrow \text{coeffSum}(T \setminus \{(a, \ell)\}) < d \rightarrow \text{evalLit}(v, \ell) = 1$
<code>pbcsound</code>	$C.\text{deg} \leq \text{coeffSum}(C) \rightarrow (\forall v. F.\text{allSat } v \rightarrow C.\text{negate}.\text{sat } v \rightarrow \perp) \rightarrow F.\text{proof } C$
<i>Redundance (red/dom)</i>	
<code>applySubstConstr_sat_rev</code>	$C _{\omega}.\text{sat } v \rightarrow C.\text{sat } (\omega(v))$
<code>constr_sat_applyValuation_noSubst</code>	$\text{unaffected}(\omega, C) \rightarrow C.\text{sat } v \rightarrow C.\text{sat } (\omega(v))$

We give two verification approaches with complementary trade-offs.

Explicit proof terms. The first approach builds a Lean Expr for the target theorem by composing soundness lemmas. The construction maintains a *proof accumulator*: a finite map from constraint identifiers to Exprs witnessing $c.\text{sat } v$ for each derived constraint c , where v is the universally quantified valuation in scope. For `po1` steps, the checker traverses the RPN expression on a stack: ID tokens push the corresponding witness from the accumulator, while addition, multiplication, division, and saturation apply `add_sat`, `mul_sat`, `div_sat`, and `saturate_sat` to the witnesses on top. For `rup` steps, the target

is negated and combined with the listed hint constraints. The proof constructor maintains a local list of literal facts $evalLit(v, \ell) = 1$ accumulated by previous propagations on this RUP step. Each hint is processed by computing its *residual* (T', d') with respect to these facts: terms whose literal is already satisfied are removed and their coefficients subtracted from d' , while terms whose literal is forced false are removed without changing d' . A residual literal $\ell \in T'$ is propagated via *propagation_forces* when $coeffSum(T' \setminus \{(a, \ell)\}) < d'$, yielding a new fact $evalLit(v, \ell) = 1$. A contradiction is reached when the full residual already satisfies $coeffSum(T') < d'$, at which point *contra_unsat* closes the RUP. The partial assignment is therefore not a separate object: it is the (typed) list of literal facts under the universally quantified valuation v . For pbc subproofs, the accumulator is extended with the negated target as an additional hypothesis and the inner steps are verified recursively; on completion, *pbc_sound* discharges the assumption. Lean’s kernel type-checks the final Expr; as the soundness lemmas are themselves kernel-checked, the trusted base is the kernel together with the faithfulness of their statements to VERIPB’s semantics. The disadvantage is that the accumulated proof terms exhaust memory for instances with thousands of constraints.

Reflection. To overcome this limitation, our second approach follows the pattern of Lean’s *bv_decide* [13]. We define a Boolean function $check(F, \pi)$ that takes a constraint set F and a proof string π , replays all derivation steps, and returns *true* iff a contradictory constraint is derived. We then prove a soundness theorem: if $check(F, \pi) = true$ then F is unsatisfiable. This theorem is fully proved in Lean without custom axioms. Lean’s *native_decide* runs the checker as compiled code, yielding a kernel proof term consisting solely of the soundness theorem applied to the computation witness. The resulting theorem has the same trust basis as *bv_decide* and *native_decide*—namely, Lean’s standard axioms (*propext*, *Classical.choice*, *Quot.sound*) plus *Lean.ofReduceBool*—and scales to proofs with thousands of steps.

Trust model. The reflection-based checker relies on *Lean.ofReduceBool*, the axiom asserting that values produced by Lean’s native compiler (via *Lean.reduceBool*) agree with the term they represent. Lean’s kernel does not verify native compilation; a bug in the compiler could in principle produce an incorrect evaluation result. Our soundness theorem is proved against the purely functional specification of the checker; an *@[implemented_by]* annotation swaps in an array-based runtime replacement during native evaluation (Section 4), which becomes part of the trusted computation just as the compiler does. The explicit proof-term approach (our first method above) avoids both *Lean.ofReduceBool* and the *@[implemented_by]* substitution; it serves as a fallback when these are unacceptable.

