Extended Resolution as Certificates for Propositional Logic

Chantal Keller

LIX and INRIA Saclay–Île-de-France at École Polytechnique, Palaiseau, France
Chantal.Keller@inria.fr

Abstract

When checking answers coming from automatic provers, or when skeptically integrating them into proof assistants, a major problem is the wide variety of formats of certificates, which forces to write lots of different checkers. In this paper, we propose to use the extended resolution as a common format for every propositional prover. To be able to do this, we detail two algorithms transforming proofs computed respectively by tableaux provers and provers based on BDDs into this format. Since this latter is already implemented for SAT solvers, it is now possible for the three most common propositional provers to share the same certificates.

1 Introduction

Different theorem provers can communicate to benefit from each other capabilities. It is the case for instance when *automatic theorem provers*, which can prove efficiently even hard problems, cooperate with *interactive theorem provers*, well known for their trustworthiness. The powerful Isabelle [23] tactic `sledgehammer` [22] implements such an interaction, by calling in parallel various kinds of automatic provers to solve Isabelle goals.

To take advantage of efficiency without compromising soundness, the cooperation must be *skeptical*: in addition to a yes/no answer, the automatic prover must return a *proof witness* that can be checked or reconstructed in the proof assistant. In the `sledgehammer` tactic, this is for instance the case for the SMT solver Z3 whose proof witnesses are reconstructed to produce Isabelle theorems [4].

In addition to the fact that most automatic provers do not give (detailed enough) proof witnesses and thus must be taken at face value, the ones that do provide such witnesses all implement their own formats to prove the validity or the unsatisfiability of a given formula. It thus requires much effort to write a checker for a new prover, even when some already exist for other tools.

Besson *et al.* [3] proposed a format for the particular case of SMT solvers which was argued to be both easy to generate and easy to check. This affirmation was actually backed up: the competitive SMT solver veriT [5] is able to return a variant of these proof witnesses at small cost, which can then be efficiently checked in Coq [1, 2]. The propositional part of this format is based on extended resolution [28], into which theory reasoning can be plugged.

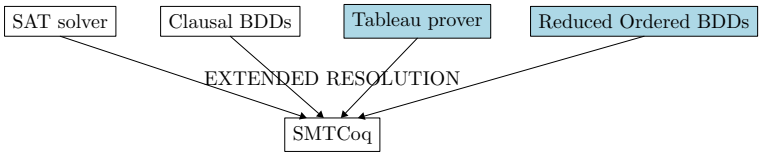
The aim of this work is to promote extended resolution as a common format for certificates about propositional logic, based on several observations:

- on a theoretical point of view, extended resolution is known to p-simulate most existing proof systems [29, 24];
- on a practical point of view, we can efficiently translate reasoning performed by some proofs systems in extended resolution (see eg. [20] for DPLL with backjumping and [26] for clausal BDDs) and efficiently check such certificates [1, 2];

- this format is easily extendable beyond propositional logic, like theory reasoning as already implemented for SMT solvers [3, 2] or quantifiers [10].

The work presented here extends this scheme with two other families of provers: the method of analytic tableaux [27, 9] and the reduced ordered binary decision diagrams [6] (BDDs in short). To do so, we detail two algorithms translating the proofs found by each of these provers into a proof in extended resolution which is polynomial in the length of the original proof.

The objective is to have a common proof format in order to share checkers and thus save a lot of human work. The work presented in this paper can be used to instrument already existing provers in order to return certificates in the concrete format of [3] – which corresponds to extended resolution – and thus directly plugged into checkers understanding it like SMTCoq [2]. As such, we could check with great confidence answers coming from these provers without having to write new code in an interactive theorem prover; this could then be extended into tactics in order to enjoy tableaux and BDDs automation inside Coq without compromising soundness.



Moreover, to add safe propositional automation into another interactive prover than Coq, one single checker would be sufficient.

Note that the goal of this paper is to give two new algorithms to generate certificates, but not to explain how to efficiently check them after: this has already been detailed in previous work [1, 2].

The paper is organized as follows. After explaining the extended resolution proofs (Section 2) and their already existing applications to SAT and SMT, we present in Section 3 the method of analytic tableaux and the algorithm to deduce a resolution certificate from a tableau proof. The same approach is applied to the BDD method in the following section (Section 4). We finally discuss related and future work in Section 5 before concluding.

2 Extended resolution

Extended resolution [28] is an extension of the well known resolution proof system [25] with the possibility to add new variables representing larger terms, giving more compact proofs than standard resolution. We first recall its definition and present our notations, before giving examples of applications.

2.1 Definitions

We are given a countable set of propositional variables $\mathcal{V}$. A *literal* l is a variable v (in which case it is called a *positive* literal) or its negation $\bar{v}$ (in which case it is called a *negative* literal). A *clause* C is a disjunction of literals, written $l_1 \vee \cdots \vee l_n$ when it is nonempty and $\square$ otherwise. A *conjunctive normal form* (CNF in short) $\mathcal{S}$ is a set of clauses seen as their conjunction.

