FERAT: A New Expansion-Based Certification Framework for Quantified Boolean Formulas

Marcel Simader
Johannes Kepler University
Austria, Linz
marcel.simader@jku.at

Adrián Rebola-Pardo
Johannes Kepler University
Austria, Linz
Vienna University of Technology
Austria, Vienna
adrian.rebola_pardo@jku.at

Martina Seidl
Johannes Kepler University
Austria, Linz
martina.seidl@jku.at

Abstract

To witness the correctness of unsatisfiablility results of SAT solvers, the powerful resolution asymmetric tautology (RAT) proof system has been introduced, for which efficient proof checkers are available. To harness the power of recent SAT technology for solving quantified Boolean formulas (QBFs), the extension of SAT with quantifiers over the Boolean variables, we introduce the proof system ∀-Exp+RAT. With this proof system, it becomes possible to use modern SAT solvers for expansion-based QBF solving, one of the most successful QBF solving paradigms. So far, expansion-based QBF solving relied on the resolution-based ∀-Exp+Res proof system which is less powerful than ∀-Exp+RAT.

Based on the ∀-Exp+RAT proof system, we present the new certification framework FERAT for generating and checking ∀-Exp+RAT certificates. In a detailed evaluation, we show that with the FERAT pipeline, more formula instances can be certified than with the previous FERP pipeline which relies on the ∀-Exp+Res proof system.

CCS Concepts

• **Theory of computation** → **Logic and verification**; **Proof theory**; *Proof complexity*.

Keywords

Quantified Boolean Formulas, Certification, Proof Checking

ACM Reference Format:
Marcel Simader, Adrián Rebola-Pardo, and Martina Seidl. 2025. FERAT: A New Expansion-Based Certification Framework for Quantified Boolean Formulas. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3672608.3707863

1 Introduction

Quantified Boolean formulas (QBFs), the extension of propositional logic (SAT) with universal and existential quantifiers over the Boolean variables, provide a powerful framework to encode and solve many problems with a complexity beyond NP. Examples are applications from formal verification and synthesis, two-player games, planning and scheduling, etc. (see [22] for a survey). Strongly inspired by the success story of SAT [5, 6], very efficient solving technology has been developed for solving the decision problem of QBFs, i.e., for deciding if a given formula is true or false. In this paper, we are dealing with the challenging task of ensuring that the result returned by a QBF solver is correct.

While there have been attempts to build verified SAT solvers [11, 24], i.e., solvers which are provably correct, these solvers are usually far less performant than state-of-the-art solvers. In order to guarantee the correctness of a solving result, modern SAT solvers follow a different approach. In case that the formula is satisfiable, the solver has to produce a model (an assignment of the Boolean variables under which the formula evaluates to true). In case the formula is unsatisfiable, the solver has to produce a proof that can be checked efficiently by an independent tool. Modern SAT solvers produce certificates in the form of DRAT proofs [25].

To certify the correctness of QBF solving results, an approach similar to that of SAT is possible. In contrast to SAT, where conflict-driven clause learning (CDCL) [23] is the predominant solving paradigm, the QBF landscape is more heterogeneous, involving QCDCL, the extension of CDCL for reasoning with quantifiers, expansion-based solving, clausal abstraction, as well as incomplete pre- and inprocessing techniques (see [2] for an overview). In this work, we focus on expansion-based solving [7, 17].

Expansion-based solving was one of the most successful solving paradigms introduced over the last few years [18]. Relying on classical expansion, one type of quantifier is eliminated in a truth-preserving manner, and the resulting formula is then solved by a SAT solver. As the elimination of all quantifiers leads to an exponential blowup of the formula size, modern solvers realize the expansion iteratively, in a counter-example guided abstraction refinement (CEGAR)-like fashion [9] that builds only a partial expansion sufficient to decide the truth value of the formula. Following [19], expansion-based solving can be characterized by a proof system called ∀-Exp+Res which consists of two rules: (1) the expansion rule necessary to obtain the (partial) propositional expansion, and (2) the resolution rule necessary to refute the propositional expansion. To the best of our knowledge, the ∀-Exp+Res proof system has only been implemented in the FERP framework [14], which provides a complete toolchain to generate and check certificates for false QBFs; true formulas can be handled by negating the formula. This framework, however, requires that the underlying SAT solvers produce propositional resolution proofs. This does not reflect the

This research was funded in part by the Austrian Science Fund (FWF) [10.55776/COE12] and the State of Upper Austria (LIT AI Lab).

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707863

state-of-the-art of SAT solving, because no modern SAT solver produces resolution proofs, but proofs based on the RAT proof system.[1] As a consequence, the most recent SAT solvers cannot be used in the FERP framework.

To overcome this restriction, we introduce the ∀-Exp+RAT proof system in section 3, replacing the resolution part of ∀-Exp+Res with resolution asymmetric tautology (RAT). This enables the construction of a more advanced certification framework, which is capable of integrating state-of-the art SAT solvers. We describe this system, called FERAT, in detail in section 4, after which we present our empirical findings in section 5. Finally, we summarize our findings with closing remarks in section 6.

2 Background

We consider QBFs $\Phi = \Pi.\phi$ in prenex conjunctive normal form (PCNF) where Π is a sequence of quantifier bindings $Q_1 x_1 \ldots Q_n x_n$, with $Q_i \in \{\forall, \exists\}$, and $x_i \neq x_j$ for $i \neq j$. We denote $\mathrm{Vars}(\Pi) = \{x_1, \ldots, x_n\}$, $\mathrm{UVars}(\Pi) = \{x_i \mid Q_i = \forall\}$, and $\mathrm{EVars}(\Pi) = \{x_i \mid Q_i = \exists\}$. The prefix induces a partial order on $\mathrm{Vars}(\Pi)$ variables with $x_i \leq_\Pi x_j$ for $i \leq j$.