Redundance and dominance. Unlike *po1* and *rup*, which derive constraints *implied* by the current database, the *red* and *dom* rules add a constraint C whose addition need only preserve satisfiability. Each step provides a witness substitution ω together with subproofs: under the temporary assumption $\neg C$, the substituted constraint $C|_\omega$ must be derivable, and $D|_\omega$ must be derivable for every constraint D in the current database affected by ω . The checker validates each subproof in a context extended with $\neg C$ and verifies coverage of all affected constraints; for unaffected constraints, *constr_sat_applyValuation_noSubst* (Table 1) gives preservation directly. Combined with *applySubstConstr_sat_rev*, this yields the key invariant by case analysis on an assignment v satisfying the original database: if $v \models C$, then v already satisfies the augmented database; otherwise $v \models \neg C$, and the subproofs under $\neg C$ together with the substitution-preservation lemmas show that $\omega(v)$ satisfies the augmented database. We refer to this weaker invariant as *database satisfiability preservation* (DBSat): if the original database is satisfiable, then so is the augmented one. This is strictly weaker than the implication-based invariant (DBSound) maintained by *po1/rup*, but suffices for refutation: a contradiction in the augmented database implies unsatisfiability of the original. PBLEAN treats *dom* via the same substitution-coverage mechanism as *red*; the order/finite-descent argument required for full dominance over an objective is left to future work. We exercise *red* on $\neg PHP(3, 2)$ via a cyclic substitution witness for symmetry breaking.

4. Implementation

PBLEAN is a standalone Lean 4 project with no Mathlib dependency. All reported experiments use Lean v4.28.0-rc1; the code has since been ported to v4.30.0. The kernel layer (`PseudoBoolean.lean`, ~900 lines) consists of types and 15 soundness lemmas. The code, OPB encodings, and kernel proofs are available on GitHub.¹

The reflection-based checker (`Reflect.lean`, ~2500 lines) uses a line-by-line tokenizer to convert VERIPB proofs into an abstract syntax tree, handling nested pbc subproofs recursively. It maintains a constraint database organized as a hash map, with an `@[implemented_by]` annotation providing an array-based runtime replacement for performance during native evaluation; soundness is proved against the hash-map specification, and the replacement enters the trusted base on the same footing as the compiler. Each rule updates or queries this database: `po1` evaluates the RPN expression, `rup` derives a conflict by negating the target and applying PB unit propagation, `pbv` saves the database, adds the negated assumption, verifies inner steps recursively, then restores the snapshot, and `red/dom` verify coverage of affected constraints and process subproofs for each proof goal before adding the equisatisfiable constraint. The checker returns `true` only when a contradictory constraint arises, and we establish soundness by induction over proof steps, connecting the Boolean computation to the kernel lemmas.

To exploit the integration, one provides an OPB file and a kernel-format proof file. The command `veripb_reflect` parses both, runs the checker via `native_decide`, and registers the resulting unsatisfiability theorem under a given name. For domain-specific applications, wrapper commands such as `independent_set_reflect` additionally generate the encoding from a problem instance and compose the bridge theorem automatically.

5. Trusted Encoding

We demonstrate our approach on a problem where the encoding is also formalized in Lean, closing a trust gap. For SAT, encoding correctness was only recently formalized [10]; for PB reasoning, no such formalization existed. Figure 1 shows the overall workflow.

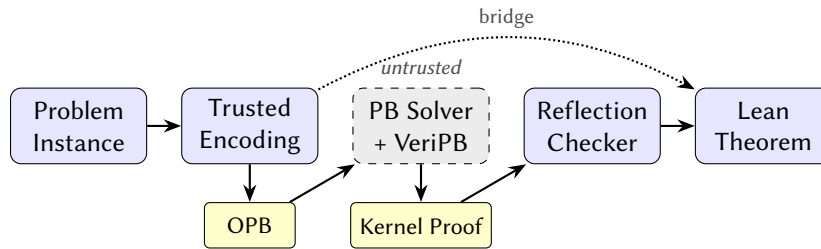


Figure 1: End-to-end workflow for trusted encodings. OPB is the standard input format for PB solvers. Solid boxes are verified in Lean; the dashed box is untrusted. The dotted arrow represents the bridge theorem composing encoding correctness with the PB refutation.