A *valuation* $\rho : \mathcal{V} \to \{\top, \bot\}$ is a total function mapping variables to one of the values *true* or *false*. Given a valuation ρ, it is possible to define the interpretation of literals, clauses and CNFs (respectively written $|l|_\rho$, $|C|_\rho$ and $|\mathcal{S}|_\rho$) in the standard way. We say that a CNF $\mathcal{S}$ is *satisfiable* if there exist a valuation ρ such that $|\mathcal{S}|_\rho = \top$; otherwise, we say that $\mathcal{S}$ is *unsatisfiable*.

The *resolution rule* is a deduction rule that builds a new clause from two existing clauses:

$$\frac{v \vee C \qquad \bar{v} \vee D}{C \vee D}$$

where v does not appear in C nor D, and no variable appears with one polarity in C and the other in D. The variable v is called the *resolution variable*. A comb tree of resolutions is a *resolution chain*. This rule is *refutationally complete*: a CNF $\mathcal{S}$ is unsatisfiable if and only if the empty clause can be derived by applications of the resolution rule starting with the clauses of $\mathcal{S}$.

Extended resolution extends the resolution rule with additional rules without premisses that introduce new clauses containing fresh variables implicitly representing terms of propositional logic.

A typical use consists in folding and unfolding logical connectives: to express that a fresh variable x represents $f_1 \star \cdots \star f_n$ where $\star$ is a connective, the rules are the tautological clauses stating that $x \Leftrightarrow f_1 \star \cdots \star f_n$. Such rules are used for instance to transform a Boolean problem into an equisatisfiable one in CNF [28, 3].

In the remaining of this paper, we are going to use these rules that fold and unfold connectives for all the connectives. In this section we give the examples of the $\wedge$, $\Rightarrow$ and ite (if ... then ... else ...) connectives; the same method applies for all the others (one may refer to [28, 3] for more details).

Example 2.1. A fresh variable x can represent the conjunction $x_1 \wedge x_2$ by introducing the following three rules:

$$\overline{\bar{x} \vee x_1} \qquad \overline{\bar{x} \vee x_2} \qquad \overline{x \vee \bar{x}_1 \vee \bar{x}_2}$$

respectively stating that x implies x_1, x implies x_2, and x_1 and x_2 together imply x.

Similarly, a fresh variable y can represent the implication $y_1 \Rightarrow y_2$ by introducing the following three rules:

$$\overline{\bar{y} \vee \bar{y}_1 \vee y_2} \qquad \overline{y \vee y_1} \qquad \overline{y \vee \bar{y}_2}$$

and a fresh variable z can represent the branching $\mathrm{ite}(z_1, z_2, z_3)$ (stating "if z_1 then z_2 else z_3") by introducing the following four rules:

$$\overline{\bar{z} \vee z_1 \vee z_3} \qquad \overline{\bar{z} \vee \bar{z}_1 \vee z_2} \qquad \overline{z \vee z_1 \vee \bar{z}_3} \qquad \overline{z \vee \bar{z}_1 \vee \bar{z}_2}$$

2.2 Applications

Introduced to establish lower bounds on the minimal length of proofs, extended resolution was shown to bypass resolution since it has the same power as the Extended Frege Systems [8, 29], the most powerful known proof systems. It is also known to provide short proofs to problems hard for resolution like Haken's pigeon-hole formulae [7].

The clauses learned during conflict analysis performed by modern SAT solvers can be easily derived by a resolution tree [20], and thus state-of-the-art SAT solvers like zChaff [14] or MiniSat [12] are instrumented to return resolution proofs for unsatisfiable problems. The extension

allows to define certificates for more complex proof systems like clausal BDDs [26] or the Boolean part of SMT solvers [3] (which do not require their inputs to be in CNF). Even if such a proof witness can be rather huge, it takes a negligible cost to output it compared to finding that a formula is unsatisfiable. On the other side, it can be efficiently checked, for instance inside proof assistants [1, 2].

Extended resolution is thus a good candidate as a common proof format for propositional reasoning. It has already been widely studied and implemented for SAT solvers. In the remaining of this paper, we focus on two other popular propositional proof methods: the method of analytic tableaux and full BDDs and show that we can as well translate their reasoning into certificates based on extended resolution.

3 Certificates for the method of analytic tableaux

The method of analytic tableaux [27, 9] is a decision procedure for various kinds of logic. Applied to propositional logic, it can establish the unsatisfiability of any quantifier-free formula without requiring it to be in some normal form, contrary to SAT solvers. The popularity of this method comes from the fact that it can be extended to a large spectrum of standard features like quantifiers or modal logic. Its efficiency and simplicity make it largely used in applications requiring great confidence: it is for instance at the heart of the widely used `blast` tactic of the Isabelle proof assistant [21].