The *matrix* ϕ is a propositional formula in *conjunctive normal form (CNF)*; that is, ϕ is a conjunction of clauses, a clause is a disjunction of literals, and a literal is a variable or the negation of a variable. Furthermore, PCNF formulas are *closed*, i.e. all variables in ϕ occur in $\mathrm{Vars}(\Pi)$. For convenience, we regard clauses as sets of literals and the matrix as a set of clauses. For a variable x we define $\mathrm{Var}(x) = \mathrm{Var}(\neg x) = x$.

Given variables $x_1, \ldots, x_n, y_1, \ldots, y_n$ and a propositional CNF formula F, we denote by $F[x_1/y_1, \ldots, x_n/y_n]$ the formula obtained by simultaneously substituting each occurrence of x_i by y_i in F. Sometimes we also perform a partial evaluation of F by assigning a variable x to a truth value $t \in \{\top, \bot\}$, denoted likewise by $F[x/t]$; in this case we implicitly assume the resulting formula is simplified accordingly (i.e. clauses containing $\top$ are removed, as well as occurrences of $\bot$ within clauses).

We recursively define a QBF to be *false* if it is either: *i)* $\bot$, or *ii)* $\forall x \Pi.\phi$, where either $\Pi.\phi[x/\top]$ or $\Pi.\phi[x/\bot]$ is false, or *iii)* $\exists x \Pi.\phi$, where both $\Pi.\phi[x/\top]$ and $\Pi.\phi[x/\bot]$ are false. Otherwise, the QBF is *true*. In the rest of the paper, we use the following false QBF as a running example:

$$\Phi = \exists x \forall a \exists y.((x \vee a \vee y) \wedge (\neg x \vee \neg a \vee y) \wedge (\neg y)) \qquad (1)$$

Expansion-based QBF Solving. QBF solving is the problem of deciding whether a QBF is true or false. Expansion-based solving decides this problem by reducing it to a single propositional satisfiability check as follows. First, either of the two quantifier types is chosen. Variables for that quantifier are then eliminated according to the corresponding quantifier semantics. This leads to multiple copies of the matrix, where the remaining variables are indexed by assignments over the eliminated variables occurring to their left in the quantifier prefix. The obtained QBF is equivalent to the original one, and only contains quantifiers of the same kind. The problem then can be reduced to either propositional satisfiability or validity, depending on the remaining quantifier.

[1]https://satcompetition.github.io/2024/output.html

Consider a QBF of the form $\Phi = \Pi.\forall a \exists x_1 \ldots \exists x_n.\phi$, where a is the innermost universal variable. It can be shown [3] that Φ is false if and only if the QBF

$$\Pi.\exists x_1^a x_1^{\bar{a}} \ldots x_n^a x_n^{\bar{a}}.\ \phi[a/\top, x_1/x_1^a, \ldots, x_n/x_n^a] \wedge \phi[a/\bot, x_1/x_1^{\bar{a}}, \ldots, x_n/x_n^{\bar{a}}]$$

is false, where x_i^a, $x_i^{\bar{a}}$ are fresh variables for $i \leq n$. This process can be recursively applied until all universal variables have been eliminated, at the cost of exponentially increasing the QBF size. Eventually, we obtain a QBF equivalent to the input QBF whose prefix only contains existential quantifiers; we call this formula the *full expansion* of Φ, denoted by $\Phi^{\exp}$. This QBF is false if and only if its propositional matrix is unsatisfiable, which can be checked using a SAT solver. As an example, the full expansion of the QBF from (1) is $\Phi^{\exp} = \exists x \exists y^a \exists y^{\bar{a}}.\phi[a/\top, y/y^a] \wedge \phi[a/\bot, y/y^{\bar{a}}]$, whose matrix is the unsatisfiable CNF formula:

$$(\neg x \vee y^a) \wedge (\neg y^a) \wedge (x \vee y^{\bar{a}}) \wedge (\neg y^{\bar{a}}) \qquad (2)$$

Note that, when a universal variable a is eliminated from the prefix Π, an existential variable x is only duplicated into $x^a, x^{\bar{a}}$ if $a <_\Pi x$. For an existential variable x, we need to consider the n universal variables a_i with $a_i \leq_\Pi x$; let us call them $a_1 <_\Pi \ldots <_\Pi a_n <_\Pi x$. The copies of x brought about by full expansion are all of the form $x^{l_1 \cdots l_n}$, where $\mathrm{Var}(l_i) = a_i$ for $i \leq n$; we call variables $x^{l_1 \cdots l_n}$ *expansion variables*, since they are the only variables occurring in the full expansion $\Phi^{\exp}$.

For a universal assignment $\tau\colon \mathrm{UVars}(\Pi) \to \{\top, \bot\}$, we also introduce the notation $x^{[\tau]}$ to denote the variable $x^{l_1 \cdots l_m}$, where $l_i = a_i$ if $\tau(a_i) = \top$ and $l_i = \overline{a_i}$ if $\tau(a_i) = \bot$. With this notation, we can write

$$C^{[\tau]} = \{l^{[\tau]} \mid l \in C, \mathrm{Var}(l) \in \mathrm{EVars}(\Pi)\}$$

where $C \in \phi$ and $\tau(C) \neq \top$. For the first clause $C = (x \vee a \vee y)$ of our example formula (1) we get $C^{[\tau]} = (x \vee y^{\bar{a}})$ for assignment $\tau = \{a \mapsto \bot\}$. For assignment $\sigma = \{a \mapsto \top\}$, the annotated clause $C^{[\sigma]}$ is undefined, because C evaluates to true under σ. We extend these annotations to formulas and get:

$$\Phi^{\exp} = \{C^{[\tau]} \mid C \in \phi,\ \tau\colon \mathrm{UVars}(\Pi) \to \{\top, \bot\} \text{ and } \tau(C) \neq \top\}$$

