# Unrestricted vs Restricted Cut in a Tableau Method for Boolean Circuits*

Matti Järvisalo, Tommi Junttila, and Ilkka Niemelä

Laboratory for Theoretical Computer Science
Helsinki University of Technology
P.O. Box 5400, FI-02015 TKK, Finland
matti.jarvisalo,tommi.junttila,ilkka.niemela@tkk.fi

**Abstract.** This paper studies the relative efficiency of variations of a tableau method for Boolean circuit satisfiability checking. The considered method is a non-clausal generalisation of the Davis-Putnam-Logemann-Loveland (DPLL) procedure to Boolean circuits. The variations are obtained by restricting the use of the cut (splitting) rule in several natural ways. It is shown that the more restricted variations cannot polynomially simulate the less restricted ones. For each pair of methods $T$, $T'$, an infinite family $\{\mathcal{C}_n\}$ of circuits is devised for which $T$ has polynomial size proofs while in $T'$ the minimal proofs are of exponential size w.r.t. $n$, implying exponential separation of $T$ and $T'$ w.r.t. $n$.
The results also apply to DPLL for formulas in conjunctive normal form obtained from Boolean circuits by using Tseitin's translation. Thus DPLL with the considered cut restrictions, such as allowing splitting only on the variables corresponding to the input gates, cannot polynomially simulate DPLL with unrestricted splitting.

## 1 Introduction

The *propositional satisfiability problem* (SAT) of determining whether a given propositional formula has a truth assignment under which it evaluates to `true` is an archetypical **NP**-complete problem, see e.g. [26]. Because of its universal nature, a variety of important problems, e.g., in the areas of planning [19, 20], model checking of finite state systems [5, 4], testing [22], and hardware verification [3], can be reduced to SAT. Due to this, there is a high demand for more feasible ways of solving SAT instances, ranging from industrial applications to pure research. Various methods for solving SAT instances have been developed (see [14] and [32] for surveys) and applied successfully to many interesting domains.

Recognising the factors that affect the difficulty of satisfiability checking, i.e. the time needed to determine whether an instance is satisfiable or not, is crucial when developing more efficient methods for the task. The basis of most

* The financial support from Academy of Finland under grant 53695 is gratefully acknowledged.

state-of-the-art SAT checkers today is the *Davis-Putnam-Logemann-Loveland procedure* (DPLL) [10, 9]. The efficiency of a typical DPLL based SAT checking system depends on

– the applied search space pruning techniques, e.g., *non-branching deduction rules*, *non-chronological backtracking* (see e.g. [23]), and *conflict-driven learning* (see e.g. [31]), and on
– the *splitting rule*, i.e., on which Boolean variables to apply the *explicit cut* that induces branching, and what kind of *heuristics* is this decision based on.

For measuring the efficiency of SAT checking methods there are several alternatives. One can compare SAT checkers by *experimental evaluation*, i.e., investigate how long it takes for checkers to solve different types of instances. Another approach is *worst-case analysis* of SAT checking algorithms (see e.g. [8]), i.e., giving analytic proofs of upper bounds on the running times of algorithms w.r.t. the instance size. A third approach, the one taken in this work, is to investigate how large the minimal-size proofs (refutations) are for different families of formulas. This measure is called *proof complexity*, see e.g. [2]. Proof complexity is of our interest as it allows one to differentiate heuristic performance from the proof rules in a method and to consider how small proofs can be established assuming *optimal* heuristic behaviour.

Relative efficiency of proof systems can be measured using the notion of *polynomial simulation*. If a proof method $T$ can polynomially simulate another method $T'$, then $T$ is considered to be at least as efficient as $T'$. Showing that $T'$ cannot polynomially simulate $T$ gives a way of establishing that $T$ is substantially stronger than $T'$. This is because the lack of polynomial simulation means that moving from $T$ to $T'$ cannot be done with only a polynomial loss of efficiency, i.e., there are proofs in $T$ which do not have any polynomial size counter-part in $T'$.

Currently most successful DPLL-based SAT checkers assume that the input formulas are in *conjunctive normal form* (CNF). The reason for this is that it is simpler to develop efficient data structures and algorithms for CNF than for arbitrary formulas. On the other hand, using CNF makes efficient modelling of an application cumbersome. Fortunately, propositional formulas can be transformed in polynomial time into CNF while preserving the satisfiability of the instance, see e.g. [27]. Therefore one usually employs a more general formula representation in modelling and then transforms the formula into CNF. However, such a polynomial time translation introduces auxiliary variables which can have an exponential effect on the performance of a typical SAT checker in the worst-case.

In addition, by translating other representations to CNF one often hides information about the structure of the original problem. One way of representing propositional formulas in a more general, structure-preserving way is to use *Boolean circuits*, see e.g. [26]. Basically, Boolean circuits are acyclic directed graphs in which the nodes—representing sub-formulas of the instance—are called *gates*, and the edges represent dependencies between the gates. Boolean circuits are interesting because they allow for a compact and natural representation that can be simplified by sharing common subexpressions, while preserving natural structures and concepts of the domain. Boolean circuits can be translated into CNF using a standard translation often referred to as *Tseitin's translation* [30]. This translation introduces a new variable for each gate in the circuit, resulting in a linear size CNF.

In this work we are interested in solving Boolean circuit satisfiability problems using an approach that exploits the highly successful DPLL type techniques but works directly on circuits. One approach is to translate the circuit to CNF and use the clausal DPLL method as the basis but add extra information from the circuit to enhance the performance of the method. See e.g. [13, 24] for interesting work in this direction. Another approach is to develop a generalisation of the DPLL method that works directly on the circuit structure. This direction has been pursued, e.g., in [18, 21, 11, 29]. Here we study the latter approach and use as the basis of the work a simplified version of a *tableau method* for Boolean circuit satisfiability checking that works directly with circuits [18] (see [17] for an implementation of the method). The method is a non-clausal generalisation of DPLL to Boolean circuits which does not include learning or non-chronological backtracking techniques (see [29] for recent work on incorporating these techniques to a generalised DPLL). The method is closely related to standard tableau techniques [6] but works directly on a circuit (rather than formula) representation. Moreover, it employs a direct cut rule combined with deterministic (non-branching) deduction rules making it similar to the tableau system KE [7]. More information on the advantages of using a direct cut rule compared to typical cut free tableaux can be found in [6, 7, 25].

In this work we focus on the splitting/cut rule of the tableau method for Boolean circuits, the research problem being:

> *How do restrictions on the use of the cut rule affect proof complexity in Boolean circuit satisfiability checking based on tableaux?*

For instance, one may think that it is a good idea to restrict the cuts to the *input gates* only as they determine the values of all other gates. Therefore, the search space for a circuit with $K$ gates and $N$ input gates, $K \geq N$, would be $2^N$ instead of $2^K$. This approach is proposed, for example, in [28, 12, 13]. However, our results show that this kind of a restricted cut rule cannot polynomially simulate the unrestricted cut rule. In particular, we show that there is an infinite family $\{\mathcal{C}_n\}$ of circuits which have polynomial size proofs using the unrestricted cut rule but for which minimal proofs are of exponential size w.r.t. $n$ if cuts are restricted to input gates only. In addition to the input gate restricted cuts, we study several other natural restrictions that are based on the structure of the circuit and the state of the search. Examples of such *dynamic* restrictions are "top-down" cuts that can be applied only on the children of the gates with a determined value in the current state of the search and "bottom-up" cuts that can be applied on input gates and on the parents of the already determined gates. Our results show that none of the considered restricted variations can polynomially simulate the unrestricted cut rule. Furthermore, we devise families of circuits for which the size of the proofs using different cut rules differs exponentially, i.e. for rules $R_1$ and $R_2$ we construct an infinite family $\{\mathcal{C}_n\}$ of circuits which have polynomial size proofs using $R_2$ but for which minimal proofs are of exponential size in $n$ using $R_1$, implying exponential separation of $R_1$ and $R_2$ w.r.t. $n$.

The main results in this paper directly apply to DPLL based SAT checkers without conflict-driven learning for CNF formulas obtained from Boolean circuits by using Tseitin's translation. This is because the rules of the introduced tableau method match those of DPLL under CNF clauses produced by Tseitin's translation.

The rest of this work is organised as follows. Basic concepts of Boolean circuits are introduced in Section 2. The tableau method and its locality based variations are described in Section 3. The concepts of proof complexity and polynomial simulation are explained in Section 4. In addition, Section 4 establishes some basic results concerning these concepts and the tableau method. The main results of this work with proofs are presented in Section 5. The relevance of the results to the DPLL method is established in Section 6. Finally, Section 7 concludes and gives some future research directions based on this work.

## 2 Boolean Circuits

Informally, a *Boolean circuit* (see e.g. [26]) is an acyclic directed graph in which the nodes are called *gates*. The gates can be divided into three categories: (i) a unique *output gate* with incoming edges but no outgoing edges, (ii) *intermediate gates* with both incoming and outgoing edges, and (iii) *input gates* with outgoing edges but no incoming edges. A Boolean function is associated to the output gate and each intermediate gate.

Formally, we present a Boolean circuit $\mathcal{C}$ with the set of gates $\mathcal{V}$ as a set of equations of the form $v = f(v_1, \ldots, v_k)$, where $v, v_1, \ldots, v_k \in \mathcal{V}$ and $f$ is a Boolean function. It is required that (i) each $v \in \mathcal{V}$ has at most one equation, (ii) the equations are non-recursive, and (iii) exactly one gate (i.e. the output gate) does not appear on the right hand side of any equation. We define the *size* of a Boolean circuit to be the number of gates and edges in the circuit. For a Boolean circuit $\mathcal{C}$, we denote the set of gates appearing in $\mathcal{C}$ by $V(\mathcal{C})$.

*Example 1.* Graphically, the Boolean circuit

$$\{v = \mathsf{and}(e, f, g, h),\ e = \mathsf{or}(a, b),\ f = \mathsf{or}(b, c),\ g = \mathsf{or}(a, d),$$
$$h = \mathsf{or}(c, d),\ c = \mathsf{not}(a),\ d = \mathsf{not}(b)\}$$

is shown in Figure 1. In this circuit, $a$ and $b$ are input gates, $c, d, e, f, g$ and $h$ intermediate gates, and $v$ is the output gate.

A *truth assignment* for a Boolean circuit $\mathcal{C}$ is a function $\tau : V(\mathcal{C}) \to \{\mathtt{true}, \mathtt{false}\}$. Assignment $\tau$ is *consistent* if $\tau(v) = f(\tau(v_1), \ldots, \tau(v_k))$ holds for each equation $v = f(v_1, \ldots, v_k)$ in $\mathcal{C}$. A consistent truth assignment that assigns true to the output gate of a circuit is a *satisfying truth assignment* for the circuit. If there

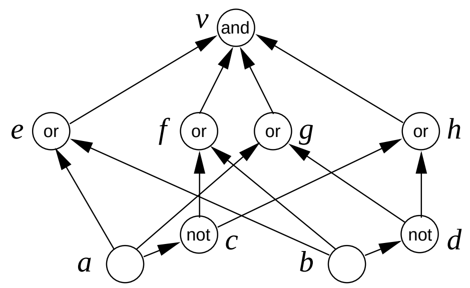


**Fig. 1.** A Boolean circuit.

is a satisfying truth assignment for a circuit, the circuit is *satisfiable*. Otherwise the circuit is *unsatisfiable*.