After presenting the method for propositional logic and theoretical results concerning its power, we explain how to deduce certificates based on extended resolution from tableaux proofs.

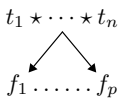
3.1 The method

A *refutation tableau* is a tree whose nodes are labeled with propositional formulas such that:

- a decomposition rule is applied at each node; and
- every branch from the root to a leaf contains at least a formula and its negation – in this case, we say that a branch is *closed*.

It is established that **a formula F is unsatisfiable if and only if there exists a refutation tableau of root F**. Tableaux thus give a complete method to establish the unsatisfiability of propositional formulas.

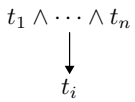
A *decomposition rule* splits a formula labeling a node above in the tree (not necessarily the current node) into one or more sub-formulas, depending on the head symbol. It can be generically described by the node:



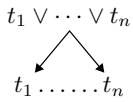
where $\star$ can be any connective, $p \leqslant n$, and any f_j can be either t_i or $\bar{t}_i$ for some i, depending on $\star$.

We give examples of these decomposition rules for the $\wedge$, $\vee$ and $\Rightarrow$ connectives.

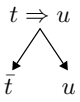
Example 3.1. A conjunction can be projected into any of its direct sub-terms:



An n-ary disjunction decomposes into n branches:

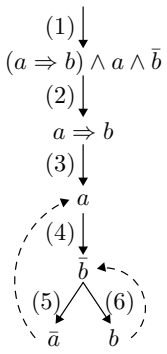


Similarly, an implication decomposes into 2 branches:



The following example proves the unsatisfiability of $(a \Rightarrow b) \wedge a \wedge \bar{b}$.

Example 3.2. A refutation tableau proving the unsatisfiability of $(a \Rightarrow b) \wedge a \wedge \bar{b}$ is:



The first extra-edge (1) simply states the initial formula as the root. Edges (2) to (4) decompose it as a conjunction. Edges (5) and (6) decomposes the implication $a \Rightarrow b$. Finally, the backwards dashed arrows illustrate the closure of each branch.

We theoretically know that resolution p-simulates analytic tableaux on CNF formulas (**Theorem 5.1** of [29]). The converse is not true: on some classes of problems, resolution can build exponentially smaller proofs than tableaux. To our knowledge, there is no link between extended resolution and full analytic tableaux.

3.2 From refutation tableaux to extended resolution

In this section, we describe a generic algorithm to transform any refutation tableau proving the unsatisfiability of f into a proof of the empty clause in extended resolution starting from f, **without requiring f to be in normal form** (contrary to [29]). It produces a proof tree whose number of nodes is **linear** in the number of nodes in the original tableau proof.

3.2.1 The algorithm on an example

To understand the idea of the algorithm, we first conduct it step by step on **Example 3.2**.

First, we assign fresh variables to each (non-strict) sub-formula of the initial formula which is not a literal. In our example, we thus add two fresh variables: $f \triangleq a \Rightarrow b$ and $g \triangleq f \wedge a \wedge \bar{b}$.

Second, we build a piece of a proof tree for each edge in the tableau in the following way:

(1) We initiate the process by stating that the fresh variable assigned to the initial formula holds: g.

(2) This step is the first projection of g, which can be derived from (1) in extended resolution:

$$\dfrac{\dfrac{}{\bar{g} \vee f} \qquad \dfrac{(1)}{g}}{f}$$

(3) - (4) Similarly, these steps are the second and third projections of g:

$$\dfrac{\dfrac{}{\bar{g} \vee a} \qquad \dfrac{(1)}{g}}{a} \qquad\qquad \dfrac{\dfrac{}{\bar{g} \vee \bar{b}} \qquad \dfrac{(1)}{g}}{\bar{b}}$$

(5) This step corresponds to decomposing f which has been obtained at step (2). This is the resolution of what has been obtained at (2) with the rule of extended resolution to decompose an implication:

$$\dfrac{\dfrac{}{\bar{f} \vee \bar{a} \vee b} \qquad \dfrac{(2)}{f}}{\bar{a} \vee b}$$

(6) This step is obtained when closing the left branch of the tree: it is a resolution between the piece of tree labeling the edge above a (3) and the the piece of tree labeling the edge above $\bar{a}$ (5):

$$\dfrac{\dfrac{(3)}{a} \qquad \dfrac{(5)}{\bar{a} \vee b}}{b}$$

Finally, we consider the closure of the last branch, as a resolution between the piece of tree labeling the edge above $\bar{b}$ (4) and the the piece of tree labeling the edge above b (6):

$$\dfrac{\dfrac{(4)}{\bar{b}} \qquad \dfrac{(6)}{b}}{\square}$$

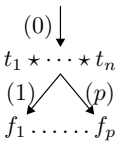
Putting everything together, we obtain the following proof of the empty clause from g:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\bar{g} \vee f} \quad g}{f} \qquad \dfrac{}{\bar{f} \vee \bar{a} \vee b}}{\bar{a} \vee b} \qquad \dfrac{g \quad \dfrac{}{\bar{g} \vee a}}{a}}{b} \qquad \dfrac{g \quad \dfrac{}{\bar{g} \vee \bar{b}}}{\bar{b}}}{\square}$$