Modern expansion-based solvers implement abstraction refinement approaches to build partial expansions that are sufficient to decide the given formula [7, 18]. These techniques try to find only those assignments which are needed to derive a contradition, i.e., they try to avoid the construction of all and every $C^{[\tau]}$ clauses. In order to characterize expansion-based solving, the ∀-Exp+Res proof system has been introduced. A refutation of a false QBF $\Pi.\phi$ in the ∀-Exp+Res calculus [19] uses two rules:

- The *axiom rule* introduces a clause from the full expansion. For any clause $C \in \phi$ and any partial assignment $\tau\colon \mathrm{UVars}(\Pi) \to \{\top, \bot\}$ with $\tau(C) \neq \top$, the folowing rule can be applied:

$$\frac{}{C^{[\tau]}}\ \mathrm{Ax}(\tau)$$

- The well-known *resolution rule* of propositional logic:

$$\frac{C_1 \vee x \qquad C_2 \vee \neg x}{C_1 \vee C_2}\ \textsc{Res}(x)$$

As an example, the formula shown in (1) can be refuted by the following proof

$$\textsc{Res}(y^{\bar a})\,\dfrac{\textsc{Res}(y^a)\,\dfrac{\textsc{Res}(x)\,\dfrac{\text{Ax}(\tau)\,\dfrac{}{x \lor y^{\bar a}} \qquad \dfrac{}{\neg x \lor y^a}\,\text{Ax}(\sigma)}{y^{\bar a} \lor y^a} \qquad \dfrac{}{\neg y^a}\,\text{Ax}(\sigma)}{y^{\bar a}} \qquad \dfrac{}{\neg y^{\bar a}}\,\text{Ax}(\tau)}{\bot} \tag{3}$$

where $\tau = \{a \mapsto \bot\}$ and $\sigma = \{a \mapsto \top\}$. Note that the clauses derived by Ax rules are the clauses in the full expansion in (2).

These two rules play very different roles. The axiom rule captures the expansion of the input QBF into a propositional CNF formula; the resolution rule captures the refutation of the latter. It follows that the resolution rule can be replaced by any other propositional proof system able to refute clause sets. In this work, we suggest to replace the resolution part by the more powerful RAT + deletion (DRAT) proof system as it is used in state-of-the-art SAT solvers.

3 FERAT Proof System

We propose the new proof system ∀-Exp+RAT which modifies the ∀-Exp+Res proof system [19] by exchanging the resolution rule by the more powerful DRAT proof system [20, 25]. This idea follows an insight from proof complexity [8] which—to the best of our knowledge—has not been practically realized and evaluated. For example let us write the ∀-Exp+Res proof (3) as a proof sequence:

$$x \lor y^{\bar a},\ \neg x \lor y^a,\ \neg y^a,\ y^{\bar a},\ y^{\bar a} \lor y^a,\ y^{\bar a},\ \neg y^{\bar a},\ \bot$$

There, the last 4 clauses represent the resolution part of the proof. We can replace these by a DRAT proof, which only needs one clause:

$$x \lor y^{\bar a},\ \neg x \lor y^a,\ \neg y^a,\ y^{\bar a},\ \bot$$

DRAT belongs to a class of proof systems called *interference-based* proof systems [16]. Unlike resolution, deriving a clause in an interference-based proof not only depends on two clauses that have previously been derived, but also on those that have not. To add a clause C to the proof, it has to fullfill certain redundancy properties w.r.t. the current status of the proof. If another clause D is derived before, it could be that these properties are violated for C and C cannot be derived. However, it could be fine to derive C first, and then D. Consequently, interference-based proofs cannot be expressed as traditional proof trees [21]. The ∀-Exp+RAT proof system simply exchanges the propositional resolution part of a ∀-Exp+Res proof with a propositional DRAT proof. For the technical realization, we introduce the FERAT proof system which consists of three parts:

(1) The first part, called the *expansion mapping*, defines the correspondence between propositional variables $x^{[\tau]}$ in the expanded formula, the existential variables x in the input QBF, and the universal variable assignments $\tau\colon \text{UVars}(\Pi) \to \{\top, \bot\}$. This part is not strictly relevant for reasoning; rather, it is an artifact caused by the use of integers to encode literals in the `DIMACS` and `QDIMACS` formats.[2]
(2) The second part, called the *expansion proof*, roughly corresponds to the axiom rule in ∀-Exp+Res. The proof derives some of the propositional clauses contained in the full expansion of the input QBF, using the variable correspondence specified by the expansion mapping.
(3) The third part is the *clausal proof*, which contains a DRAT refutation of the propositional clauses introduced in the expansion proof.

In the usual fashion for modern proof checking in SAT, the proofs are generated in that sequence, and then checked in reverse order, which allows for shorter checking runtime [15].

3.1 Expansion Mapping and Proof

As mentioned earlier, the usual PCNF and CNF file formats do not allow for arbitrary variable names, resorting instead to integers to encode literals. In our setting, the variables in Φ^{exp} have the form $x^{l_1 \cdots l_n}$, where x is an existential variable and $\text{Var}(l_1), \ldots, \text{Var}(l_n)$ are the universal variables preceding x in Π. Due to variable name restrictions, though, we need a mechanism to make the proof checker aware of this correspondence.

This can be modeled as having a propositional CNF formula (the one actually being refuted by the clausal proof) which we call F in the following; a correct proof expects F to be a subset of Φ^{exp} modulo renaming of variables. The proof reports this correspondence by providing an expansion map ε that maps each propositional variable $y \in \text{Vars}(F)$ to an expansion variable $x^{l_1 \cdots l_n} \in \text{Vars}(\Phi^{\text{exp}})$. To validate such a mapping, we must check that: *i)* $x \in \text{EVars}(\Pi)$, and *ii)* $\text{Var}(l_i) \in \text{UVars}(\Pi)$ and $\text{Var}(l_i) <_\Pi x$ for all $i \leq n$, and *iii)* $\text{Var}(l_1), \ldots, \text{Var}(l_n)$ are pairwise distinct and cover all universal variables that are before x in prefix Π.