Here, we are interested in CIRCUIT SAT, the *satisfaction problem* for Boolean circuits, in which we are given a Boolean circuit and asked whether the circuit is satisfiable. The CIRCUIT SAT problem is obviously **NP**-complete and closely related to SAT.

In the following we consider the class of Boolean circuits in which the following three types of Boolean functions are allowed.

- $\mathsf{not}(v) = \mathtt{true}$ if and only if $v$ is $\mathtt{false}$,
- $\mathsf{or}(v_1, \dots, v_k) = \mathtt{true}$ if and only if at least one $v_i$, $1 \leq i \leq k$, $k \geq 2$, is $\mathtt{true}$, and
- $\mathsf{and}(v_1, \dots, v_k) = \mathtt{true}$ if and only if all $v_i$, $1 \leq i \leq k$, $k \geq 2$, are $\mathtt{true}$.

Notice that it is straightforward to extend this class with additional Boolean functions such as $\mathsf{xor}$ and equivalence (see e.g. [18]).

Consider a propositional formula in conjunctive normal form, i.e. a set $\varphi = \{C_1, \dots, C_k\}$ of *clauses* of the form

$$C_i = (v_{i,1} \vee \cdots \vee v_{i,m_i} \vee \neg w_{i,1} \vee \cdots \vee \neg w_{i,n_i}).$$

The *canonical Boolean circuit representation* of $\varphi$ is

$$\begin{aligned} C(\varphi) = \{&v = \mathsf{and}(o_1, \dots, o_k)\} \cup \\ &\{o_i = \mathsf{or}(a_{v_{i,1}}, \dots, a_{v_{1,m_i}}, b_{w_{i,1}}, \dots, b_{w_{i,n_i}}) \mid 1 \leq i \leq k\} \cup \\ &\{b_{w_{i,j}} = \mathsf{not}(a_{w_{i,j}}) \mid 1 \leq i \leq k, 1 \leq j \leq n_i\}, \end{aligned}$$

where $v$ is the output gate of $C(\varphi)$. It is straightforward to establish that the CNF formula $\varphi$ is satisfiable if and only if the Boolean circuit $C(\varphi)$ is satisfiable.

*Example 2.* In Figure 1 the canonical Boolean circuit representation $C(\mathrm{UNSAT}_{a,b})$ is shown for

$$\mathrm{UNSAT}_{a,b} \stackrel{\mathrm{def}}{=} \{(a \vee b), (a \vee \neg b), (\neg a \vee b), (\neg a \vee \neg b)\}.$$

## 3 A Tableau Method

We study a tableau method for Boolean circuit satisfiability checking we call **BC**. The method consists of the rules shown in Figure 2. It is a simplified version of the tableau system introduced in [18].[1] The method is a non-clausal generalisation of DPLL to Boolean circuits: the explicit cut rule (a) in **BC** corresponds to the splitting rule in DPLL, and the rules (b)-(h) correspond to unit clause propagation, i.e., standard Boolean constraint propagation.

$$\frac{v \in \mathcal{V}}{\mathbf{T}v \mid \mathbf{F}v}$$

(a) The explicit cut rule

$$\frac{v = \mathsf{not}(v_1) \quad \mathbf{F}v_1}{\mathbf{T}v} \qquad \frac{v = \mathsf{not}(v_1) \quad \mathbf{T}v_1}{\mathbf{F}v} \qquad \frac{v = \mathsf{not}(v_1) \quad \mathbf{F}v}{\mathbf{T}v_1} \qquad \frac{v = \mathsf{not}(v_1) \quad \mathbf{T}v}{\mathbf{F}v_1}$$

(b) "Up" rules for $\mathsf{not}$ (c) "Down" rules for $\mathsf{not}$

$$\frac{v = \mathsf{or}(v_1, \dots, v_k) \quad \mathbf{F}v_1, \dots, \mathbf{F}v_k}{\mathbf{F}v} \qquad \frac{v = \mathsf{or}(v_1, \dots, v_k) \quad \mathbf{T}v_i, i \in \{1, \dots, k\}}{\mathbf{T}v} \qquad \frac{v = \mathsf{or}(v_1, \dots, v_k) \quad \mathbf{F}v}{\mathbf{F}v_1, \dots, \mathbf{F}v_k}$$

(d) "Up" rules for $\mathsf{or}$ (e) "Down" rule for $\mathsf{or}$

$$\frac{v = \mathsf{and}(v_1, \dots, v_k) \quad \mathbf{T}v_1, \dots, \mathbf{T}v_k}{\mathbf{T}v} \qquad \frac{v = \mathsf{and}(v_1, \dots, v_k) \quad \mathbf{F}v_i, i \in \{1, \dots, k\}}{\mathbf{F}v} \qquad \frac{v = \mathsf{and}(v_1, \dots, v_k) \quad \mathbf{T}v}{\mathbf{T}v_1, \dots, \mathbf{T}v_k}$$

(f) "Up" rules for $\mathsf{and}$ (g) "Down" rule for $\mathsf{and}$

$$\frac{\begin{array}{c} v = \mathsf{or}(v_1, \dots, v_k) \\ \mathbf{F}v_1, \dots, \mathbf{F}v_{j-1}, \mathbf{F}v_{j+1}, \dots \mathbf{F}v_k \\ \mathbf{T}v \end{array}}{\mathbf{T}v_j} \qquad \frac{\begin{array}{c} v = \mathsf{and}(v_1, \dots, v_k) \\ \mathbf{T}v_1, \dots, \mathbf{T}v_{j-1}, \mathbf{T}v_{j+1}, \dots \mathbf{T}v_k \\ \mathbf{F}v \end{array}}{\mathbf{F}v_j}$$

(h) "Last undetermined child" rules for $\mathsf{or}$ and $\mathsf{and}$

**Fig. 2.** Tableau method **BC** for Boolean circuits.

Given a Boolean circuit $\mathcal{C}$, a **BC**-tableau for $\mathcal{C}$ is a binary tree such that the root node of the tree consists of the equations in $\mathcal{C}$ and additionally the entry $\mathbf{T}v$, where $v$ is the output gate of $\mathcal{C}$. The other nodes in the tree are entries of the form $\mathbf{T}v$ or $\mathbf{F}v$, where $v \in V(\mathcal{C})$, generated by extending the tableau using the rules in Figure 2 in the following standard way [6]. Given a tableau rule and a branch in the tableau such that the prerequisites of the rule hold in the branch, the tableau can be extended by adding new nodes to the end of the branch as

[1] The method introduced in [18] provides additionally, e.g., rules for $\mathsf{xor}$ and equivalence gates as well as circuit simplification rules.

specified by the rule. If the rule is (a), then entries $\mathbf{T}v$ and $\mathbf{F}v$ are added as the left and right child in the end of the branch. For the other rules, the consequents of the rule are added to the end of the branch (as a linear subtree in case of multiple consequents).

A branch in the tableau is *contradictory* if it contains both $\mathbf{F}v$ and $\mathbf{T}v$ entries for a gate $v \in V(\mathcal{C})$. Otherwise, the branch is *open*. A branch is *complete* if it is contradictory, or if there is a $\mathbf{F}v$ or a $\mathbf{T}v$ entry for each $v \in V(\mathcal{C})$ in the branch and the branch is closed under the rules (b)–(h). A tableau is *finished* if all the branches of the tableau are complete. A tableau is *closed* if all of its branches are contradictory. A closed **BC**-tableau for a circuit is called a **BC**-*refutation* for the circuit. For each $v \in V(\mathcal{C})$, we say that the entry $\mathbf{T}v$ ($\mathbf{F}v$) can be *deduced* in a branch if the entry $\mathbf{T}v$ ($\mathbf{F}v$) can be generated by applying rules (b)–(h) only.

*Example 3.* For the circuit shown in Figure 1, a **BC**-refutation is shown in Figure 3. For instance, in the refutation the entry $\mathbf{F}c$ (15) is deduced from the entries $c = \mathsf{not}(a)$ and $\mathbf{T}a$, while $\mathbf{T}a$ (13) is generated by applying the cut rule.

1. $v = \mathsf{and}(e, f, g, h)$
2. $e = \mathsf{or}(a, b)$
3. $f = \mathsf{or}(b, c)$
4. $g = \mathsf{or}(a, d)$
5. $h = \mathsf{or}(c, d)$
6. $c = \mathsf{not}(a)$
7. $d = \mathsf{not}(b)$
8. $\mathbf{T}v$
9. $\mathbf{T}e$ $(1, 8)$
10. $\mathbf{T}f$ $(1, 8)$
11. $\mathbf{T}g$ $(1, 8)$
12. $\mathbf{T}h$ $(1, 8)$

| | |
|---|---|
| 13. $\mathbf{T}a$ (Cut) | 14. $\mathbf{F}a$ (Cut) |
| 15. $\mathbf{F}c$ $(6, 13)$ | 20. $\mathbf{T}b$ $(2, 9, 14)$ |
| 16. $\mathbf{T}b$ $(3, 10, 15)$ | 21. $\mathbf{T}d$ $(4, 11, 14)$ |
| 17. $\mathbf{F}d$ $(7, 16)$ | 22. $\mathbf{F}d$ $(7, 20)$ |
| 18. $\mathbf{F}h$ $(5, 15, 17)$ | 23. $\times$ $(21, 22)$ |
| 19. $\times$ $(12, 18)$ | |

**Fig. 3.** A **BC**-refutation for $C(\mathrm{UNSAT}_{a,b})$.

We study variations of **BC** in which the application of the explicit cut rule is restricted to certain types of gates. The idea is to study the effects of restrictions which are based on the circuit structure. A natural starting point is a system where cuts are restricted to input gates only. A dynamic generalisation of this idea is to allow bottom-up cuts, i.e., to start from the input gates and permit the use of the cut rule in a tableau branch for a gate only when an entry for one of its children in the circuit has been deduced in the branch. A dual approach is to start from the entry of the output gate and allow top-down cuts. It is also possible to combine these approaches. The considered variations of **BC** are thus the following.

- $\mathbf{BC}_{\mathrm{i}}$: Application of explicit cut is restricted to input gates (*input cuts*).
- $\mathbf{BC}_{\mathrm{bu}}$: Application of explicit cut in a branch is restricted to input cuts and gates $v$ for which there is a $\mathbf{T}v'$ or a $\mathbf{F}v'$ entry in the branch for some child $v'$ of $v$ in the circuit (*bottom-up cuts*).
- $\mathbf{BC}_{\mathrm{td}}$: Application of explicit cut in a branch is restricted to the output gate and gates $v$ for which there is a $\mathbf{T}v'$ or a $\mathbf{F}v'$ entry in the branch for some parent $v'$ of $v$ in the circuit (*top-down cuts*).
- $\mathbf{BC}_{\mathrm{i+td}}$: Application of explicit cut is restricted to input and top-down cuts.
- $\mathbf{BC}_{\mathrm{bu+td}}$: Application of explicit cut is restricted to bottom-up and top-down cuts.