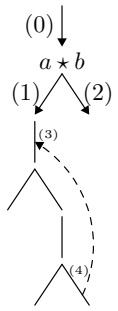
3.2.2 Formal description

Algorithm As we explained, the first step is to assign fresh variables to each (non-strict) sub-formula of the initial formula which is not a literal, and to define the very simple piece of certificate – which is a clause containing only one literal – stating that the initial formula holds.

The second step consists in successively labeling the edges with pieces of certificates, from top to bottom and from left to right. We consider the generic rule decomposing a connective $\star$:



where (0) has already been computed. We explain how to compute (1) to (p). The label of the left edge, (1), is a resolution between (0) and the rule of extended resolution which corresponds to the decomposition of $\star$. Then, for $i \in [\![2; p]\!]$, (i) is the resolution between the two pieces of certificates leading to the two formulas that finally close the branch directly on the left. For instance, on the following tableau shape, (2) is the resolution between (3) and (4).



Finally, we obtain the empty clause by a resolution between the two pieces of certificates leading to the two formulas that close the last branch.

Remarks The correctness of this algorithm mainly relies on the following invariant: a piece of certificate labeling an edge above the formula f proves a clause containing f. This ensures that the resolutions performed are always possible.

For each edge in the tableau proof, the corresponding piece of certificate contains at most two nodes: a resolution and possibly a rule of extended resolution. The final step adds a resolution to the final proof. It entails that the number of nodes in the obtained proof is linear in the number of nodes in the tableau proof.

4 Certificates for reduced ordered binary decision diagrams

A reduced ordered binary decision diagram is a normalized decision tree of a propositional formula. BDDs enjoy the property to be canonical [18]: two equivalent formulas have the same BDD – up to the order of the variables, as we will see below. As a result, it provides a decision procedure for the unsatisfiability of propositional formulas [6], which consists in progressively building the BDD of the formula, and check that the result is the false BDD.

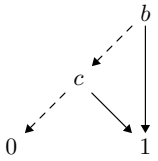
This method is rather popular since it is very efficient for certain classes of SAT problems, and well suited for circuit generation and simplification [11, 13]. Its efficiency mostly relies on the choice of a good order for the variables, which is an active research area [19].

After presenting the method for propositional logic and theoretical results concerning its power, we explain how to deduce certificates based on extended resolution from BDD proofs.

4.1 The method

A BDD is a directed acyclic graph whose nodes have either zero or two children and are labeled with propositional variables with respect to a given order. The idea is that, for every variable whose value has an influence on the formula, we construct the two sub-BDDs obtained by successively putting this variable to $\bot$ and $\top$: this is called the *Shannon expansion* of the variable.

Example 4.1. The BDD corresponding to $(a \Rightarrow (b \vee c)) \wedge (a \vee b \vee c)$ with the ordering $a > b > c$ is:



It is established that two equisatisfiable formulas have the same BDD up to the order of the variables. It entails that **every unsatisfiable formula has the** 0 **BDD**. BDDs thus give a complete method to establish the unsatisfiability of a formula F: it is sufficient to build the BDD of F and check that it is 0.

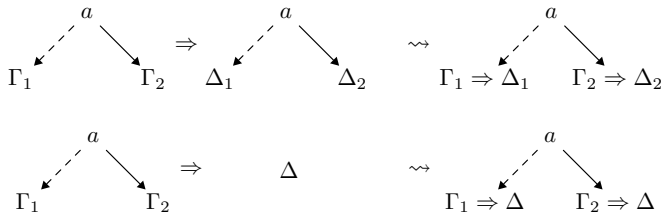
The difficulty is that the naive algorithm to compute the BDD of a formula is equivalent to computing a truth table, and thus impossible to run in practice. The idea to cope with this issue is to **build the BDD little by little and simplify it at the same time**.

To build the BDD corresponding to a formula F, we thus start with the BDDs corresponding to the variables appearing in the formulas, and we alternate between two phases:

1. building the BDD associated to a sub-formula of F of which every sub-formula has already been treated;
2. simplifying the obtained BDD.

The first step consists in recursively running through the BDDs concerned by the connective, until we reach the leaves. The following example presents the rules corresponding to the implication.

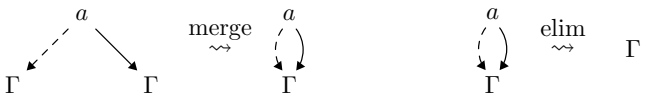
Example 4.2. Given two BDDs, their implication can be constructed using the following rules:



$$0 \Rightarrow \Delta \rightsquigarrow 1 \qquad 1 \Rightarrow \Delta \rightsquigarrow \Delta \qquad \Gamma \Rightarrow 0 \rightsquigarrow \neg\Gamma \qquad \Gamma \Rightarrow 1 \rightsquigarrow 1$$

The second rule applies when the top variable of Δ is smaller than a. The symmetric rule when this variable is greater than a is similar.