The proof also contains the expansion proof part, which is simply a list of the clauses in the CNF formula F. To validate the expansion proof, each clause $C \in F$ must be matched to a clause $D \in \phi$ and a universal assignment $\tau\colon \text{UVars}(\Pi) \to \{\top, \bot\}$ such that $D^{[\tau]} = \varepsilon(C)$. In practice, the existence of such a τ is infered during proof checking. For clauses $C \in F$ and $D \in \phi$, we can compute $\varepsilon(C) = x_1^{T_1} \lor \cdots \lor x_n^{T_n}$, where the T_n treat superscript literals as a set. Then, a τ with $D^{[\tau]} = \varepsilon(C)$ exists if: *i)* $T_1 \cup \cdots \cup T_n$ is complement-free, and *ii)* D is disjoint from $(T_1 \cup \cdots \cup T_n)$, and *iii)* $\{x_1, \ldots, x_n\} = \{x \in D \mid \text{Var}(x) \in \text{EVars}(\Pi)\}$.

As an example, the expanded formula shown in Eq. (2) might be passed to the SAT solver as the CNF formula

$$(\neg x' \lor y') \land (\neg y') \land (x' \lor z') \land (\neg z') \tag{4}$$

where the expansion mapping contains $\varepsilon(x') = x$, $\varepsilon(y') = y^a$, and $\varepsilon(z') = y^{\bar a}$.

3.2 Clausal Proof

The clausal proof shows the unsatisfiability of the propositional CNF formula F resulting from expansion (modulo renaming through ε). This proof is exactly the DRAT proof generated by the SAT solver. Due to the aforementioned characteristics of interference-based proofs, clausal proofs are incremental: they consist of a list of *instructions* that iteratively introduce or delete a clause starting from CNF F.

State-of-the-art proofs for CDCL-based solvers strongly rely on a technique called *reverse unit propagation (RUP)* [13]. A clause C is

[2]See `https://satcompetition.github.io` for a description of DIMACS and `https://qbflib.org` for a description of QDIMACS

a RUP clause over F whenever $F \wedge \bigwedge_{l \in C} \neg l$ leads to a conflict via unit propagation (also known as Boolean constraint propagation).

The DRAT proof system includes three instructions, which each modify the *current formula* F' as follows:

- *Deletions* arbitrarily remove any clause C from F'.
- *RUP introductions* insert a clause C in the formula, under the condition that C is a RUP clause over F'. The resulting formula is equivalent to F'.
- *RAT introductions* insert a clause C in the formula, under the condition that there is a literal $l \in C$ such that $C \vee D \setminus \{\bar{l}\}$ is a RUP clause over F' for all clauses $D \in F'$. The resulting formula is not, in general, equivalent to F'; however, it is guaranteed to be satisfiable if and only if F' is.

A DRAT proof is correct if the conditions on each instruction hold, and the empty clause is contained in the accumulated formula at the end of the proof.

Take for example the formula ϕ given by 4. In this case, the clause $\perp$ is a RUP clause in ϕ: unit propagation on $\phi \wedge \bar{\perp}$ propagates literals $\neg y', \neg z', x', \neg x'$, which results in a conflict.

The inclusion of deletions in the proof system is due to both performace and semantic reasons. On the one hand, since the DAG-like dependencies of resolution proofs are not available in DRAT, deletions are needed to curb the growth of the accumulated formula [15]. On the other hand, the absence of clauses can enable new RAT introductions, so not including them can turn correct proofs into incorrect ones and vice versa [1]. In our framework, validation is deferred to an off-the-shelf DRAT checker, so we do not discuss it in detail; we refer the interested reader to [1, 15, 25].

3.3 File Format

In order to express ∀-Exp+RAT proofs, we contribute the FERAT file format. Figure 1 shows its extended Backus-Naur form (EBNF) grammar, disregarding comments (which are the same as in DIMACS) and white space.

Proof lines starting with "x" contain an expansion mapping from propositional variables to their corresponding QBF expansion variables, along with a list of literals which represent their partial universal assignment. These correspond to the mapping ε defined in Section 3.1.

For example, consider the QBF prefix $\exists x \forall y \exists z u \forall v \exists w$, and assume the partial expansion uses propositional expansion variables x, $z^{\bar{y}}$, $u^{\bar{y}}$, z^{y} and $w^{\bar{y},v}$. Let us presume that we choose the following

```
    ⟨FERAT⟩ → { ⟨VarMap⟩ } [⟨ClauseMap⟩]
              { ⟨Exp⟩ } { ⟨Intro⟩ | ⟨Del⟩ }
   ⟨VarMap⟩ → "x" { ⟨Pos⟩ } "0" { ⟨Pos⟩ } "0" { ⟨Lit⟩ } "0" NL
⟨ClauseMap⟩ → "o" { ⟨Pos⟩ } "0" NL
      ⟨Exp⟩ → "e" { ⟨Lit⟩ } "0" NL
    ⟨Intro⟩ → { ⟨Lit⟩ } "0" NL
      ⟨Del⟩ → "d" { ⟨Lit⟩ } "0" NL
      ⟨Lit⟩ → ["-"] ⟨Pos⟩
```

Figure 1: EBNF grammar of the FERAT proof file format, where ⟨Pos⟩ is a positive integer value.

```
     QDIMACS:               FERAT:
p cnf 3 3              x 1 0 1 0 0
e 1 0                  x 2 0 3 0 2 0
a 2 0                  x 3 0 3 0 -2 0
e 3 0                  o 1 2 3 3 0
1 2 3 0                e 1 3 0
-1 -2 3 0              e -1 2 0
-3 0                   e -3 0
                       e -2 0
                       0
```

Figure 2: Example of a FERAT proof of Φ, along with Φ in QDIMACS format.