By soundness we mean that the existence of a closed tableau for a given circuit implies that the circuit is unsatisfiable, and by completeness that there is a closed tableau for any unsatisfiable circuit.

The following soundness theorem follows straightforwardly from the observation that the deduction rules (b)–(h) preserve satisfiability, i.e., if the premises of a rule are consistent for a truth assignment, then so is the conclusion.

**Theorem 1.** *$\mathbf{BC}_{\mathrm{i}}$, $\mathbf{BC}_{\mathrm{td}}$, $\mathbf{BC}_{\mathrm{i+td}}$, $\mathbf{BC}_{\mathrm{bu}}$, $\mathbf{BC}_{\mathrm{bu+td}}$, and* **BC** *are sound proof systems for Boolean circuits.*

The following completeness theorem is obvious by the cut rule. Input cuts with the deduction rules (b)–(h) are sufficient in order to obtain a complete branch. Moreover, input cuts can be simulated with top-down cuts in a straightforward manner.

**Theorem 2.** *$\mathbf{BC}_{\mathrm{i}}$, $\mathbf{BC}_{\mathrm{td}}$, $\mathbf{BC}_{\mathrm{i+td}}$, $\mathbf{BC}_{\mathrm{bu}}$, $\mathbf{BC}_{\mathrm{bu+td}}$, and* **BC** *are complete proof systems for Boolean circuits.*

## 4 Propositional Proof Complexity and Simulation

Generally, a *propositional proof* is a certificate for the unsatisfiability of a propositional expression (e.g., of a propositional formula or Boolean circuit). *A propositional proof system* (see e.g. [2]) for a class of propositional expressions $E$ is then a polynomial-time computable predicate $T$ such that for all expressions $\alpha \in E$ it holds that $\alpha$ is unsatisfiable if and only if there is a proof $P$ for $\alpha$ such that $T(\alpha, P)$. If such a $P$ exists, it is a $T$-*proof* for $\alpha$.

For instance, resolution is a propositional proof system that produces proofs for the unsatisfiability (i.e. refutations) of propositional formulas in conjunctive normal form. Similarly, **BC** is a propositional proof system for the unsatisfiability of Boolean circuits.

Let $T$ be a proof system. The *proof complexity* (or *complexity* in short) of a propositional expression $\alpha$ in $T$ is the size of a minimal $T$-proof for $\alpha$. The *size of a resolution refutation* is defined in the standard way as the number of resolution steps in the refutation sequence. We define the *size of a* **BC**-*refutation* as the number of nodes in the closed tableau. For example, the size of the **BC**-refutation shown in Figure 3 is 14.

We use the notion of *polynomial simulation* to study the relative efficiency of proof systems. For any two proof systems $T$ and $T'$, we say that $T$ *polynomially simulates* $T'$, denoted by $T \succeq T'$, if there is a polynomial $q$ such that, for any $\alpha$, if there is a $T'$-proof for $\alpha$ of size $n$, then there is a $T$-proof for $\alpha$ of size at most $q(n)$. Hence, $T \succeq T'$ indicates that the proof system $T$ is at least as strong as $T'$ (up to a polynomial loss of efficiency). The relation $\succeq$ is transitive. If $T \succeq T'$ holds but $T' \succeq T$ does not, we write $T \succ T'$. If neither $T \succeq T'$ nor $T' \succeq T$ holds, we write $T \# T'$.

We denote by $\succeq_\chi$ the restricted form of polynomial simulation, in which $T \succeq_\chi T'$ holds if there is a polynomial $q$ such that, for any $\alpha \in \chi$, if there is a $T'$-proof for $\alpha$ of size $n$, then there is a $T$-proof for $\alpha$ of size at most $q(n)$. If both $T \succeq_\chi T'$ and $T' \succeq_\chi T$ hold, we write $T \equiv_\chi T'$.

An obvious ordering of **BC** and its restricted variations based on the polynomial simulation relation, resulting from the restricted nature of the variations, is shown in Figure 4.

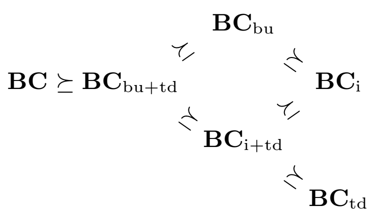


**Fig. 4.** An obvious ordering of **BC** and its variations based on the polynomial simulation relation.

It turns out that all the considered variations of the **BC** method are equivalent under the polynomial simulation relation when the set of propositional expressions considered is restricted to the set of canonical Boolean circuit representations of sets of clauses. For the following, let $\Phi$ be the family of all sets of clauses, and $C(\Phi) = \{C(\varphi) \mid \varphi \in \Phi\}$.

**Theorem 3.** $\mathbf{BC} \equiv_{C(\Phi)} \mathbf{BC}_{\mathrm{i}}$.

To see this, notice that for any set of clauses $\varphi$, using the "down" rule for and we can deduce $\mathbf{T}g$ for all or gates $g$ in $C(\varphi)$. Thus we can assume that there is a minimal-size refutation for $C(\varphi)$ in which the and rule is applied to deduce the entries concerning the or gates that are needed to achieve the closed tableau.

Then it is straightforward to see that we can limit the application of the cut rule to input gates. This shows $\mathbf{BC}_{\mathrm{i}} \succeq_{C(\Phi)} \mathbf{BC}$, while $\mathbf{BC} \succeq_{C(\Phi)} \mathbf{BC}_{\mathrm{i}}$ holds trivially.

By further noticing that for any circuit in the family $C(\Phi)$ input cuts can be polynomially simulated with top-down cuts in a straightforward manner, we have the following corollary.

**Corollary 1.** $\mathbf{BC} \equiv_{C(\Phi)} T$ *for all* $T \in \{\mathbf{BC}_{\mathrm{i}}, \mathbf{BC}_{\mathrm{td}}, \mathbf{BC}_{\mathrm{i+td}}, \mathbf{BC}_{\mathrm{bu}}\}$.

The following theorem states that, for the canonical Boolean circuit representation of a set of clauses, **BC**-proofs can be simulated by *tree-like* resolution. This is fairly straightforward to establish from a well-known construction for reading a tree-like resolution refutation from a DPLL refutation, see e.g. [1].

**Theorem 4.** *There is a polynomial $p$ such that for any set of clauses $\varphi$, if there is a* **BC***-refutation for $C(\varphi)$ of size $n$, then there is a tree-like resolution refutation for $\varphi$ of size $p(n)$.*

Again, by the restricted nature of the variants of the **BC** method, we have the following corollary.

**Corollary 2.** *For each* $T \in \{\mathbf{BC}_{\mathrm{i}}, \mathbf{BC}_{\mathrm{td}}, \mathbf{BC}_{\mathrm{i+td}}, \mathbf{BC}_{\mathrm{bu}}, \mathbf{BC}_{\mathrm{bu+td}}\}$ *it holds that there is a polynomial $p$ such that for any set of clauses $\varphi$, if there is a $T$-refutation for $C(\varphi)$ of size $n$, then there is a tree-like resolution refutation for $\varphi$ of size $p(n)$.*

We note that tree-like resolution (and thus DPLL) and **BC** are equally efficient proof systems in the sense that (i) given a set of clauses $\varphi$, **BC** on $C(\varphi)$ can polynomially simulate tree-like resolution on $\varphi$ and, moreover, (ii) given a circuit $\mathcal{C}$ and the set of clauses $\varphi_{\mathcal{C}}$ obtained from $\mathcal{C}$ using Tseitin's translation, DPLL and thus tree-like resolution on $\varphi_{\mathcal{C}}$ can polynomially simulate **BC** on $\mathcal{C}$. The latter fact is discussed in more detail in Section 6 in which it is shown that the rules of **BC** match those of DPLL.

## 5 Relative Efficiency of Restricted Cuts

The main results of this paper are summarised in Figure 5 showing that there is no two-way polynomial simulation between the variations of the **BC** method. The results shed new light on the strength of proof systems of this kind, where the strength of the system is measured as the size of the minimal proofs producible for a given proposition. The results show that it is possible to increase the strength of a system significantly by extending the use of the cut rule in a controlled local manner w.r.t. the circuit structure. For example, moving from input cuts to bottom-up cuts can make a substantial difference in the sense that input cuts cannot polynomially simulate bottom-up cuts. If top-down cuts are additionally allowed, a similar substantial increase in the strength of the system is obtained. However, general cuts are still substantially stronger than any of the restricted variations considered. It should be noticed that the results obviously hold for circuits with additional Boolean functions if the set of rules involving and, or, and not gates remains unchanged. Furthermore, the results imply that the restrictions on the splitting rule in DPLL have the same effect on the proof complexity of CNF formulas obtained from Boolean circuits by using Tseitin's translation as will be discussed in Section 6.

The rest of this section is devoted to proofs of these results. The proofs rely on certain circuit families which are constructed from building blocks such as a Boolean circuit representation of the pigeon-hole principle. First we define the building blocks we call *gadgets* and then give the proofs of the main theorems 6–14. Combining these theorems, the resulting ordering of **BC** and its restricted variations based on the polynomial simulation relation, shown in Figure 5, is obtained by the transitivity of $\succeq$.

### 5.1 Gadget Constructions

We begin by defining the $\mathrm{PHP}_n^{n+1}$, $\mathrm{TD}_n$, $\mathrm{XOR}_n$, and UNSAT gadgets. They are used in constructing families of circuits which are used in proving the main theorems of this paper. Some lemmas involving properties of the gadgets are given.