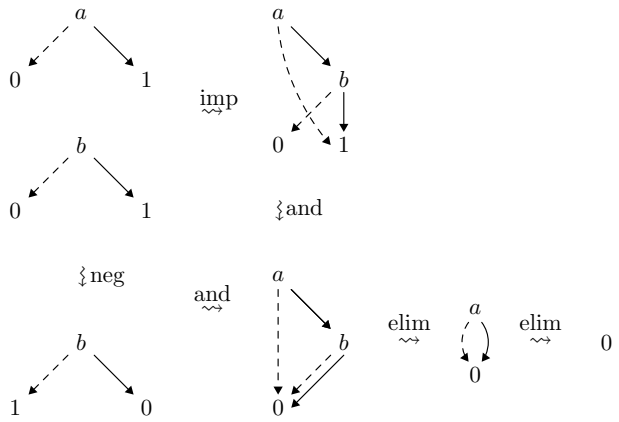
The second step is an application of the following two rules, respectively called *merge* and *elim*, until the BDD is normalized:



This step is the key point to avoid building exponential BDDs.

We give as an example the proof of unsatisfiability of $(a \Rightarrow b) \wedge a \wedge \bar{b}$.

Example 4.3. The following steps build the proof of unsatisfiability of $(a \Rightarrow b) \wedge a \wedge \bar{b}$:



We already theoretically know that extended resolution p-simulates BDDs (**Corollary 1** of [24]).

4.2 From BDDs to extended resolution

In this section, we describe a generic algorithm to transform any BDD proof of unsatisfiability into a proof of the empty clause in extended resolution, **without requiring the initial formula to be in normal form**. It produces a proof tree whose number of nodes is **polynomial** in the number of nodes in the length of the whole BDD proof.

Peltier [24] proved that extended resolution p-simulates BDDs, and this proof of course contains an algorithm to transform a BDD proof into a proof in extended resolution. We propose here a variant which is more implementation-oriented. Some ideas remain the same, but contrary to [24], we build the clauses corresponding to extended rules on demand (and not at the beginning), which changes the way we handle connectives. The construction remains polynomial.

The idea is the following:

- we define how a set of clauses can represent a BDD: it mainly consists in labeling all the nodes with names, and expressing the Shannon expansion in terms of clauses *a la* Tseitin;
- starting from the variables, we progressively build both the BDD and the resolution proof: for each step of the building of a BDD, we explain how to transform a set of clauses representing the initial BDD into a set of clauses representing the final BDD, by applying the rules of extended resolution;
- in the end, since we obtain the 0 BDD, we have built a close proof in extended resolution of the negation of the original formula. It only remains to resolve with the original formula.

This algorithm thus builds two kinds of objects:

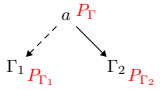
- the sets of clauses representing the intermediate BDDs: it is not mandatory to actually construct them, but the fact that they represent the intermediate BDDs is the invariant making this algorithm correct;
- the resolution proof produced little by little to switch between these sets of clauses: this is the final output of the algorithm.

This time we do not first present the algorithm on an example, since the proof which is obtain (even for a simple formula like $a \wedge \bar{a}$) is rather huge and unreadable; it is in fact easier to understand each step in the general case.

4.2.1 Algorithm

A set of clauses representing a BDD relates connected nodes by clauses.

First, all the nodes have been given fresh names. Then, for each internal node, we add to the set four clauses corresponding to the Shannon expansion that this node represent:

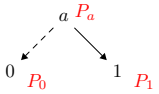


adds the four clauses $\bar{P}_\Gamma \vee a \vee P_{\Gamma_1}; P_\Gamma \vee a \vee \bar{P}_{\Gamma_1}; \bar{P}_\Gamma \vee \bar{a} \vee P_{\Gamma_2}; P_\Gamma \vee \bar{a} \vee \bar{P}_{\Gamma_2}$ to the set.

Finally, we add one clause a leaf depending on its value: 0_{P_0} adds the clause $\bar{P}_0$ and 1_{P_1} adds the clause P_1.

Example 4.4.