A verified encoding consists of three components: (1) an encode function translating problem instances to PB constraints, (2) a soundness theorem proving that unsatisfiability of the encoding implies non-existence of solutions, and (3) a bridge theorem composing encoding correctness with the imported VERIPB certificate to yield a theorem about the original problem.

As an example, we consider the independence number (the size of a largest independent set) of Paley graphs. For a prime $p \equiv 1 \pmod{4}$, the Paley graph $\text{Paley}(p)$ is defined as the graph with the vertex set $\mathbb{Z}_p$ and edges (i, j) whenever $j - i$ is a nonzero quadratic residue. These graphs are self-complementary with an independence number α of at most $\sqrt{p}$ by the Hoffman eigenvalue bound. The encoding for

¹<https://github.com/leansolving/pblean>

“does G have an independent set of size $\geq k$?” uses

$$\begin{aligned} \forall (u, v) \in E(G) : \bar{x}_u + \bar{x}_v &\geq 1 && \text{(independence)} \\ : x_1 + \dots + x_n &\geq k && \text{(cardinality)} \end{aligned}$$

The Lean formalization establishes that if these constraints are unsatisfiable, then G has no independent set of size $\geq k$. Combining the `VERIPB` certificate with this result yields a theorem about the graph. The `independent_set_reflect` command automates the whole pipeline: given a graph and a target size k , it produces a Lean theorem of the form $\neg \text{hasIndependentSet}(G, k)$.

6. Evaluation

We evaluate `PBLean` by measuring wall-clock time on an Apple M2 with cached `.olean` files. Measured times include parsing, checker execution via `native_decide`, and kernel verification. We use `ROUNDINGSAT` to solve, and `veripb --elaborate` to produce kernel-format proofs.

Table 2 shows results for independence numbers of Paley(p) for primes $p \equiv 1 \pmod{4}$ from 13 to 101.

Here the proofs rely on two PB rules: cutting-planes (`po1`) and reverse unit propagation (`rup`). The largest instance, Paley(101), has 2525 edges and about 63 000 proof lines, and is verified in about 3.5 minutes. A pigeonhole instance also exercises the `red` rule for symmetry breaking.

Table 2

Independent set on Paley graphs. p : prime, α : independence number. Vars: PB variables, Cstrs: PB constraints. Direct: explicit proof-term construction. Reflect: reflection-based checker. TO = timeout at 60 000 *ms*.

p	α	Vars	Cstrs	Solve (<i>ms</i>)	Elab (<i>ms</i>)	Lines	Direct (<i>ms</i>)	Reflect (<i>ms</i>)
13	3	13	40	59	12	35	1 150	802
17	3	17	69	38	4	66	2 310	802
29	4	29	204	53	5	233	19 922	926
37	4	37	334	105	6	554	TO	1 314
41	5	41	411	131	6	704	TO	1 536
53	5	53	690	421	13	1 848	TO	3 494
61	5	61	916	850	24	4 151	TO	6 743
73	5	73	1 315	3 145	64	9 808	TO	20 616
89	5	89	1 959	17 166	234	36 795	TO	96 234
97	6	97	2 329	21 673	272	43 214	TO	115 197
101	5	101	2 526	29 267	385	62 924	TO	200 460

A noteworthy byproduct of this evaluation is a formally verified proof that $\alpha(\text{Paley}(p))$ is not monotone in p . In particular, $\alpha(\text{Paley}(97)) = 6 > 5 = \alpha(\text{Paley}(101))$. Here, each equality combines the `VERIPB` upper bound with a lower bound from an explicit witness. This witness is verified by `native_decide`.

```
theorem paley_alpha_not_monotone :
  ∃ p q : Nat, p < q ∧ ∃ a b : Nat,
    independenceNumber (paley p) a ∧
    independenceNumber (paley q) b ∧ a > b
```