**Pigeon-Hole Principle and the $\mathbf{PHP}_{n}^{n+1}$ Gadget** An example of a propositional formula with high proof complexity in many proof systems is the *pigeon-hole principle* $\mathrm{PHP}_n^m$, see e.g. [16]. The pigeon-hole principle states that there is no injective mapping from a finite $m$-element set into a finite $n$-element set if $m > n$ (that is, $m$ pigeons cannot sit in less than $m$ holes so that every pigeon has its own hole). In the following we consider the case $m = n+1$. As a set of clauses, we have

$$\mathrm{PHP}_n^{n+1} \stackrel{\mathrm{def}}{=} \bigcup_{1 \leq i \leq n+1} \{P_i\} \cup \bigcup_{1 \leq i < i' \leq n+1, 1 \leq j \leq n} \{H_{i,i'}^j\},$$

where the clauses $P_i$ and $H_{i,i'}^j$ are defined as $P_i \stackrel{\mathrm{def}}{=} \bigvee_{j=1}^n x_{i,j}$ and $H_{i,i'}^j \stackrel{\mathrm{def}}{=} (\neg x_{i,j} \vee \neg x_{i',j})$, and each $x_{i,j}$ is a Boolean variable with the interpretation "$x_{i,j} = \texttt{true}$ if and only if the $i^{\mathrm{th}}$ pigeon sits in the $j^{\mathrm{th}}$ hole". The $P_i$ clauses state that each

$$\begin{tikzcd}
 & & \mathbf{BC}_{\mathrm{bu}} \arrow[dr, phantom, "\succ" sloped] & \\
\mathbf{BC} \arrow[r, phantom, "\succ"] & \mathbf{BC}_{\mathrm{bu+td}} \arrow[ur, phantom, "\succ" sloped] \arrow[dr, phantom, "\succ" sloped] & \# & \mathbf{BC}_{\mathrm{i}} \\
 & & \mathbf{BC}_{\mathrm{i+td}} \arrow[ur, phantom, "\succ" sloped] \arrow[dr, phantom, "\succ" sloped] & \# \\
 & & & \mathbf{BC}_{\mathrm{td}}
\end{tikzcd}$$

**Fig. 5.** Summary of the ordering of **BC** and its restricted variations based on the polynomial simulation relation. The case $\mathbf{BC}_{\mathrm{bu}} \# \mathbf{BC}_{\mathrm{td}}$ is omitted from the picture for clarity.

pigeon has to sit in some hole, while clauses $H^j_{i,i'}$ state that no two pigeons can sit in the same hole. The union of all the clauses $P_i$ and $H^j_{i,i'}$ is obviously (by the pigeon-hole principle) unsatisfiable.

The canonical Boolean circuit representation of $\mathrm{PHP}^{n+1}_n$ is shown in part in Figure 6(a). We call $C(\mathrm{PHP}^{n+1}_n)$ the $\mathrm{PHP}^{n+1}_n$ *gadget*. Notice that as $\mathrm{PHP}^{n+1}_n$ is unsatisfiable, so is $C(\mathrm{PHP}^{n+1}_n)$. Formally the $\mathrm{PHP}^{n+1}_n$ gadget is the set of

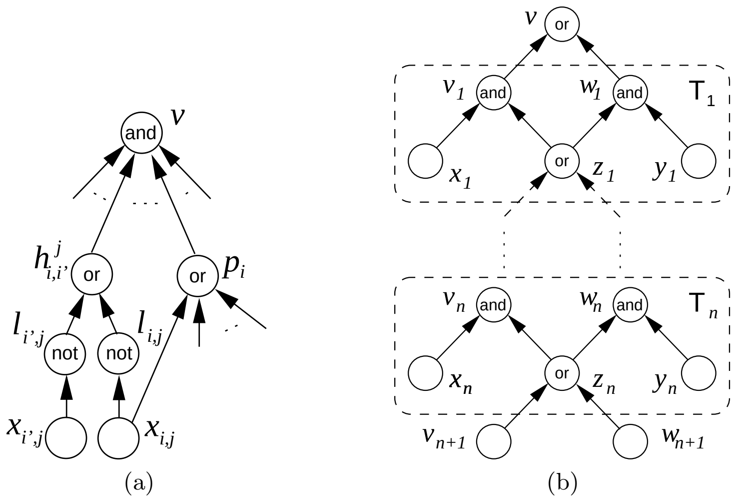


**Fig. 6.** (a) A part of the $\mathrm{PHP}^{n+1}_n$ gadget in detail, (b) the structure of the TD gadget.

equations

$$\begin{aligned} C(\mathrm{PHP}^{n+1}_n) = \; & \{v = \mathsf{and}(p_1, \ldots, p_{n+1}, h^1_{1,2}, \ldots, h^n_{n+1,n})\} \cup \\ & \{p_i = \mathsf{or}(x_{i,1}, \ldots, x_{i,n}) \mid 1 \le i \le n+1\} \cup \\ & \{h^j_{i,i'} = \mathsf{or}(l_{i,j}, l_{i',j}) \mid 1 \le i < i' \le n+1, 1 \le j \le n\} \cup \\ & \{l_{i,j} = \mathsf{not}(x_{i,j}) \mid 1 \le i \le n+1, 1 \le j \le n\}, \end{aligned}$$

where $h^1_{1,2}, \ldots, h^n_{n+1,n}$ stands for all $h^j_{i,i'}$, where $1 \le i < i' \le n+1$ and $1 \le j \le n$.

By the results in [15] we have the following theorem.

**Theorem 5.** *The size of the minimal resolution refutations for* $\mathrm{PHP}^{n+1}_n$ *is exponential w.r.t.* $n$.

Combining Theorem 4, Corollary 2, and Theorem 5, we have the following corollary.

**Corollary 3.** *For each* $T \in \{\mathbf{BC}_{\mathrm{i}}, \mathbf{BC}_{\mathrm{td}}, \mathbf{BC}_{\mathrm{i+td}}, \mathbf{BC}_{\mathrm{bu}}, \mathbf{BC}_{\mathrm{bu+td}}, \mathbf{BC}\}$*, the size of the minimal* $T$*-refutations for* $C(\mathrm{PHP}^{n+1}_n)$ *is exponential w.r.t.* $n$.

We use the pigeon-hole principle formulas because they are well-known and the family $\{\mathrm{PHP}_n^{n+1}\}$ has some nice features: (i) the size of each $\mathrm{PHP}_n^{n+1}$ CNF instance is polynomial w.r.t. $n$, and (ii) the size of the minimal resolution refutation for $\mathrm{PHP}_n^{n+1}$ is exponential w.r.t. $n$. Notice that other hard CNF formula families with similar properties could have been used instead.

**The TD Gadget** The structure of the $\mathrm{TD}_n$ *gadget* is shown in Figure 6(b). Formally the $\mathrm{TD}_n$ gadget is the set of equations

$$\begin{aligned}\mathrm{TD}_n = {} & \{v = \mathsf{or}(v_1, w_1)\} \cup \\ & \{v_i = \mathsf{and}(x_i, z_i) \mid 1 \le i \le n\} \cup \\ & \{w_i = \mathsf{and}(y_i, z_i) \mid 1 \le i \le n\} \cup \\ & \{z_i = \mathsf{or}(v_{i+1}, w_{i+1}) \mid 1 \le i \le n\}.\end{aligned}$$

The following lemma on the $\mathrm{TD}_n$ gadget will be useful in the proofs of our main results. This lemma derives from the fact that in order to have an entry for gate $z_n$ in every branch of a $\mathbf{BC}_{\mathrm{td}}$ tableau, one has to apply the cut rule on $v_i$ or $w_i$ for each $1 \le i \le n$, which leads to having an exponential number of entries in the tableau w.r.t. $n$.

**Lemma 1.** *Every $\mathbf{BC}_{\mathrm{td}}$-tableau for $\mathrm{TD}_n$ in which each branch has an entry for the gate $z_n$ is of exponential size w.r.t. $n$.*

*Proof.* The entry $\mathbf{T}v$ implies $\mathbf{T}v_1$ or $\mathbf{T}w_1$, but we cannot deduce one or the other. Thus we must apply the cut rule on either $v_1$ or $w_1$ in $\mathbf{BC}_{\mathrm{td}}$. Assume that we cut on $v_1$ (cutting on $w_1$ is symmetric). Now consider the branch in which we have $\mathbf{F}v_1$. Due to $v = \mathsf{or}(v_1, w_1)$ we must have $\mathbf{T}w_1$. Then from $w_1 = \mathsf{and}(y_1, z_1)$ we deduce $\mathbf{T}y_1$ and $\mathbf{T}z_1$ in the branch. In the branch where we have $\mathbf{T}v_1$ using the "down" rule for $\mathsf{and}$ we deduce $\mathbf{T}x_1$ and $\mathbf{T}z_1$. Nothing else can be deduced. Inductively on $i$, in order to have an entry for the gate $z_i$ in every branch of the tableau, the tableau must contain at least $2^i$ branches, all of which remain open. This is because we must for each $i$ apply the cut rule on either $v_i$ or $w_i$. This is demonstrated in Figure 7. Thus every tableau in which there is an entry for the gate $z_n$ in every branch of the tableau is of size at least in the order of $2^n$, that is, $2^{|V(\mathrm{TD}_n)|/5}$.

**The XOR Gadget** The Boolean $\mathsf{xor}$ function $\mathsf{xor}(x, y) = (x \wedge \neg y) \vee (\neg x \wedge y)$ evaluates to `true` if and only if exactly one of $x, y$ is `true`. Based on the $\mathsf{xor}$ function we can construct a Boolean circuit, as shown in Figure 8(a), for which it holds that the output gate $a_{i,j}$ evaluates to `true` if and only if $\mathsf{xor}(a_{i+1,i+2}, a_{i+1,j})$ evaluates to `true`. When we use this circuit construct as a part of a circuit, we represent it graphically as an "$\mathsf{xor}$ gate" $\oplus$.

Using the "$\mathsf{xor}$ gate" we construct a family of $\mathrm{XOR}_n$ gadgets as shown in Figure 8(b), having $n$ layers $\mathrm{X}_i$, $1 \le i \le n$, of $\mathsf{xor}$ gates. Formally, the $\mathrm{XOR}_n$

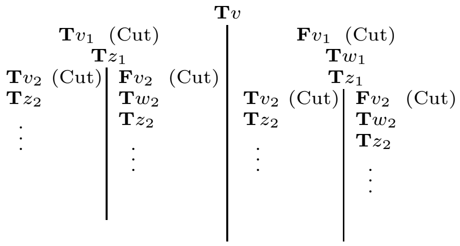


**Fig. 7.** Why $\mathbf{BC}_{\mathrm{td}}$-refutations for the circuit in Figure 6(b) are large.

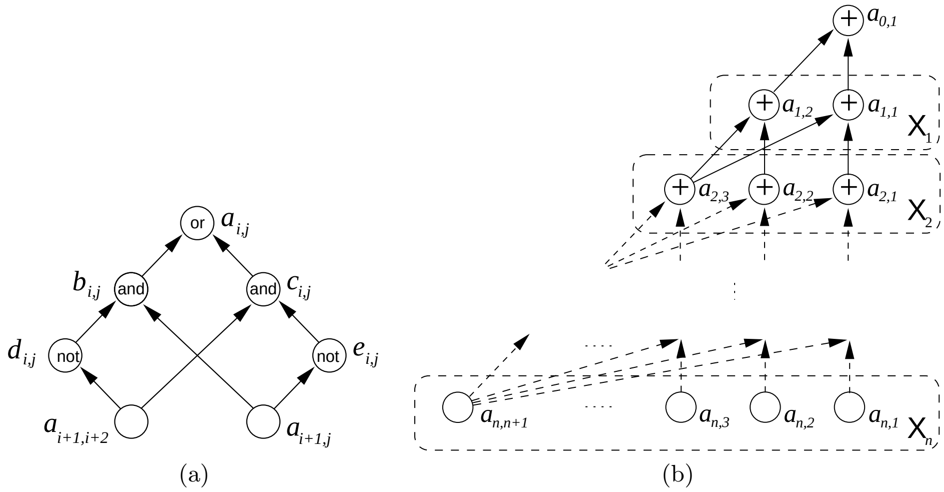


**Fig. 8.** (a) The Boolean function xor as a Boolean circuit, (b) The structure of the $\mathrm{XOR}_n$ gadget.

gadget is the set of equations

$$
\begin{aligned}
\mathrm{XOR}_n = \; & \{a_{i,j} = \mathsf{or}(b_{i,j}, c_{i,j}) \mid 0 \le i < n, 1 \le j \le i+1\} \cup \\
& \{b_{i,j} = \mathsf{and}(d_{i,j}, a_{i+1,j}) \mid 0 \le i < n, 1 \le j \le i+1\} \cup \\
& \{c_{i,j} = \mathsf{and}(e_{i,j}, a_{i+1,i+2}) \mid 0 \le i < n\} \cup \\
& \{d_{i,j} = \mathsf{not}(a_{i+1,i+2}) \mid 0 \le i < n\} \cup \\
& \{e_{i,j} = \mathsf{not}(a_{i+1,j}) \mid 0 \le i < n, 1 \le j \le i+1\}.
\end{aligned}
$$