QDIMACS encoding for the QBF variables:

$$1 = x \quad 2 = y \quad 3 = z \quad 4 = u \quad 5 = v \quad 6 = w$$

We also choose the DIMACS encoding for the propositional expansion variables given by the following ε:

$$\varepsilon(1) = x \quad \varepsilon(2) = z^{\bar{y}} \quad \varepsilon(3) = u^{\bar{y}} \quad \varepsilon(4) = z^{y} \quad \varepsilon(5) = w^{\bar{y},v}$$

Note that the DIMACS variables above and below are conceptually different: the variables above encode the variables from the QBF, whereas the ones below encode the propositional variables *in the partial expansion.* This mapping ε would be expressed by the following proof lines:

```
x 1 0 1 0 0
x 2 3 0 3 4 0 -2 0
x 4 0 3 0 2 0
x 5 0 6 0 -2 5 0
```

Here, for example, the second line means that the propositional expansion DIMACS variables 2 and 3 correspond to the expansion variables $3^{\bar{2}}$ and $4^{\bar{2}}$ respectively, which in turn are $z^{\bar{y}}$ and $u^{\bar{y}}$; the fourth line means that 5 corresponds to $6^{\bar{2},5}$, which is $w^{\bar{y},v}$. Observe that only mappings which share the same partial universal assignment can be put in the same mapping line [14].

Lines starting with "e" contain an propositional expansion clause, i.e. a clause from F. To curb the burden of search in matching propositional expansion clauses and QBF clauses proposed in subsection 3.1, a line prefixed with "o" can optionally be provided. These lines contain a list with as many entries as propositional expansion clauses in the proof. Thus, the line `o 1 3 3 0` specifies that the first "e"-prefixed clause was expanded from the first clause in the QDIMACS input QBF formula, and the second and third propositional clauses were expanded from the third QBF clause. If an "o" line is not supplied, FERAT-TOOLS is still able to perform the check, albeit with a performance penalty.

The remaining lines, either unprefixed or prefixed by "d", constitute the DRAT proof refuting the clause set specified by the "e"-prefixed lines.

Figure 2 shows our running example expressed as a QDIMACS and a FERAT file. Note that the DRAT proof itself spans only the final 0 line, showcasing the greater expressiveness of DRAT compared to resolution in this example.

4 Implementation

Our framework combines multiple executables (QBF solver, SAT solvers, DRAT checkers) as well as our own utility to check ∀-Exp steps (FERAT-TOOLS), and a PYTHON pipeline, which handles connections between these various binaries[3]. These tools can also be exchanged easily, e.g. to support linear RAT (LRAT) proofs using CADICAL [4] and LRAT-TRIM [10] as SAT solver and checker. Two modes are accessible in the interface: generation, and checking. Figure 3 gives an overview of our framework as (simplified) flow diagram.

4.1 Solving and Proof Generation

The FERAT framework solving pipeline proceeds as follows:

(1) The input QBF, expressed in the `QDIMACS` format, is solved using an expansion-based QBF solver; in our case, we use IJTIHAD + LINGELING presented in [14]. An *expansion trace* is the output; this trace includes the expansion CNF formula as well as the mapping ε. This is expressed as a `DIMACS` file, with ε encoded within comments, similarly to [14].
(2) The expansion trace is passed to a SAT solver to generate a DRAT proof; this step also verifies that the expansion is unsatisfiable. Any SAT solver that supports the necessary proof generation can be used here. In the experiments presented in the next section, we evaluate the performance of different SAT solvers.
(3) The validity of the DRAT proof is then ascertained with a proof checker; our pipeline uses DRAT-TRIM with some additional instrumentation. The checker outputs a trimmed proof, where redundant proof instructions and formula clauses have been optimized away [15]. The trimmed proof becomes the clausal proof in the final FERAT proof, whereas the unsatisfiable core output by the proof checker becomes the expansion proof.
(4) The unsatisfiable core is then validated against the expansion mapping ε from step (1) and the input QBF using a ∀-Exp checker. For this, we developed FERAT-TOOLS as a checker.
(5) The expansion mapping ε, the unsatisfiable core and the trimmed proof are finally combined into a FERAT proof.

4.2 Proof Validation

While checking already happens during proof generation, this does not enable independent validation of the proof. Our FERAT framework also allows for this, albeit with a slight runtime and memory consumption penalty.

First, the expansion mapping and proof (containing "x", "o" and "e" lines) and the DRAT proof ("d" and unprefixed lines) are separated. On the one hand, the DRAT proof is validated using "e"-prefixed lines as the input formula using a proof checker (in our case, an instrumented version of DRAT-TRIM). On the other hand, the expansion mapping and proof are validated against the input QBF using a ∀-Exp+RAT checker (our FERAT-TOOLS tool, in our pipeline).

[3]Our tools and experimental data are publicly available in `https://github.com/MarcelSimader/FERAT`.

Solver	Solved	TO	OOM
CaDiCaL	44	6	8
Glucose	43	9	6
Kissat	49	6	3
Lingeling	46	8	4
Lingeling-14	41	13	4
PicoSAT	36	18	4

Table 1: DRAT proof generation SAT solver backend results, including timeouts (TO) and out-of-memory (OOM).

5 Experimental Evaluation

We evaluated the FERAT pipeline with the goal of answering two questions. First, we wanted to identify any particular SAT solver that performed best as a SAT backend to the proof generation process. Second, we wanted to compare our contributed FERAT framework to the existing FERP frameworks in terms of runtime and proof size. We experimentally evaluated these questions. For our experiments, we used a timeout of 900 seconds and memory limited to 8 GiB over AMD EPYC 7313 CPUs.