Recall from Section 3 that the explicit proof-term approach does not scale. The Direct column shows that Paley(29) already takes 20 s, and all instances with $p \geq 37$ exceed the 60 s timeout. In contrast, the reflection-based checker verifies Paley(53) in 3.5 s and Paley(101) in 200 s. The bottleneck appears to be the compiled checker execution rather than Lean’s kernel, which only checks the soundness theorem application.

We also compare against CAKEPB [12], a standalone verified VERIPB checker. On our instances it checks Paley(101) in under 1 s, against about 200 s for PBLEAN. This roughly $200\times$ gap is the price of producing a reusable Lean theorem rather than a binary verdict, a trade-off we revisit in Section 7.

Beyond Paley graphs, the reflection-based checker also handles four larger combinatorial impossibility results: the upper-bound sides of $S(3) = 13$ (Schur) and $W(2, 4) = 35$ (Van der Waerden), the bound $R(3, 4) \leq 9$ (Ramsey), and the impossibility of $L(2, 9)$ (Langford). Each verifies in under three minutes, with proof sizes up to 44 342 lines. These examples exercise larger or more general encodings: k -colorings for Schur and Van der Waerden, asymmetric clique parameters for Ramsey, and a larger instance of the existing Langford encoding.

A bin-packing instance further illustrates where native PB pays off over a clausal pipeline. Its encoding uses just 17 pseudo-Boolean inequalities over 60 variables with no auxiliary variables, whereas the standard sequential-weight-counter CNF encoding [16] requires roughly 960 variables and 2 617 clauses—an overhead that grows linearly with the bin capacity.

7. Related Work

CAKEPB [12] provides verified end-to-end subgraph solving in HOL4/CakeML. It includes both a PB proof checker and verified graph-problem encoders compiled to a standalone binary. As discussed in Section 1, the approaches are complementary. CAKEPB targets standalone verification with a small trusted computing base. PBLEAN yields Lean theorems that can serve as lemmas in larger formal developments. As reported in Section 6, CAKEPB checks our largest Paley instance about $200\times$ faster than PBLEAN; this gap reflects the overhead of our theorem-producing pipeline compared with a standalone verified checker. The VERIPB framework has been extended to certify MaxSAT preprocessing [17], solution-improving search [18], dynamic programming and decision diagrams [19], and pseudo-Boolean optimization [20]. These extensions all elaborate to the same kernel format that PBLEAN checks. Therefore, the proof-checking core is reusable. Importing them into Lean for decision-style applications would only require an appropriate front-end encoding, analogous to our independent set or bin packing examples. Optimization or solution-logging applications would additionally require Lean bridge theorems for objectives, solutions, and BOUNDS conclusions, as well as full dominance-with-order support. These are left to future work. The verified front-ends in CAKEPB [12] for subgraph problems demonstrate the viability of the encoding pattern.

An alternative to native PB proof checking is to compile PB reasoning to clausal proofs. Bryant et al. [11] showed that this is possible via BDD compilation. Their approach efficiently handles pigeonhole and parity instances via extension variables. Direct VERIPB import provides a simpler pipeline that works with the native proof format of PB solvers.

Approaches for SAT certificate checking in theorem provers include Heule et al.’s verified LRAT checker in ACL2 [7], Lammich’s GRAT in Isabelle/HOL [21], and Lean’s FromLRAT [8], whose architecture our integration closely follows. The empty hexagon theorem combined SAT solving with verified checking [9]; its encoding was later formalized in Lean [10].

Rather than checking proof certificates, one can verify the solver itself. Blanchette et al. [22] formalized CDCL in Isabelle/HOL as a chain of refinements from abstract calculus to executable code. Fleury and Lammich [23] extended this to IsaSAT, a verified SAT solver.