The following two lemmas on the $\mathrm{XOR}_n$ gadget will be useful in the proofs of our main results. The lemmas derive from the property of the xor gate that, disregarding gates $b_{i,j}, c_{i,j}, d_{i,j}, e_{i,j}$, one cannot deduce an entry for one of $a_{i,j}, a_{i+1,j}, a_{i+1,i+2}$ without having an entry for both of the other two gates in the branch. This leads to the observation that in order to reach the output gate with bottom-up cuts or an input gate with top-down cuts one must apply the cut rule a number of times linear in $n$, and thus to having an exponential number of entries in the tableau w.r.t. $n$.

**Lemma 2.** *Every* $\mathbf{BC}_{\text{bu}}$*-tableau for* $\text{XOR}_n$ *in which each branch has an entry for the gate* $a_{1,1}$ *or* $a_{1,2}$ *is of exponential size w.r.t.* $n$*.*

*Proof.* We show by induction on $n$ that every $\mathbf{BC}_{\text{bu}}$-tableau for $\text{XOR}_n$ in which each branch has an entry for the gate $a_{1,1}$ or $a_{1,2}$ is of size at least $2^n$.

For $n = 1$, we must use the cut rule first on $a_{1,2}$ or $a_{1,1}$. Thus the branch has 1 application of the cut rule and the whole tableau is at least of size $2^1$.

Now assume that the lemma holds for $n$. Suppose that for $\text{XOR}_{n+1}$ there is a $\mathbf{BC}_{\text{bu}}$-tableau of size less than $2^{n+1}$ in which there is in each branch an entry for $a_{1,1}$ or $a_{1,2}$. In such a tableau, we must use the cut rule first on one of the gates $a_{n+1,j}$, $1 \leq j \leq n+2$. Due to the xor-nature of the $a$ gates and the use of bottom-up cuts, no gate of form $a_{k,l}$, $k \leq n$, can have a deduced entry before we have an entry for the gate $a_{n+1,n+2}$ (produced either (i) by a cut on $a_{n+1,n+2}$, or (ii) by a cut on an undetermined gate $b_{n,j}$ or $c_{n,j}$ such that a cut was already made on $a_{n+1,j}$, forcing an entry for $a_{n+1,n+2}$, too). Therefore, we can assume that the first cut was actually made on $a_{n+1,n+2}$. Because of the assumption that the size of the tableau is less than $2^{n+1}$, one of the sub-tableaux below this cut has size less than $2^n$. In such a sub-tableau, if the gate $a_{n+1,n+2}$ has an entry $\mathbf{F}a_{n+1,n+2}$, the gate $a_{n,m}$ has an entry $\mathbf{F}a_{n,m}$ ($\mathbf{T}a_{n,m}$) if and only if the gate $a_{n+1,m}$ has an entry $\mathbf{F}a_{n+1,m}$ ($\mathbf{T}a_{n+1,m}$). Similarly, if the gate $a_{n+1,n+2}$ has an entry $\mathbf{T}a_{n+1,n+2}$, the gate $a_{n,m}$ has an entry $\mathbf{F}a_{n,m}$ ($\mathbf{T}a_{n,m}$) if and only if the gate $a_{n+1,m}$ has an entry $\mathbf{T}a_{n+1,m}$ ($\mathbf{F}a_{n+1,m}$). Therefore, if we have made a cut on a gate $a_{n+1,m}$ in the sub-tableau, we could have equivalently made a cut on $a_{n,m}$. By replacing each such cut on $a_{n+1,m}$ with a cut on $a_{n,m}$ and removing other entries on gates not appearing in $\text{XOR}_n$, we transform the sub-tableau for $\text{XOR}_{n+1}$ to a $\mathbf{BC}_{\text{bu}}$-tableau for $\text{XOR}_n$ having size less than $2^n$. But this contradicts the induction hypothesis and thus both sub-tableaux must have size at least $2^n$ and the whole $\mathbf{BC}_{\text{bu}}$-tableau for $\text{XOR}_{n+1}$ is of size at least in the order of $2^{n+1}$.

**Lemma 3.** *Every* $\mathbf{BC}_{\text{td}}$*-tableau for* $\text{XOR}_n$ *in which each branch has an entry for some gate* $a_{n,i}$*,* $1 \leq i \leq n+1$*, is of exponential size w.r.t.* $n$*.*

*Proof.* As shown in Figures 9 and 10, in order to deduce an entry for $a_{i+1,j}$ or $a_{i+1,i+2}$ from an entry for $a_{i,j}$, one has to apply the cut on one of the gates in the xor gate. This causes branching, while no branches can be closed.

By branching on $b_{0,1}$ (or symmetrically on $c_{0,1}$), we can thus deduce entries for both $a_{1,1}$ and $a_{1,2}$ in both branches. Thus in each branch we may continue as in either of Figures 9 or 10. Notice that after branching we have an entry for $a_{i+1,i+2}$ in every branch. In addition to entries for all $a_{i,j}$, where $1 \leq j \leq i+1$, this is enough to deduce entries for all $a_{i+1,j}$. Still, for every $i$, we must apply the cut on some gate on level $i$ to deduce entries for the gates $a_{i+1,j}$, $1 \leq j \leq i+2$, doubling the number of open branches for each $i$. Thus every tableau in which

1. $\mathbf{F}a_{i,j}$
2. $\mathbf{F}b_{i,j}$
3. $\mathbf{F}c_{i,j}$

| | | | |
|---|---|---|---|
| 4. $\mathbf{T}a_{i+1,j}$ | (Cut) | 5. $\mathbf{F}a_{i+1,j}$ | (Cut) |
| 6. $\mathbf{F}d_{i,j}$ | $(2,4)$ | 9. $\mathbf{T}e_{i,j}$ | (5) |
| 7. $\mathbf{T}a_{i+1,i+2}$ | (6) | 10. $\mathbf{F}a_{i+1,i+2}$ | $(3,9)$ |
| 8. $\mathbf{F}e_{i,j}$ | (4) | 11. $\mathbf{T}d_{i,j}$ | (10) |

**Fig. 9.** A top-down sub-tableau for the xor circuit with an $\mathbf{F}a_{i,j}$ entry.

1. $\mathbf{T}a_{i,j}$

| | | | |
|---|---|---|---|
| 2. $\mathbf{T}b_{i,j}$ | (Cut) | 3. $\mathbf{F}b_{i,j}$ | (Cut) |
| 4. $\mathbf{T}d_{i,j}$ | (2) | 9. $\mathbf{T}c_{i,j}$ | $(1,3)$ |
| 5. $\mathbf{T}a_{i+1,j}$ | (2) | 10. $\mathbf{T}a_{i+1,i+2}$ | (9) |
| 6. $\mathbf{F}a_{i+1,i+2}$ | (4) | 11. $\mathbf{T}e_{i,j}$ | (9) |
| 7. $\mathbf{F}c_{i,j}$ | (6) | 12. $\mathbf{F}a_{i+1,j}$ | (11) |
| 8. $\mathbf{F}e_{i,j}$ | (5) | 13. $\mathbf{F}d_{i,j}$ | (10) |

**Fig. 10.** A top-down sub-tableau for the xor circuit with a $\mathbf{T}a_{i,j}$ entry.

there is in every branch an entry for some gate $a_{n,i}$, $1 \leq i \leq n+1$, is of size at least in the order of $2^n$, that is, $2^{\sqrt{|V(\mathrm{XOR}_n)|}}$.

### 5.2 $\mathbf{BC_{bu}}$ vs $\mathbf{BC_i}$

We now show that $\mathbf{BC}_\mathrm{i}$ cannot polynomially simulate $\mathbf{BC}_\mathrm{bu}$. The proof utilises the UNSAT (i.e., the circuit shown in Figure 1) and $\mathrm{PHP}_n^{n+1}$ gadgets.

**Lemma 4.** *There is an infinite family $\{\mathcal{C}_n\}$ of circuits such that (i) the size of $\mathcal{C}_n$ is $\mathcal{O}(n^3)$, and (ii) there is a $\mathbf{BC}_\mathrm{bu}$-refutation for $\mathcal{C}_n$ of constant size while any minimal $\mathbf{BC}_\mathrm{i}$-refutation for $\mathcal{C}_n$ is of size exponential in $n$.*

*Proof.* Consider the family of circuits of the type shown in Figure 11(a). Any circuit in the family is obviously unsatisfiable. For an arbitrary $n$, for $\mathbf{BC}_\mathrm{bu}$ we can construct a constant size refutation as follows. First, deduce $\mathbf{T}e$, $\mathbf{T}f$, $\mathbf{T}g$, $\mathbf{T}h$ from $\mathbf{T}v$. Then apply (say, in the $\mathrm{PHP}_n^{n+1}$ gadget on the left) the cut rule first on one of the input gates $x_{i,j}$, and deduce an entry for $l_{i,j}$. After this, apply the cut rule on $h^j_{i,i'}$ in both of the induced branches. Now we have induced four branches in total, having in each branch a constant number of entries, and can apply the cut rule on $a$ in each branch. After having an entry on $a$, each branch can be closed in a constant number of steps similarly to the refutation shown in Figure 3. Thus the generated closed tableau is of constant size.

Notice that to generate a refutation we need to reach the UNSAT gadget, i.e., it is impossible to generate a contradiction in all the branches of a tableau without having an entry for some of the gates in the UNSAT gadget in the tableau.