- The set of clauses associated to the BDD 0_{P_0} is $\{\bar{P}_0\}$.
- The BDD of a variable a is:



  The corresponding set of clauses is $\{\bar{P}_a \vee a \vee P_0; P_a \vee a \vee \bar{P}_0; \bar{P}_a \vee \bar{a} \vee P_1; P_a \vee \bar{a} \vee \bar{P}_1; \bar{P}_0; P_1\}$.
- We come back to **Example 4.1** and give the names P_b to the node labeled with b, P_c to the node labeled with c, P_0 to the node labeled with 0 and P_1 to the node labeled with 1. The set of clauses associated to this BDD is $\{\bar{P}_b \vee b \vee P_c; P_b \vee b \vee \bar{P}_c; \bar{P}_b \vee \bar{b} \vee P_1; P_b \vee \bar{b} \vee \bar{P}_1; \bar{P}_c \vee c \vee P_0; P_c \vee c \vee \bar{P}_0; \bar{P}_c \vee \bar{c} \vee P_1; P_c \vee \bar{c} \vee \bar{P}_1; \bar{P}_0; P_1\}$.

All these clauses correspond to the rules for the ite, $\top$ or $\bot$ connectives in extended resolution, and are thus provable in extended resolution.

As we said, the algorithm consists in starting with the BDDs of the variables and the corresponding sets of clauses, and then successively apply the connectives, merge and elim rules both on the BDDs and on the set of clauses. In the end, we will obtain the BDD 0_{P_Γ}, and thus a proof in extended resolution of $\bar{P}_\Gamma$, where P_Γ is the fresh variable associated to the initial formula. We will finally conclude by resolving with it.

It thus remains to explain how we transform sets of clauses representing BDDs into a set of clauses representing the BDD obtained after an application of a connective or the merge and elim rules. In this paper, we focus on the connectives, since the simplification rules are handled like in [24] (and it is roughly the same ideas as for connectives).

To handle a connective $f \star g$, we prove by induction on the sum of the number of nodes in Γ and Δ that we can at the same time:

(a) transform the sets of clauses associated to Γ and Δ into a set of clauses representing $\Gamma \star \Delta$;

(b) generate the extended rules corresponding to $\Gamma \star \Delta$;

where Γ is the BDD corresponding to f and Δ is the BDD corresponding to g. Only the first item is needed for the final algorithm to work, but the second item is required in the proof by induction. This is how we avoid to first transform the formula into a set of clauses, like in [24].

We show the proof in the case of the implication: this both variant and covariant connective illustrates well the process.

Base cases We consider only the base case $0_{P_\Gamma} \Rightarrow \Delta_{P_\Delta} \rightsquigarrow 1_{P_{\Gamma\Rightarrow\Delta}}$ since the others are similar.

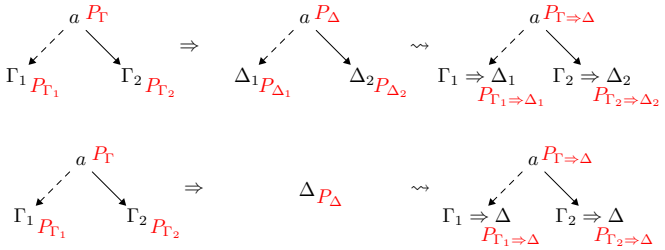
We first define the name $P_{\Gamma\Rightarrow\Delta}$ by extended resolution:

$$\frac{}{\bar{P}_{\Gamma\Rightarrow\Delta} \vee \bar{P}_\Gamma \vee P_\Delta}\ (1) \qquad \frac{}{P_{\Gamma\Rightarrow\Delta} \vee P_\Gamma}\ (2) \qquad \frac{}{P_{\Gamma\Rightarrow\Delta} \vee \bar{P}_\Delta}\ (3)$$

This already fulfills step (b).

The set of clauses representing 0 is $\{\bar{P}_\Gamma\}$, obtained by extended resolution on the connective $\bot$. By resolving it with (2), we obtain a proof of $P_{\Gamma\Rightarrow\Delta}$, which fulfills step (a).

Inductive cases The inductive cases correspond to the following labeling:



We are going to concentrate only on the first one; the second one is similar (with even fewer resolutions).

The induction hypothesis for (b) gives us:

$$\begin{cases} \bar{P}_{\Gamma_1\Rightarrow\Delta_1} \vee \bar{P}_{\Gamma_1} \vee P_{\Delta_1} & (1) \\ P_{\Gamma_1\Rightarrow\Delta_1} \vee P_{\Gamma_1} & (2) \\ P_{\Gamma_1\Rightarrow\Delta_1} \vee \bar{P}_{\Delta_1} & (3) \end{cases} \text{ and } \begin{cases} \bar{P}_{\Gamma_2\Rightarrow\Delta_2} \vee \bar{P}_{\Gamma_2} \vee P_{\Delta_2} & (4) \\ P_{\Gamma_2\Rightarrow\Delta_2} \vee P_{\Gamma_2} & (5) \\ P_{\Gamma_2\Rightarrow\Delta_2} \vee \bar{P}_{\Delta_2} & (6) \end{cases}$$