SMTCoq [24] imports SMT witnesses into Coq. Besson’s micromega [25] uses reflexive certificate checking for linear arithmetic in Coq. This approach is based on Farkas’ Lemma rather than cutting planes. Moreover, it does not import external PB proofs. A related project [26] formalizes general CSP definitions in Lean 4. It provides verified translations to MiniZinc and SMT-LIB solver formats. PBLEAN extends verified PB checking to Lean 4. It complements existing SAT checking in Isabelle and PB checking in HOL4.

8. Conclusion

We propose a Lean 4 integration for VERIPB pseudo-Boolean proof certificates. The key to PBLEAN’s scalability is reflection: a Boolean checker with a fully proved soundness theorem, evaluated by Lean’s native-code mechanism under the standard `Lean.ofReduceBool` assumption. As demonstrated in Section 6, this verifies proofs with tens of thousands of steps that would exhaust memory under explicit proof construction. Our largest instance, Paley(101) with about 63 000 proof lines, is verified in about 3.5 minutes.

With trusted encodings, users obtain end-to-end verified combinatorial results: the independence number of Paley(101) is a Lean proposition, not an external verdict. This composability, not directly available with standalone verified checkers, enables PB reasoning to serve as a building block in larger formalizations.

PBLEAN pays off when cutting-plane certificates are substantially smaller than the corresponding resolution proofs—pigeonhole, parity, and symmetry-broken instances [2, 4, 5], for which a clausal LRAT pipeline is at an asymptotic disadvantage; our pigeonhole example already falls in this regime.

Performance leaves room to improve: since the cost appears to be dominated by compiled checker execution rather than kernel checking, streaming proofs instead of building a full syntax tree and lowering the checker’s per-step overhead are promising directions. Further work includes a `veripb` tactic for proof automation, full dominance-with-order support, and integration with other Lean formalizations.

Acknowledgments

Research was carried out within the Cluster of Excellence *Bilateral AI* (10.55776/COE12) of the Austrian Science Funds (FWF).

Declaration on Generative AI

During the preparation of this work, the author used Claude (Anthropic) and GPT (OpenAI) for grammar and spelling check, code review, and literature search assistance. After using these tools, the author reviewed and edited the content as needed and takes full responsibility for the publication’s content.

References

- [1] J. Elffers, J. Nordström, Divide and conquer: Towards faster pseudo-boolean solving, in: J. Lang (Ed.), Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden, *ijcai.org*, 2018, pp. 1291–1299. URL: <https://doi.org/10.24963/ijcai.2018/180>. doi:10.24963/IJCAI.2018/180.
- [2] A. Haken, The intractability of resolution, *Theor. Comput. Sci.* 39 (1985) 297–308. URL: [https://doi.org/10.1016/0304-3975\(85\)90144-6](https://doi.org/10.1016/0304-3975(85)90144-6). doi:10.1016/0304-3975(85)90144-6.
- [3] W. J. Cook, C. R. Coullard, G. Turán, On the complexity of cutting-plane proofs, *Discret. Appl. Math.* 18 (1987) 25–38. URL: [https://doi.org/10.1016/0166-218X\(87\)90039-4](https://doi.org/10.1016/0166-218X(87)90039-4). doi:10.1016/0166-218X(87)90039-4.
- [4] S. Gocht, J. Nordström, Certifying parity reasoning efficiently using pseudo-boolean proofs, in: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2–9, 2021, AAAI Press, 2021, pp. 3768–3777. URL: <https://doi.org/10.1609/aaai.v35i5.16494>. doi:10.1609/AAAI.V35I5.16494.
- [5] B. Bogaerts, S. Gocht, C. McCreesh, J. Nordström, Certified symmetry and dominance breaking for combinatorial optimisation, in: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI

- 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelfth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022, AAAI Press, 2022, pp. 3698–3707. URL: <https://doi.org/10.1609/aaai.v36i4.20283>. doi:10.1609/AAAI.V36I4.20283.
- [6] J. Elffers, S. Gocht, C. McCreesh, J. Nordström, Justifying all differences using pseudo-boolean reasoning, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 1486–1494. URL: <https://doi.org/10.1609/aaai.v34i02.5507>. doi:10.1609/AAAI.V34I02.5507.
 - [7] M. Heule, W. A. Hunt Jr., M. Kaufmann, N. Wetzler, Efficient, verified checking of propositional proofs, in: M. Ayala-Rincón, C. A. Muñoz (Eds.), Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings, volume 10499 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 269–284. URL: https://doi.org/10.1007/978-3-319-66107-0_18. doi:10.1007/978-3-319-66107-0_18.
 - [8] Mathlib Contributors, FromLRAT: LRAT proof checking in Lean 4, <https://github.com/leanprover-community/mathlib4>, 2024. Mathlib.Tactic.Sat.FromLRAT.
 - [9] M. J. H. Heule, M. Scheucher, Happy ending: An empty hexagon in every set of 30 points, in: B. Finkbeiner, L. Kovács (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I, volume 14570 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 61–80. URL: https://doi.org/10.1007/978-3-031-57246-3_5. doi:10.1007/978-3-031-57246-3_5.
 - [10] B. Subercaseaux, W. Nawrocki, J. Gallicchio, C. R. Codel, M. Carneiro, M. J. H. Heule, Formal verification of the empty hexagon number, in: Y. Bertot, T. Kutsia, M. Norrish (Eds.), 15th International Conference on Interactive Theorem Proving, ITP 2024, Tbilisi, Georgia, September 9-14, 2024, volume 309 of *LIPICs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, pp. 35:1–35:19. URL: <https://doi.org/10.4230/LIPICs.ITP.2024.35>. doi:10.4230/LIPICs.ITP.2024.35.
 - [11] R. E. Bryant, A. Biere, M. J. H. Heule, Clausal proofs for pseudo-boolean reasoning, in: D. Fisman, G. Rosu (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I, volume 13243 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 443–461. URL: https://doi.org/10.1007/978-3-030-99524-9_25. doi:10.1007/978-3-030-99524-9_25.
 - [12] S. Gocht, C. McCreesh, M. O. Myreen, J. Nordström, A. Oertel, Y. K. Tan, End-to-end verification for subgraph solving, in: M. J. Wooldridge, J. G. Dy, S. Natarajan (Eds.), Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada, AAAI Press, 2024, pp. 8038–8047. URL: <https://doi.org/10.1609/aaai.v38i8.28642>. doi:10.1609/AAAI.V38I8.28642.
 - [13] Lean Contributors, `bv_decide`: Verified bitvector reasoning in Lean 4, <https://github.com/leanprover/lean4>, 2024. Std.Tactic.BVDecide.
 - [14] M. Vinyals, J. Elffers, J. Giráldez-Cru, S. Gocht, J. Nordström, In between resolution and cutting planes: A study of proof systems for pseudo-boolean SAT solving, in: O. Beyersdorff, C. M. Wintersteiger (Eds.), Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings, volume 10929 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 292–310. URL: https://doi.org/10.1007/978-3-319-94144-8_18. doi:10.1007/978-3-319-94144-8_18.
 - [15] L. de Moura, S. Ullrich, The lean 4 theorem prover and programming language, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings, volume 12699 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 625–635. URL: https://doi.org/10.1007/978-3-030-79876-5_37. doi:10.1007/978-3-030-79876-5_37.

1007/978-3-030-79876-5_37.