Now consider $\mathbf{BC}_{\mathrm{i}}$. From $\mathbf{T}v$ we can deduce $\mathbf{T}e$, $\mathbf{T}f$, $\mathbf{T}g$, and $\mathbf{T}h$, but nothing else. As $\mathrm{PHP}_n^{n+1}$ is unsatisfiable, it is impossible to deduce $\mathbf{T}a$ or $\mathbf{T}b$ with "up" rules. Thus we can only have $\mathbf{F}a$ and $\mathbf{F}b$ entries in any branch. In addition, a closed tableau can only be achieved after deducing an entry for gate $a$ or gate $b$. Thus we must have either $\mathbf{F}a$ or $\mathbf{F}b$ in every branch. But if we have $\mathbf{F}a$ or $\mathbf{F}b$ in every branch, then we effectively have a $\mathbf{BC}_{\mathrm{i}}$-refutation for $C(\mathrm{PHP}_n^{n+1})$. By Corollary 3, the size of such a refutation must be exponential in $n$.

**Theorem 6.** $\mathbf{BC}_{\mathrm{bu}} \succ \mathbf{BC}_{\mathrm{i}}$.

*Proof.* Obviously, $\mathbf{BC}_{\mathrm{bu}} \succeq \mathbf{BC}_{\mathrm{i}}$. By Lemma 4, there is a family $\{\mathcal{C}_n\}$ of circuits for which $\mathbf{BC}_{\mathrm{bu}}$ has constant size refutations while the minimal $\mathbf{BC}_{\mathrm{i}}$-refutations are of exponential size w.r.t. the circuit index $n$. As the size of the circuit $\mathcal{C}_n$ is $\mathcal{O}(n^3)$, the minimal $\mathbf{BC}_{\mathrm{i}}$-refutations are of size super-polynomial in the size of the circuit. Based on these facts, $\mathbf{BC}_{\mathrm{i}} \succeq \mathbf{BC}_{\mathrm{bu}}$ cannot hold.

### 5.3 $\mathbf{BC}_{\mathbf{i+td}}$ vs $\mathbf{BC}_{\mathbf{i}}$

We now proceed to show that $\mathbf{BC}_{\mathrm{i}}$ cannot polynomially simulate $\mathbf{BC}_{\mathrm{i+td}}$. This follows directly from the following lemma. In the proof, we re-use the ideas employed in the proof of Lemma 4.

**Lemma 5.** *There is an infinite family $\{\mathcal{C}_n\}$ of circuits such that (i) the size of $\mathcal{C}_n$ is $\mathcal{O}(n^3)$, and (ii) there is a $\mathbf{BC}_{\mathrm{i+td}}$-refutation for $\mathcal{C}_n$ of constant size while any minimal $\mathbf{BC}_{\mathrm{i}}$-refutation for $\mathcal{C}_n$ is of exponential size w.r.t. $n$.*

*Proof.* Consider again the family of circuits of the type shown in Figure 11(a). We have that any circuit in the family is unsatisfiable. For $\mathbf{BC}_{\mathrm{i+td}}$ we can construct a constant size refutation by first deducing $\mathbf{T}e$, then applying the cut rule on gate $a$, and then closing each branch similarly to the refutation shown in Figure 3. For $\mathbf{BC}_{\mathrm{i}}$, all $\mathbf{BC}_{\mathrm{i}}$-refutations will be of exponential size w.r.t. $n$ as argued in the proof of Lemma 4.

**Theorem 7.** $\mathbf{BC}_{\mathrm{i+td}} \succ \mathbf{BC}_{\mathrm{i}}$.

### 5.4 $\mathbf{BC}_{\mathbf{i+td}}$ vs $\mathbf{BC}_{\mathbf{td}}$

Next we show that $\mathbf{BC}_{\mathrm{td}}$ cannot polynomially simulate $\mathbf{BC}_{\mathrm{i+td}}$ by establishing the following lemma. The proof is based on a circuit constructed from two UNSAT gadgets and a $\mathrm{TD}_n$ gadget.

**Lemma 6.** *There is an infinite family $\{\mathcal{C}_n\}$ of circuits such that (i) the size of $\mathcal{C}_n$ is $\mathcal{O}(n)$, and (ii) there is a $\mathbf{BC}_{\mathrm{i+td}}$-refutation for $\mathcal{C}_n$ of linear size while any minimal $\mathbf{BC}_{\mathrm{td}}$-refutation for $\mathcal{C}_n$ is of exponential size w.r.t. $n$.*

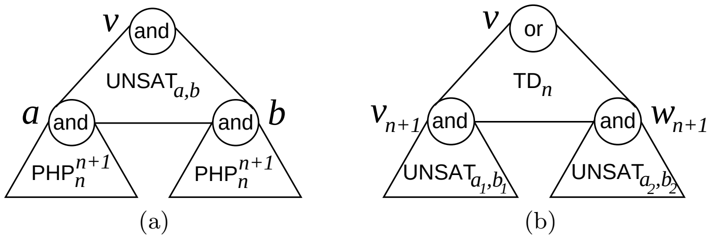


**Fig. 11.** (a) Circuit for Theorems 6 and 7. (b) Circuit for Theorem 8.

*Proof.* Consider the family of circuits of the type shown in Figure 11(b). Any circuit in this family is obviously unsatisfiable. For $\mathbf{BC}_{\mathrm{i+td}}$ we can construct a refutation of linear size w.r.t. $n$ as follows. First apply consecutively the cut rule on gates $a_1, b_1, a_2, b_2$ in each branch. This induces 16 branches, a constant number. We then have an entry for each of $a_1, b_1, a_2, b_2$ in every branch. Now we can deduce an entry for $v_{n+1}$ and $w_{n+1}$ in each branch. As $C(\mathrm{UNSAT}_{a,b})$ is unsatisfiable, we can only deduce $\mathbf{F}v_{n+1}$ and $\mathbf{F}w_{n+1}$. This can clearly be done in a constant number of steps. From the entries $\mathbf{F}v_{n+1}$, $\mathbf{F}w_{n+1}$ we can then deduce $\mathbf{F}z_n$, and then $\mathbf{F}v_n$, $\mathbf{F}w_n$. Proceeding recursively, we can thus deduce $\mathbf{F}v$ in every branch with a linear number of steps w.r.t. $n$.

Notice that to generate a refutation we need to reach the UNSAT gadgets, as in the proof of Lemma 4. But by Lemma 1, before reaching the gate $z_n$ top-down, we already must have generated a tableau with exponentially many entries w.r.t. $n$. Every $\mathbf{BC}_{\mathrm{td}}$-refutation is thus of exponential size w.r.t. $n$ for this family of circuits.

**Theorem 8.** $\mathbf{BC}_{\mathrm{i+td}} \succ \mathbf{BC}_{\mathrm{td}}$.

### 5.5 $\mathbf{BC}_{\mathrm{bu+td}}$ vs $\mathbf{BC}_{\mathrm{i+td}}$

In this subsection we show that $\mathbf{BC}_{\mathrm{i+td}}$ cannot polynomially simulate $\mathbf{BC}_{\mathrm{bu+td}}$. Using the ideas in the proof of Lemma 4, we construct a circuit from three circuits similar to the one employed in the proofs of Lemmas 4 and 5, an $\mathrm{XOR}_n$ gadget, and an expander sub-circuit that connects the former four. The expander circuit is an example of a simple nontrivial circuit in which deduction can be propagated through the circuit in a straightforward fashion. It is applied here so that trivial simplification of the circuit is not possible. Lemma 3 is also applied.

**Lemma 7.** *There is an infinite family $\{\mathcal{C}_n\}$ of circuits such that (i) the size of $\mathcal{C}_n$ is $\mathcal{O}(n^3)$, and (ii) there is a $\mathbf{BC}_{\mathrm{bu+td}}$-refutation for $\mathcal{C}_n$ of size $\mathcal{O}(n^2)$ while any minimal $\mathbf{BC}_{\mathrm{i+td}}$-refutation for $\mathcal{C}_n$ is of exponential size w.r.t. $n$.*

*Proof.* Consider the family of circuits of the type shown in Figure 12. Any circuit in the family is unsatisfiable. For $\mathbf{BC}_{\mathrm{bu+td}}$ we can construct a refutation

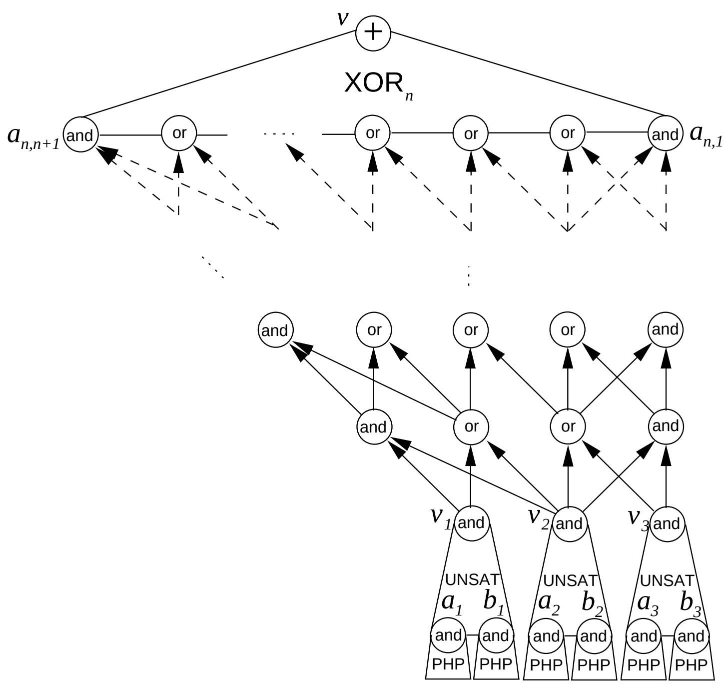


**Fig. 12.** Circuit for Theorem 9.

of polynomial size w.r.t. $n$ as follows. First apply the bottom-up strategy introduced in the proof of Lemma 4 to work through the $\mathrm{PHP}_n^{n+1}$ gadgets and to generate entries for the and gates $a_1, a_2, a_3, b_1, b_2, b_3$ in every branch. As in the proof of Lemma 4, this can be done having a constant number of entries in the tableau. Then it is straightforward to deduce entries for $v_1, v_2, v_3$ in each branch. Furthermore, as $C(\mathrm{UNSAT}_{a,b})$ is unsatisfiable, we must then have $\mathbf{F}v_1, \mathbf{F}v_2, \mathbf{F}v_3$ in every branch. Now in an arbitrary branch, it is straightforward to deduce the entries $\mathbf{F}a_{n,j}$ for all $1 \leq j \leq n+1$, generating only a number of entries in the order of $n^2$. Continuing on, generating only a number of entries in the order of $n^2$, deducing recursively $\mathbf{F}a_{i-1,j}$ from $\mathbf{F}a_{i,j}$ and $\mathbf{F}a_{i,i+1}$ we can at last deduce $\mathbf{F}v$. As we have in total a constant number of branches and $\mathcal{O}(n^2)$ entries in each branch, we clearly have a $\mathbf{BC}_{\mathrm{bu+td}}$-refutation of size $\mathcal{O}(n^2)$.