5.1 Benchmarks

To evaluate our tools, we ran the instrumented version of IJTIHAD over PCNF instances in the `QDIMACS` format, discarding true formulas and unterminated instances. Our benchmarks then are the output `DIMACS` files containing comments that encode the expansion mappings. At this stage, we used LINGELING[4] as a SAT backend for IJTIHAD[5], since our preliminary experiments showed the best runtime over false instances. Other IJTIHAD SAT backends we tested include CADICAL[6], PICOSAT[7], and GLUCOSE[8]. We excluded the solver KISSAT from benchmark generation, since Ijtihad does not natively support its API.

For benchmark generation, we used a timeout of 64 800 seconds and memory limited to 20 GiB over AMD EPYC 7313 CPUs. IJTIHAD was run over the 377 instances from the QBFGallery 2023[9]. This produced 58 successfully expanded false formulas, which constituted our evaluation benchmarks; the rest of the 377 instances either were reported as true, timed out, or exceeded the memory limit.

5.2 SAT Solver Backends

We ran different SAT solvers over our benchmarks, which would correspond to DRAT proof generation in the FERAT pipeline. The solvers we tested include: CADICAL[10], GLUCOSE[11], KISSAT[12], LINGELING[4], LINGELING-14[13], and PICOSAT[7].

[4]Version as submitted to SAT'16, `https://fmv.jku.at/lingeling`
[5]`https://extgit.iaik.tugraz.at/scos/ijtihad`
[6]Version 1.8.0, `https://github.com/arminbiere/cadical/tree/rel-1.8.0`
[7]Version v965, `https://fmv.jku.at/picosat`
[8]Version 4.0, `https://github.com/audemard/glucose/tree/4.0`
[9]`https://qbf23.pages.sai.jku.at/gallery`
[10]Version 1.9.5, `https://github.com/arminbiere/cadical/tree/rel-1.9.5`
[11]Version 4.2.1, `https://github.com/audemard/glucose/tree/4.2.1`
[12]Version 3.1.1, `https://github.com/arminbiere/kissat/tree/rel-3.1.1`
[13]Version as submitted to SAT'14, `https://fmv.jku.at/lingeling`

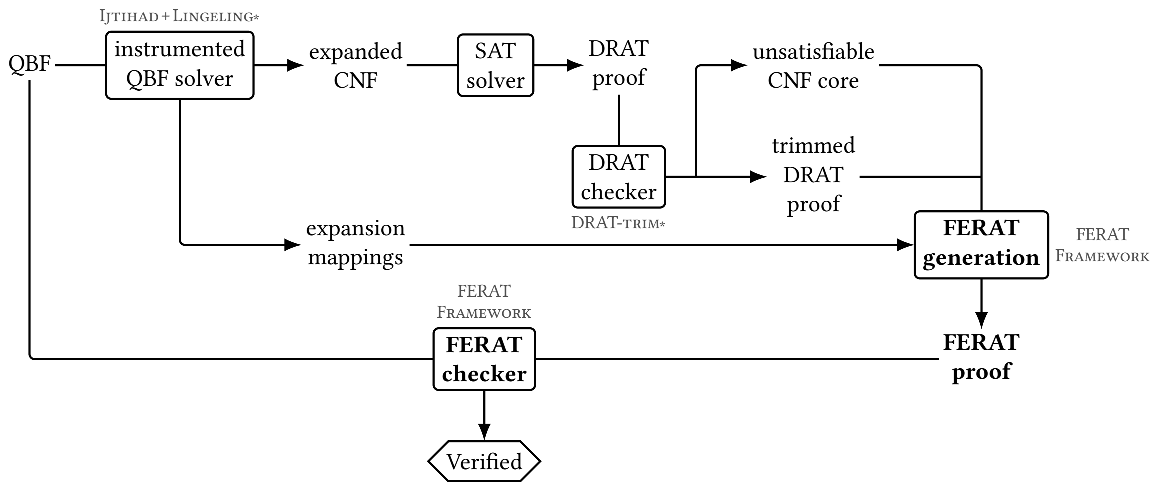


Figure 3: The FERAT pipeline. The specific tools used are shown next to boxed procedures, with an asterisk if they were modified for certification purposes. Our contribution includes the FERAT generation framework, the FERAT proof format and the FERAT checker. SAT solvers are exchangable.

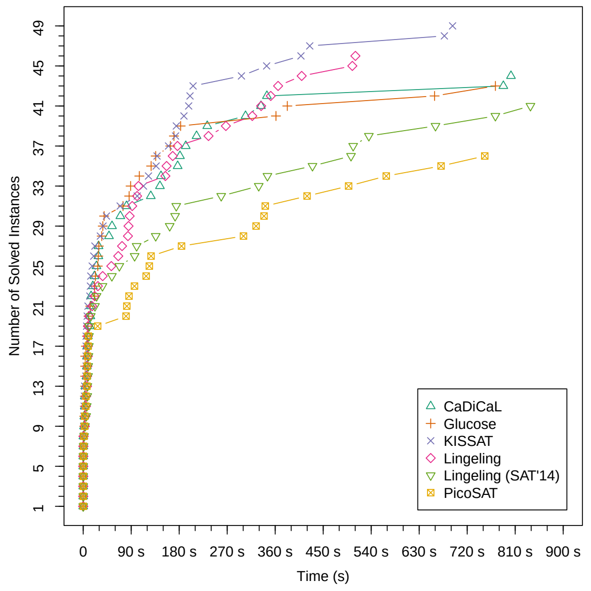


Figure 4: Runtimes of SAT solver backends for DRAT proof generation.