The induction hypothesis for (a) is:

$$\begin{cases} \bar{P}_\Gamma \vee a \vee P_{\Gamma_1} & (7) \\ P_\Gamma \vee a \vee \bar{P}_{\Gamma_1} & (8) \\ \bar{P}_\Gamma \vee \bar{a} \vee P_{\Gamma_2} & (9) \\ P_\Gamma \vee \bar{a} \vee \bar{P}_{\Gamma_2} & (10) \end{cases} \text{ and } \begin{cases} \bar{P}_\Delta \vee a \vee P_{\Delta_1} & (11) \\ P_\Delta \vee a \vee \bar{P}_{\Delta_1} & (12) \\ \bar{P}_\Delta \vee \bar{a} \vee P_{\Delta_2} & (13) \\ P_\Delta \vee \bar{a} \vee \bar{P}_{\Delta_2} & (14) \end{cases}$$

We first define (b) by extension:

$$\begin{cases} \bar{P}_{\Gamma\Rightarrow\Delta} \vee \bar{P}_\Gamma \vee P_\Delta & (15) \\ P_{\Gamma\Rightarrow\Delta} \vee P_\Gamma & (16) \\ P_{\Gamma\Rightarrow\Delta} \vee \bar{P}_\Delta & (17) \end{cases}$$

and then (a) by resolution:

$$\begin{cases} \bar{P}_{\Gamma \Rightarrow \Delta} \vee a \vee P_{\Gamma_1 \Rightarrow \Delta_1} & \text{(resolution of } 15, 8, 11, 2, 3) \\ P_{\Gamma \Rightarrow \Delta} \vee a \vee \bar{P}_{\Gamma_1 \Rightarrow \Delta_1} & \text{(resolution of } 17, 12, 1, 7, 16) \\ \bar{P}_{\Gamma \Rightarrow \Delta} \vee \bar{a} \vee P_{\Gamma_2 \Rightarrow \Delta_2} & \text{(resolution of } 15, 10, 13, 5, 6) \\ P_{\Gamma \Rightarrow \Delta} \vee \bar{a} \vee \bar{P}_{\Gamma_2 \Rightarrow \Delta_2} & \text{(resolution of } 17, 14, 4, 9, 16) \end{cases}$$

4.2.2 Remarks

As proved in [24], the whole algorithm – together with the transformation of the merge and elim rules – builds a proof whose number of nodes is polynomial in the length of the original BDD proof.

As we said, the correctness of this algorithm relies on the fact that the intermediate sets of clauses always represent the intermediate BDDs, in order to obtain in the end a proof of the negation of the initial formula.

We think that the new presentation for connectives makes this algorithm easier to implement: the functions combining BDDs for each connective just have to return the clauses generated by the (b) step in addition to the (a) step, instead of somehow looking into a large set initially computed.

5 Discussion

5.1 Related works

Lots of related works were already presented throughout the paper.

To our knowledge, this is the first transformation of full propositional tableaux (and not only clausal tableaux) into extended resolution, and this is the first work aiming at providing certificates that can be checked by an external tool (instead of a theoretical comparison of two proof systems). The Isabelle tableau prover [21] gives witnesses, but encoded directly into Isabelle proofs, and thus not applicable to systems not based on Higher-Order Logic.

The BDD algorithm highly relies on [24], but in an implementation perspective (as we argued in Section 4.2.2). [26] presented an implementation of a translator from clausal BDDs into extended resolution, but this is limited to CNF formulas, and requires to treat lots of particular cases whereas our algorithm is more generic.

Even if this work relies on different previous works for the different parts, this is a first attempt to unify certificates for three major paradigms for propositional proving: DPLL with backjumping, the method of tableaux, and BDDs.

Other proof formats for certificates in propositional logic have been proposed. The format based on extended resolution used in [26] called TraceCheck (which is in particular returned by the SAT solver BooleForce) is very close to ours, and thus could be directly used by our algorithms. The recent Reverse Unit Propagation format [15, 16] (RUP in short) gives shorter proofs than resolution, but is currently restricted to inputs in CNF– whereas tableaux and BDD provers deal with the full propositional logic without requiring preprocessing.

5.2 Future works

Obviously, the next step is to instrument existing provers in order to return these certificates, and to evaluate the efficiency (in particular, it must not be costly to output and check the

certificate compared to finding the proof). This could then be plugged into a certified checker like SMTCoq [2], in order to check *a posteriori* the answers given by the automatic provers.

The tableaux and BDD algorithms we presented here are rather naive, and we need to understand how the variants that are actually implemented by the provers enter into this schema. Some improvements do not affect our algorithms, like the choice of the order of the variables in BDDs, but others may require changes.

Our algorithms could also be extended with other features, in particular quantifiers: we know how to extend our certificates with quantifiers [10], and they are well handled by tableaux proofs. We would also like to deal with other logics than classical logic, like intuitionistic or modal logic, for which tableaux are quite frequently used.

A broad spectrum study should also extend this work to other proof formats, like an extension of the RUP proofs [15] to full propositional logic, as well as other proof search paradigms, like stochastic search algorithms.