Again, to generate a refutation we need to reach the UNSAT gadgets. With input cuts, this results in a refutation of exponential size w.r.t. $n$, as argued in the proof of Lemma 4. By Lemma 3, any top-down approach will also result in a refutation of exponential size w.r.t. $n$.

**Theorem 9.** $\mathbf{BC}_{\mathrm{bu+td}} \succ \mathbf{BC}_{\mathrm{i+td}}$.

### 5.6 $\mathbf{BC_{bu+td}}$ vs $\mathbf{BC_{bu}}$

Next we show that $\mathbf{BC}_{\mathrm{bu}}$ cannot polynomially simulate $\mathbf{BC}_{\mathrm{bu+td}}$. In addition to ideas employed in the proof of Lemma 5, we use a circuit constructed from a pair of $\mathrm{XOR}_n$ gadgets and an UNSAT gadget, and apply Lemma 2.

**Lemma 8.** *There is an infinite family $\{\mathcal{C}_n\}$ of circuits such that (i) the size of $\mathcal{C}_n$ is $\mathcal{O}(n^2)$, and (ii) there is a $\mathbf{BC}_{\mathrm{bu+td}}$-refutation for $\mathcal{C}_n$ of constant size while any minimal $\mathbf{BC}_{\mathrm{bu}}$-refutation for $\mathcal{C}_n$ is of exponential size w.r.t. $n$.*

*Proof.* Consider the family of circuits of the type shown in Figure 13(a). As $C(\mathrm{UNSAT}_{a,b})$ is unsatisfiable, any circuit in this family is also unsatisfiable.

As already described in the proof of Lemma 5, for $\mathbf{BC}_{\mathrm{bu+td}}$ we can construct a constant size refutation top-down by first deducing $\mathbf{T}e$, then applying the cut rule on gate $a$, and closing each branch similarly to the refutation shown in Figure 3.

It is impossible to generate a refutation without reaching the UNSAT gadgets, as in the previous proofs in which we had an UNSAT gadget as a part of the circuit. By Lemma 2, in order to reach the UNSAT gadget, we must generate a tableau with exponential number of branches w.r.t. $n$. Thus any $\mathbf{BC}_{\mathrm{bu}}$-refutation for any circuit in this family must be of exponential size w.r.t. $n$.

**Theorem 10.** $\mathbf{BC}_{\mathrm{bu+td}} \succ \mathbf{BC}_{\mathrm{bu}}$.

### 5.7 BC vs $\mathbf{BC_{bu+td}}$

Now we proceed by showing that $\mathbf{BC}_{\mathrm{bu+td}}$ cannot polynomially simulate $\mathbf{BC}$. The proof uses $n+1$ UNSAT gadgets and $2n+3$ $\mathrm{XOR}_n$ gadgets, and applies Lemmas 2 and 3.

**Lemma 9.** *There is an infinite family $\{\mathcal{C}_n\}$ of circuits such that (i) the size of $\mathcal{C}_n$ is $\mathcal{O}(n^3)$, and (ii) there is a $\mathbf{BC}$-refutation for $\mathcal{C}_n$ of size $\mathcal{O}(n^2)$ while any minimal $\mathbf{BC}_{\mathrm{bu+td}}$-refutation for $\mathcal{C}_n$ is of exponential size w.r.t. $n$.*

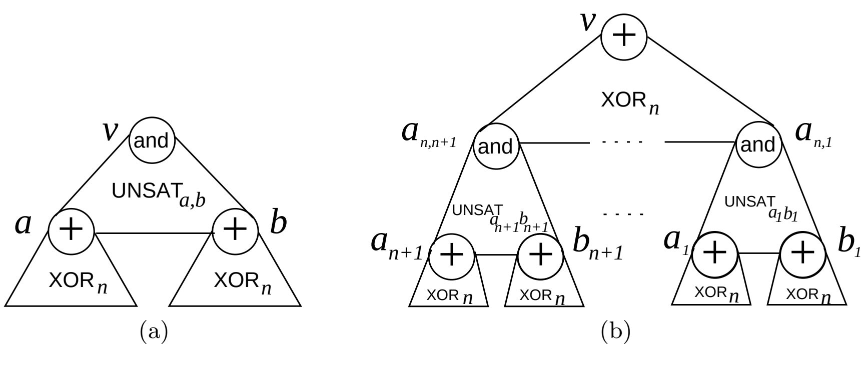


**Fig. 13.** (a) Circuit for Theorem 10. (b) Circuit for Theorem 11.

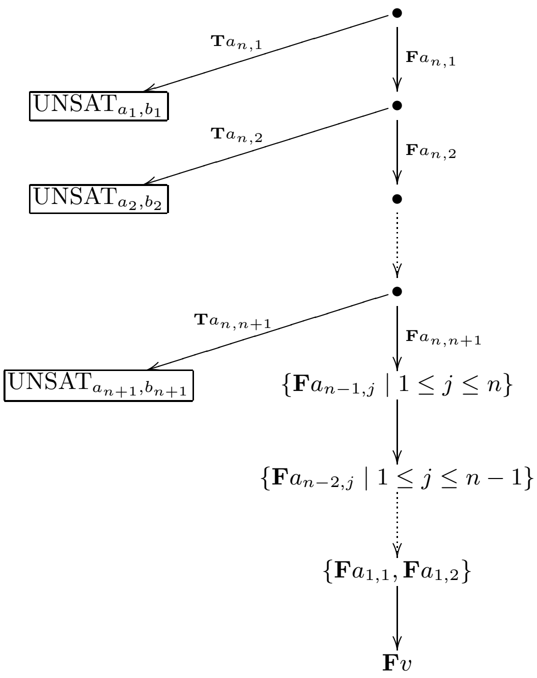


**Fig. 14.** How to generate a polynomial size **BC**-refutation for the circuit shown in Figure 13(b).

*Proof.* Consider the family of circuits of the type shown in Figure 13(b). For **BC** we can construct a refutation of polynomial size w.r.t. $n$ as follows. First apply the cut rule on $a_{n,1}$. In the branch in which we have $\mathbf{T}a_{n,1}$, apply the cut rule on $a_1$. Similarly to the refutation in Figure 3, we can close both the branch in which we have $\mathbf{T}a_1$ and the one in which we have $\mathbf{F}a_1$. In the branch in which we have $\mathbf{F}a_{n,1}$, recursively on $i$, cut first on $a_{n,i}$ and then in the branch in which we have $\mathbf{T}a_{n,i}$, cut on $a_i$ and again close both of the induced branches. This idea is shown in Figure 14. As the refutation in Figure 3 is of constant size, we end up with a tableau of linear size w.r.t. $n$ in which there is a single open branch with the entries $\mathbf{F}a_{n,i}$ for all $1 \leq i \leq n+1$. After this, generating only number of entries in the order of $n^2$, deducing recursively $\mathbf{F}a_{i-1,j}$ from $\mathbf{F}a_{i,j}$ and $\mathbf{F}a_{i,i+1}$ we can at last deduce $\mathbf{F}v$, thus generating a **BC**-refutation of size $\mathcal{O}(n^2)$.

Again, to generate a refutation we need to reach the UNSAT gadgets. By Lemma 2, any bottom-up approach will result in a refutation of exponential size w.r.t. $n$. By Lemma 3, this applies also for any top-down approach. Thus any $\mathbf{BC}_{\mathrm{bu+td}}$-refutation will be of exponential size w.r.t. $n$ for any circuit in this family.

**Theorem 11.** $\mathbf{BC} \succ \mathbf{BC}_{\mathrm{bu+td}}$.

### 5.8 $\mathbf{BC_i}$ vs $\mathbf{BC_{td}}$

We now turn to show that $\mathbf{BC}_\mathrm{i}$ and $\mathbf{BC}_\mathrm{td}$ are incomparable under the polynomial simulation relation. The proof draws heavily on the proofs of Lemmas 4 and 6.

**Theorem 12.** $\mathbf{BC}_\mathrm{i} \# \mathbf{BC}_\mathrm{td}$.

*Proof.* Consider again the family of circuits shown in Figure 11(a). In the proof of Lemma 4 it is shown that all $\mathbf{BC}_\mathrm{i}$-refutations for any circuit in this family are of exponential size w.r.t. $n$. For an idea of how to generate a $\mathbf{BC}_\mathrm{td}$-refutation of constant size we again refer the reader to the refutation shown in Figure 3.

On the other hand, consider the family $\{\mathcal{C}_n\}$ of circuits shown in Figure 11(b). By the proof of Lemma 6 any minimal $\mathbf{BC}_\mathrm{td}$-refutation for $\mathcal{C}_n$ is of exponential size w.r.t. $n$, while in the same proof it is described how to construct a linear size refutation for $\mathcal{C}_n$ by applying the cut rule only on input gates.

### 5.9 $\mathbf{BC_{bu}}$ vs $\mathbf{BC_{td}}$

Using ideas from the proof of Lemma 8 and Theorem 12, we show that $\mathbf{BC}_\mathrm{bu}$ and $\mathbf{BC}_\mathrm{td}$ are incomparable under the polynomial simulation relation.

**Theorem 13.** $\mathbf{BC}_\mathrm{bu} \# \mathbf{BC}_\mathrm{td}$.

*Proof.* By Theorem 12 $\mathbf{BC}_\mathrm{td}$ cannot polynomially simulate $\mathbf{BC}_\mathrm{i}$. As $\mathbf{BC}_\mathrm{bu} \succeq \mathbf{BC}_\mathrm{i}$, $\mathbf{BC}_\mathrm{td}$ cannot polynomially simulate $\mathbf{BC}_\mathrm{bu}$ either.

Consider the family $\{\mathcal{C}_n\}$ of circuits shown in Figure 13(a). It holds that there is a $\mathbf{BC}_\mathrm{td}$-refutation of constant size for each $\mathcal{C}_n$, while any minimal $\mathbf{BC}_\mathrm{bu}$-refutation is of exponential size w.r.t. $n$ by the proof of Lemma 8.

### 5.10 $\mathbf{BC_{bu}}$ vs $\mathbf{BC_{i+td}}$

As the last one of the main theorems of this work, we argue that $\mathbf{BC}_\mathrm{bu}$ and $\mathbf{BC}_\mathrm{i+td}$ are incomparable under the polynomial simulation relation.

**Theorem 14.** $\mathbf{BC}_\mathrm{bu} \# \mathbf{BC}_\mathrm{i+td}$.

*Proof.* By Theorem 13, $\mathbf{BC}_\mathrm{bu}$ cannot polynomially simulate $\mathbf{BC}_\mathrm{td}$. As $\mathbf{BC}_\mathrm{i+td} \succeq \mathbf{BC}_\mathrm{td}$, $\mathbf{BC}_\mathrm{bu}$ cannot polynomially simulate $\mathbf{BC}_\mathrm{i+td}$ either.