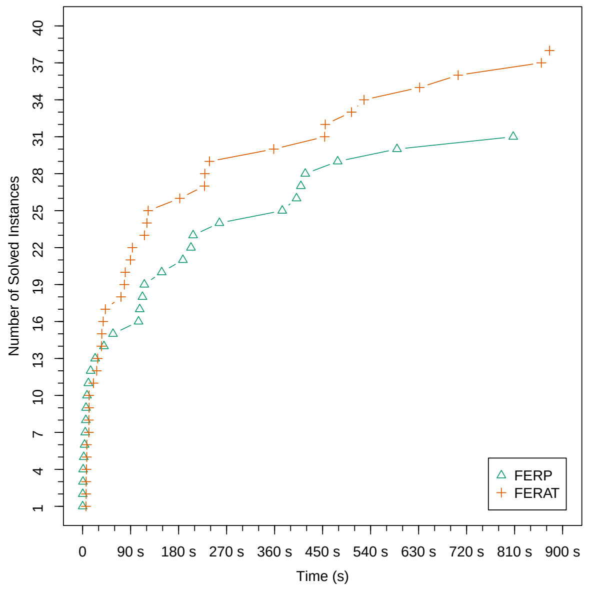


Figure 5: Runtimes of the FERAT and FERP pipelines.

As expected, every instance that was solved returned an UNSAT result. Figure 4 shows the runtimes for each solver. Overall, SAT solver KISSAT provided the best performance. It is also worth noting that the only solver producing resolution proofs, PICOSAT, is also the worst performer.

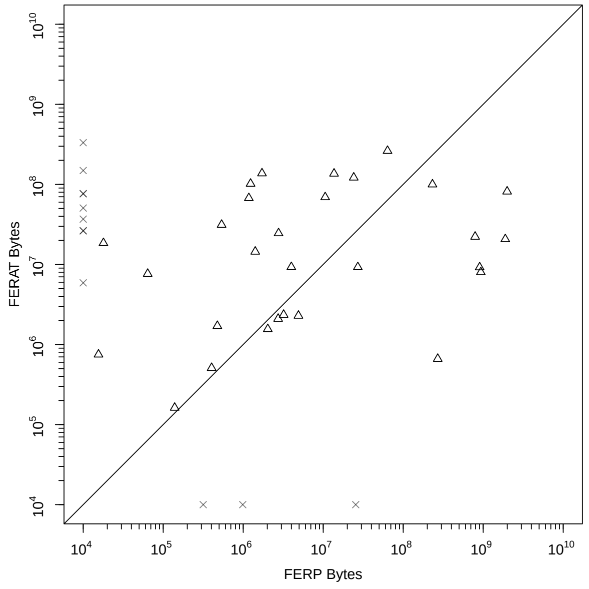
This has immediate impact on the practicality of the FERAT pipeline over the resolution-based FERP pipeline [14]: expressing the propositional refutation as a DRAT proof, as opposed to resolution proofs, enables the use of state-of-the-art SAT solvers, as opposed to relatively outdated resolution-producing solvers.

5.3 FERAT compared to FERP

In this experiment, we compared the performance of the FERAT pipeline compared to FERP on runtime and proof size. We excluded the solving process through the instrumented version of IJTIHAD from the evaluation, since this is expected to be similar in both pipelines and is not relevant for evaluating proof checking. To do so, FERP needed to be modified to allow skipping this first stage, whereas FERAT already has this feature built in.

The runtime results of this experiment are shown in Figure 5. FERAT was able to certify 38 instances out of the 58 benchmarks, while FERP was able to certify 31. For instances taking longer than 30 seconds to run, FERAT is invariably faster. Over the 28 instances validated by both pipelines, the median runtime for FERAT was 37.255 seconds, whereas FERP stood at 80.9 seconds.

We also examined the sizes of the produced FERAT and FERP certificates. Figure 6 shows scatter plots of proof size in terms of file size and number of proof instructions. For both counts, we removed comment lines.



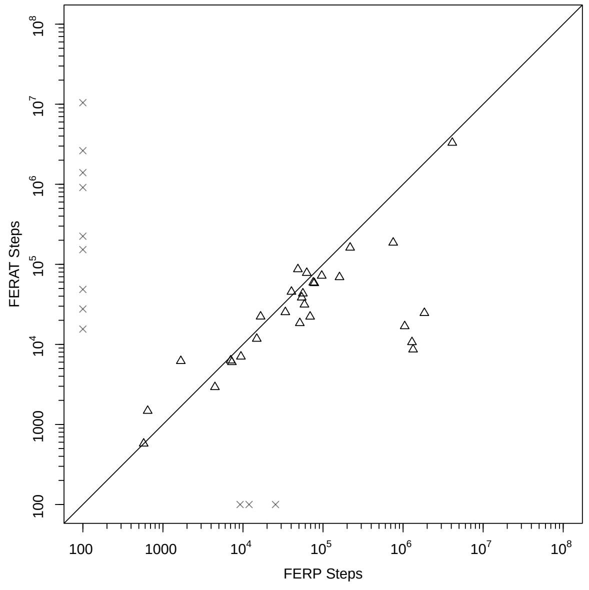


Figure 6: Sizes of proofs generated by the FERAT and FERP pipelines, measured in file size (above) and number of proof lines (below). Instances certified only by one of the pipelines are marked in the low-end of the respective axis.

FERAT and FERP produced certificates with a total size of approximately 1.178 GiB and 6.682 GiB respectively, which corresponds to 4 423 341 and 11 546 930 steps. This makes FERAT proof files c.a. 6.6 times smaller than FERP proofs, containing about 2.6 times fewer proof lines. Instances certified by only one of the pipelines are shown as crosses in the inner margin of the plot, and were not counted towards our comparison.

5.4 Results Evaluation

The data in Sections 5.2 and 5.3 suggests that the FERAT pipeline is both more performant, and produces smaller certificates, than FERP. We conjecture these results are due to two factors. First, since FERAT is based on DRAT proofs, it is compatible with modern SAT solving technology, whereas FERP's reliance on resolution proofs forced it to use outdated SAT solvers. Second, DRAT proofs are generally more compact than resolution proofs.

While it is true that in a best-case analysis an exponential separation exists between DRAT and resolution [20], we consider it unlikely that this is the source of the observed reduction in proof size. To the best of our knowledge, all the solvers we tested generate DRAT proofs containing only deletions and RUP introductions. Kissat only gained the ability to produce proofs containing RAT introductions very recently, in a later version to the one used for our experiments[14]. Rather, the RUP introduction rule allows for a (polynomial) reduction in proof size by condensing multiple resolution steps into a single RUP inference [12].