Acknowledgments The author thanks Filip Marić who asked a question that motivated this work, and the anonymous reviewers for their insightful comments.

6 Conclusion

In this paper, we presented two new algorithms to transform into certificates in extended resolution the proofs computed by two major propositional provers: tableaux provers and BDDs. Since this translation is already efficiently implemented for SAT solvers based on DPLL with backjumping, this opens the way towards a common format for certificates for propositional solvers for which we already have efficient certified checkers.

References

[1] M. Armand, B. Grégoire, A. Spiwack, and L. Théry. Extending Coq with Imperative Features and Its Application to SAT Verification. In Kaufmann and Paulson [17], pages 83–98.

[2] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In Jean-Pierre Jouannaud and Zhong Shao, editors, *CPP*, volume 7086 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2011.

[3] F. Besson, P. Fontaine, and L. Théry. A Flexible Proof Format for SMT: a Proposal. In *PxTP 2011: First International Workshop on Proof eXchange for Theorem Proving August 1, 2011 Affiliated with CADE 2011, 31 July-5 August 2011 Wrocław, Poland*, pages 15–26, 2011.

[4] S. Böhme and T. Weber. Fast LCF-Style Proof Reconstruction for Z3. In Kaufmann and Paulson [17], pages 179–194.

[5] T. Bouton, D.C.B. de Oliveira, D. Déharbe, and P. Fontaine. veriT: An Open, Trustable and Efficient SMT-Solver. In R. A. Schmidt, editor, *CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156. Springer, 2009.

[6] Randal E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.

[7] Stephen A Cook. A Short Proof of the Pigeon Hole Principle using Extended Resolution. *ACM SIGACT News*, 8(4):28–32, 1976.

[8] Stephen A. Cook and Robert A. Reckhow. On the Lengths of Proofs in the Propositional Calculus (Preliminary Version). In Robert L. Constable, Robert W. Ritchie, Jack W. Carlyle, and Michael A. Harrison, editors, *STOC*, pages 135–148. ACM, 1974.

[9] Marcello D'Agostino, Dov M Gabbay, Reiner Hähnle, and Joachim Posegga. *Handbook of Tableau Methods*. Springer, 1999.

[10] D. Deharbe, P. Fontaine, and B. W. Paleo. Quantier Inference Rules for SMT proofs. In *PxTP 2011: First International Workshop on Proof eXchange for Theorem Proving August 1, 2011 Affiliated with CADE 2011, 31 July-5 August 2011 Wrocław, Poland*, pages 33–39, 2011.

[11] Rolf Drechsler, Junhao Shi, and Görschwin Fey. Synthesis of Fully Testable Circuits From BDDs. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(3):440–443, 2004.

[12] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[13] Görschwin Fey, Junhao Shi, and Rolf Drechsler. BDD Circuit Optimization for Path Delay Fault Testability. In *DSD*, pages 168–172. IEEE Computer Society, 2004.

[14] Z. Fu, Y. Marhajan, and S. Malik. zChaff. *Research Web Page. Princeton University, USA,(March 2007)* `http://www.princeton.edu/~chaff/zchaff.html`, 2007.

[15] Allen Van Gelder. Verifying RUP Proofs of Propositional Unsatisfiability. In *ISAIM*, 2008.

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

[17] M. Kaufmann and L. C. Paulson, editors. *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6172 of *Lecture Notes in Computer Science*. Springer, 2010.

[18] D. Kroening and O. Strichman. *Decision Procedures: an Algorithmic Point of View*. Springer-Verlag New York Inc, http://www.decision-procedures.org, 2008.

[19] A. Layeb, N. Tabib, and B. Brirem. Genetic Algorithm and Variable Neighbourhood Search for BDD Variable Ordering Problem. *INFOCOMP Journal of Computer Science*, 10(1):29–35, 2011.

[20] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(). *J. ACM*, 53(6):937–977, 2006.

[21] Lawrence C. Paulson. A Generic Tableau Prover and its Integration with Isabelle. *J. UCS*, 5(3):73–87, 1999.

[22] Lawrence C Paulson and Jasmin Christian Blanchette. Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers. In G. Sutcliffe, E. Ternovska, and S. Schulz, editors, *International Workshop on the Implementation of Logics (IWIL-2010)*, 2010.

[23] L.C. Paulson. *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[24] Nicolas Peltier. Extended Resolution Simulates Binary Decision Diagrams. *Discrete Applied Mathematics*, 156(6):825–837, 2008.

[25] John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965.

[26] Carsten Sinz and Armin Biere. Extended Resolution Proofs for Conjoining BDDs. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *CSR*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006.

[27] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.

[28] G. Tseitin. On the Complexity of Proofs in Propositional Logics. In *Seminars in Mathematics*, volume 8, pages 466–483, 1970.

[29] Alasdair Urquhart. The Complexity of Propositional Proofs. *Bulletin of Symbolic Logic*, 1(4):425–467, 1995.