- [16] T. Philipp, P. Steinke, PBLib – A library for encoding pseudo-boolean constraints into CNF, in: M. Heule, S. A. Weaver (Eds.), *Theory and Applications of Satisfiability Testing – SAT 2015*, volume 9340 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 9–16. URL: https://doi.org/10.1007/978-3-319-24318-4_2. doi:10.1007/978-3-319-24318-4_2.
- [17] H. Ihalainen, A. Oertel, Y. K. Tan, J. Berg, M. Järvisalo, M. O. Myreen, J. Nordström, Certified MaxSAT preprocessing, in: C. Benz Müller, M. J. H. Heule, R. A. Schmidt (Eds.), *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I*, volume 14739 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 396–418. URL: https://doi.org/10.1007/978-3-031-63498-7_24. doi:10.1007/978-3-031-63498-7_24.
- [18] J. Berg, B. Bogaerts, J. Nordström, A. Oertel, T. Paxian, D. Vandesande, Certifying without loss of generality reasoning in solution-improving maximum satisfiability, in: P. Shaw (Ed.), *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, Girona, Spain, September 2-6, 2024*, volume 307 of *LIPICs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, pp. 4:1–4:28. URL: <https://doi.org/10.4230/LIPICs.CP.2024.4>. doi:10.4230/LIPICs.CP.2024.4.
- [19] E. Demirovic, C. McCreesh, M. J. McIlree, J. Nordström, A. Oertel, K. Sidorov, Pseudo-boolean reasoning about states and transitions to certify dynamic programming and decision diagram algorithms, in: P. Shaw (Ed.), *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, Girona, Spain, September 2-6, 2024*, volume 307 of *LIPICs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, pp. 9:1–9:21. URL: <https://doi.org/10.4230/LIPICs.CP.2024.9>. doi:10.4230/LIPICs.CP.2024.9.
- [20] W. Koops, D. L. Berre, M. O. Myreen, J. Nordström, A. Oertel, Y. K. Tan, M. Vinyals, Practically feasible proof logging for pseudo-boolean optimization, in: M. G. de la Banda (Ed.), *31st International Conference on Principles and Practice of Constraint Programming, CP 2025, Glasgow, Scotland, August 10-15, 2025*, volume 340 of *LIPICs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, pp. 21:1–21:27. URL: <https://doi.org/10.4230/LIPICs.CP.2025.21>. doi:10.4230/LIPICs.CP.2025.21.
- [21] P. Lammich, The GRAT tool chain - efficient (UN)SAT certificate checking with formal correctness guarantees, in: S. Gaspers, T. Walsh (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 457–463. URL: https://doi.org/10.1007/978-3-319-66263-3_29. doi:10.1007/978-3-319-66263-3_29.
- [22] J. C. Blanchette, M. Fleury, P. Lammich, C. Weidenbach, A verified SAT solver framework with learn, forget, restart, and incrementality, *J. Autom. Reason.* 61 (2018) 333–365. URL: <https://doi.org/10.1007/s10817-018-9455-7>. doi:10.1007/s10817-018-9455-7.
- [23] M. Fleury, P. Lammich, A more pragmatic CDCL for isasat and targetting LLVM (short paper), in: B. Pientka, C. Tinelli (Eds.), *Automated Deduction - CADE 29 - 29th International Conference on Automated Deduction, Rome, Italy, July 1-4, 2023, Proceedings*, volume 14132 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 207–219. URL: https://doi.org/10.1007/978-3-031-38499-8_12. doi:10.1007/978-3-031-38499-8_12.
- [24] B. Ekici, A. Mebsout, C. Tinelli, C. Keller, G. Katz, A. Reynolds, C. W. Barrett, Smtcoq: A plug-in for integrating SMT solvers into coq, in: R. Majumdar, V. Kuncak (Eds.), *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 126–133. URL: https://doi.org/10.1007/978-3-319-63390-9_7. doi:10.1007/978-3-319-63390-9_7.
- [25] F. Besson, Fast reflexive arithmetic tactics the linear case and beyond, in: T. Altenkirch, C. McBride (Eds.), *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, volume 4502 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 48–62. URL: https://doi.org/10.1007/978-3-540-74464-1_4. doi:10.1007/978-3-540-74464-1_4.
- [26] P. Manrique Merchan, S. Szeider, LeanCSP: Verified constraint satisfaction in Lean 4, <https://github.com/leansolving/leancsp>, 2025.