6 Conclusion

We presented a novel proof system called ∀-Exp+RAT that allows to employ recent SAT solvers for expansion-based QBF solving. We implemented a full certification tool chain in the FERAT framework which we carefully evaluated. The framework is designed in a very flexible manner such that it can be easily adapted to any future propositional proof checking tool chains. In future, we plan to investigate how to integrate solvers based on different solving paradigms and different proof systems into our framework such that it becomes possible to certify the results of portfolio QBF solvers.

References

[1] Johannes Altmanninger and Adrian Rebola-Pardo. 2020. Frying the egg, roasting the chicken: unit deletions in DRAT proofs. In *CPP*. ACM, 61–70.

[2] Olaf Beyersdorff, Mikolás Janota, Florian Lonsing, and Martina Seidl. 2021. Quantified Boolean Formulas. In *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications, Vol. 336. IOS Press, 1177–1221.

[3] Armin Biere. 2004. Resolve and Expand. In *Proc. of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT).* `http://www.satisfiability.org/SAT04/programme/93.pdf`

[4] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. 2024. CaDiCaL 2.0. In *CAV (1) (Lecture Notes in Computer Science, Vol. 14681)*. Springer, 133–152.

[5] Armin Biere, Mathias Fleury, Nils Froleyks, and Marijn J. H. Heule. 2023. The SAT Museum. In *POS@SAT (CEUR Workshop Proceedings, Vol. 3545)*. CEUR-WS.org, 72–87.

[6] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). 2021. *Handbook of Satisfiability - Second Edition.* Frontiers in Artificial Intelligence and Applications, Vol. 336. IOS Press.

[7] Roderick Bloem, Nicolas Braud-Santoni, Vedad Hadzic, Uwe Egly, Florian Lonsing, and Martina Seidl. 2018. Expansion-Based QBF Solving Without Recursion. In *FMCAD*. IEEE, 1–10.

[14] Commit `1e56df6c9bd2e43b8d313df462fb792b0a5eb891`

[8] Leroy Chew and Judith Clymo. 2020. How QBF Expansion Makes Strategy Extraction Hard. In *Proc. of the 10th International Joint Conference on Automated Reasoning (IJCAR) (Lecture Notes in Computer Science, Vol. 12166)*, Nicolas Peltier and Viorica Sofronie-Stokkermans (Eds.). Springer, 66–82. `https://doi.org/10.1007/978-3-030-51074-9_5`

[9] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM* 50, 5 (2003), 752–794. `https://doi.org/10.1145/876638.876643`

[10] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. 2017. Efficient Certified RAT Verification. In *CADE (Lecture Notes in Computer Science, Vol. 10395)*. Springer, 220–236.

[11] Mathias Fleury and Christoph Weidenbach. 2020. A Verified SAT Solver Framework including Optimization and Partial Valuations. In *LPAR (EPiC Series in Computing, Vol. 73)*. EasyChair, 212–229.

[12] Allen Van Gelder. 2012. Producing and verifying extremely large propositional refutations - Have your cake and eat it too. *Ann. Math. Artif. Intell.* 65, 4 (2012), 329–372.

[13] Evguenii I. Goldberg and Yakov Novikov. 2003. Verification of Proofs of Unsatisfiability for CNF Formulas. In *DATE*. IEEE Computer Society, 10886–10891.

[14] Vedad Hadzic, Roderick Bloem, Ankit Shukla, and Martina Seidl. 2022. FERPModels: A Certification Framework for Expansion-Based QBF Solving. In *SYNASC*. IEEE, 80–83.

[15] Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. 2013. Trimming while checking clausal proofs. In *FMCAD*. IEEE, 181–188.

[16] Marijn Heule and Benjamin Kiesl. 2017. The Potential of Interference-Based Proof Systems. In *ARCADE@CADE (EPiC Series in Computing, Vol. 51)*. EasyChair, 51–54.

[17] Mikoláš Janota, William Klieber, Joao Marques-Silva, and Edmund Clarke. 2016. Solving QBF with counterexample guided refinement. *Artificial Intelligence* 234 (2016), 1–25. `https://doi.org/10.1016/j.artint.2016.01.004`

[18] Mikolás Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. 2016. Solving QBF with counterexample guided refinement. *Artif. Intell.* 234 (2016), 1–25. `https://doi.org/10.1016/J.ARTINT.2016.01.004`

[19] Mikolás Janota and João Marques-Silva. 2015. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.* 577 (2015), 25–42.

[20] Benjamin Kiesl, Adrián Rebola-Pardo, and Marijn J. H. Heule. 2018. Extended Resolution Simulates DRAT. In *IJCAR (Lecture Notes in Computer Science, Vol. 10900)*. Springer, 516–531.

[21] Adrián Rebola-Pardo and Martin Suda. 2018. A Theory of Satisfiability-Preserving Proofs in SAT Solving. In *LPAR (EPiC Series in Computing, Vol. 57)*. EasyChair, 583–603.

[22] Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. 2019. A Survey on Applications of Quantified Boolean Formulas. In *Proc. of the 31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 78–84. `https://doi.org/10.1109/ICTAI.2019.00020`

[23] João P. Marques Silva and Karem A. Sakallah. 1999. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Trans. Computers* 48, 5 (1999), 506–521. `https://doi.org/10.1109/12.769433`

[24] Sarek Høverstad Skotåm. 2022. *CreuSAT, Using Rust and Creusot to create the world's fastest deductively verified SAT solver.* Master's thesis. University of Oslo. `https://www.duo.uio.no/handle/10852/96757`

[25] Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. 2014. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In *SAT (Lecture Notes in Computer Science, Vol. 8561)*. Springer, 422–429.