On the other hand, consider the family $\{\mathcal{C}_n\}$ of circuits shown in Figure 15. Notice that a circuit $\mathcal{C}_n$ consists of a $\mathrm{TD}_n$ gadget from the input gates of which hang two sub-circuits equivalent to the circuit in Figure 11(a). Combining Lemma 1 and the reasoning presented in the proof of Lemma 4, we have that every $\mathbf{BC}_\mathrm{i+td}$-refutation for an arbitrary circuit in this family is of exponential size w.r.t. $n$, as it is impossible to reach the UNSAT gadgets using top-down and input cuts without generating an exponential number of entries in the tableau w.r.t. $n$. For $\mathbf{BC}_\mathrm{bu}$, we can generate a refutation of linear size w.r.t. $n$ as follows. It is discussed in the proof of Lemma 4 how one can apply the cut on gate $a$ in the circuit in Figure 11(a). What we can do here is to cut through the $\mathrm{PHP}_n^{n+1}$

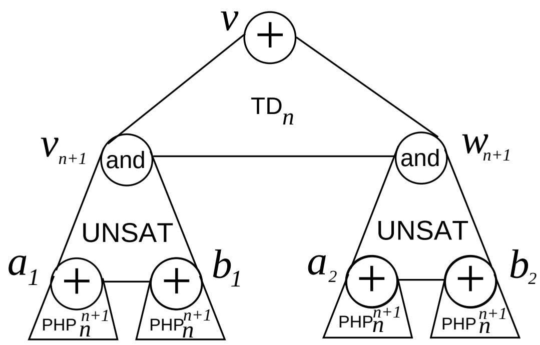


**Fig. 15.** Circuit for Theorem 14.

circuits similarly as in the proof of Lemma 4. Then we can apply the cut rule on each gate $a_1, b_1, a_2, b_2$ in each branch. After this, it is straightforward to deduce an entry for gates $v_{n+1}$ and $w_{n+1}$ in every branch. Due to the unsatisfiability of $C(\mathrm{UNSAT}_{a,b})$, with this bottom-up approach it is only possible to deduce $\mathbf{F}v_{n+1}, \mathbf{F}w_{n+1}$. At this point we note that as the UNSAT gadget with $\mathrm{PHP}_n^{n+1}$ gadgets hanging from the input gates has a constant number of gates, we have so far obviously generated a tableau with constant number of entries only. Having $\mathbf{F}v_{n+1}, \mathbf{F}w_{n+1}$ in each branch, it is possible to deduce $\mathbf{F}v$ by generating only a linear number of entries w.r.t. $n$, as explained in the proof of Lemma 6. Thus we can generate a $\mathbf{BC}_{\mathrm{bu}}$-refutation of linear size w.r.t. $n$ for any member of the family of circuits considered.

## 6 Relevance to DPLL

Each Boolean circuit $\mathcal{C}$ can be translated into a propositional formula in CNF of linear size w.r.t. the size of $\mathcal{C}$ so that the formula is satisfiable if and only if $\mathcal{C}$ is. Tseitin's translation [30] is a standard approach, introducing a new variable $v_g$ for each gate $g$ in $\mathcal{C}$ and capturing the functional dependencies in $\mathcal{C}$ by clauses. The translation is summarised in Table 1. Clearly, it is linear in the number of gates and edges in $\mathcal{C}$. The output CNF formula is the union of the sets of clauses produced by the translation. We now argue that the main results of this work apply to DPLL (without learning or non-chronological backtracking) in the case that the input is in CNF translated from a circuit using Tseitin's translation.

We assume that the reader is familiar with the basic DPLL. A DPLL-refutation can be abstractly seen as a tableau in which the entries are sets of clauses obtained by unit propagation and splitting. A branch in a tableau is contradictory if there are both of the unit clauses $(a)$ and $(\neg a)$ in the branch for some variable $a$. The rest of the terminology concerning DPLL-tableaux is synonymous in an obvious way with that of $\mathbf{BC}$-tableaux.

**Table 1.** Tseitin's translation of a Boolean circuit to a set of clauses.

| Boolean circuit | clause set |
|---|---|
| $g$ is the output gate | $\{(v_g)\}$ |
| $g = \mathsf{not}(g_1)$ | $\{(v_g \vee v_{g_1}), (\neg v_g \vee \neg v_{g_1})\}$ |
| $g = \mathsf{or}(g_1, \ldots g_k)$ | $\{(v_{g_1} \vee \cdots \vee v_{g_k} \vee \neg v_g)\} \cup \bigcup_{i=1}^{k} \{(v_g \vee \neg v_{g_i})\}$ |
| $g = \mathsf{and}(g_1, \ldots g_k)$ | $\{(\neg v_{g_1} \vee \cdots \vee \neg v_{g_k} \vee v_g)\} \cup \bigcup_{i=1}^{k} \{(\neg v_g \vee v_{g_i})\}$ |

For the following, let $\varphi$ be a set of clauses that is obtained from a Boolean circuit using Tseitin's translation. It holds that DPLL for $\varphi$ can polynomially simulate **BC** for the original circuit, and vice versa. In fact, any DPLL-refutation can be interpreted as a **BC**-refutation, and vice versa. Especially, we argue that unit clauses $(v_g)$ $((\neg v_g))$ in a DPLL-refutation correspond exactly to entries $\mathbf{T}g$ $(\mathbf{F}g)$ of the corresponding **BC**-refutation.

Obviously, the splitting rule in DPLL is equivalent to the cut rule in **BC**; adding the unit clause $(v_g)$ $((\neg v_g))$, is equivalent to extending a branch with $\mathbf{T}g$ $(\mathbf{F}g)$ by applying the cut rule, and vice versa. It is also straightforward to see that the unit propagation rule in DPLL and the deterministic tableau rules in Figure 2(b)–(h) in **BC** have equivalent deduction power. To demonstrate this, we consider the two following example cases.

Consider the last undetermined child rule for `and` gates. Assume a branch in which (i) a gate $g = \mathsf{and}(g_1, \ldots, g_k)$ is constrained to `false` by having the entry $\mathbf{F}g$ and (ii) all $g_1, \ldots, g_{k-1}$ are constrained to true by having the entries $\mathbf{T}g_1, \ldots, \mathbf{T}g_{k-1}$. One can now deduce $\mathbf{F}g_k$ by applying the rule. This deduction step can be simulated in the corresponding DPLL-tableau branch because the branch has unit clauses $(\neg v_g)$, $(v_{g_1})$,..., and $(v_{g_{k-1}})$. Thus the clause $(\neg v_{g_1} \vee \cdots \vee \neg v_{g_k} \vee v_g)$ resulting from Tseitin's translation of `and` gates can be transformed into the unit clause $(\neg v_{g_k})$ by applying unit propagation.

On the other hand, assume that it is possible to generate the unit clause $(\neg v_g)$ in a branch of the DPLL-tableau by unit propagating on an original clause of form $(v_{g_1} \vee \cdots \vee v_{g_k} \vee \neg v_g)$. This means that the branch must contain the unit clauses $(\neg v_{g_1})$,...,$(\neg v_{g_k})$. Thus the corresponding **BC**-tableau branch has the entries $\mathbf{F}g_1$,...,$\mathbf{F}g_k$ and the entry $\mathbf{F}g$ can be deduced by applying an "up" rule on the gate $g = \mathsf{or}(g_1, \ldots g_k)$ that was translated to have the clause $(v_{g_1} \vee \cdots \vee v_{g_k} \vee \neg v_g)$ in the CNF formula.

## 7 Conclusion

This work addresses the question of how restrictions on the use of the cut rule affect proof complexity in Boolean circuit satisfiability checking based on tableaux. The tableau method in question consists of a complete and sound subset of the rules in the method introduced in [18]. The results show that the methods obtained by the cut restrictions considered (any combination of input, top-down, and bottom-up cuts) cannot polynomially simulate the unrestricted method. Moreover, for each pair of restricted methods, there exist a family of circuits

$\{\mathcal{C}_n\}$ for which the sizes of the minimal proofs differ exponentially w.r.t. $n$ between the methods.

The introduced tableau method is a non-clausal generalisation of the Davis-Putnam-Logemann-Loveland method for CNF formulas. The results show that DPLL with locality based cut restrictions, such as splitting on the input gates only, cannot polynomially simulate the DPLL method with an unrestricted splitting rule. This, in turn, contradicts a common belief based on empirical results [28, 12] that for CNF formulas obtained from a circuit (or formula) representation, a significant gain in efficiency is obtained if splitting is restricted to variables corresponding to input gates (or variables in the original formula).

The results suggest a number of interesting topics of further research. They indicate that good cut heuristics, i.e. general methods for choosing gates on which the cut rule is applied, can have significant impact on efficiency. The total number of gates in a circuit can be enormous compared to the number of input gates. Hence, restricting to input cuts seems to be a computationally attractive alternative. However, our results show that allowing even slightly more general cuts can lead to significant savings. The key research question is how to limit the subset of gates on which to apply the cut, and how to choose a good cut among the candidates so that the attractive computational properties are preserved. In order to evaluate empirically the theoretical results of the paper such new cut heuristics for Circuit Sat need to be developed and implemented. Secondly, modern Sat solvers employ a number of search space pruning techniques, like one-step lookahead, equivalence reasoning, cone-of-influence (see e.g. [18]) as well as non-chronological backtracking and learning schemes (see e.g. [31, 23, 29]). An interesting question is how proof complexity is affected by these techniques.

## References


1. Paul Beame, Richard Karp, Toniann Pitassi, and Michael Saks. The efficiency of resolution and Davis-Putnam procedures. *SIAM Journal on Computing*, 31(4):1048–1075, 2002.
2. Paul Beame and Toniann Pitassi. Propositional proof complexity: Past, present, and future. *Bulletin of the European Association for Theoretical Computer Science*, 65:66–89, 1998.
3. Armin Biere and Wolfgang Kunz. SAT and ATPG: Boolean engines for formal hardware verification. In *Proceedings of 20th IEEE/ACM International Conference on Computer Aided Design*, pages 782–785. IEEE Press, 2002.
4. Per Bjesse, Tim Leonard, and Abdel Mokkedem. Finding bugs in an Alpha microprocessor using satisfiability solvers. In *Proceedings of the 13th International Conference of Computer-Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 454–464. Springer, 2001.
5. Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
6. Marcello D'Agostino, Dov M. Gabbay, Reiner Hähnle, and Joachim Posegga, editors. *Handbook of Tableau Methods*. Kluwer Academic Publishers, 1999.

7. Marcello D'Agostino and Marco Mondadori. The taming of the cut: Classical refutations with analytic cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.
8. Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for $k$-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.
9. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.
10. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.
11. M.K. Ganai, Lintao Zhang, P. Ashar, A. Gupta, and S. Malik. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *Proceedings of the 39th Conference on Design Automation*, pages 747–750. ACM, 2002.
12. Enrico Giunchiglia, Alessandro Massarotto, and Roberto Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proceedings of the 15th National Conference on Artificial Intelligence and of the 10th Conference on Innovative Applications of Artificial Intelligence*, pages 948–953. AAAI Press, 1998.
13. Enrico Giunchiglia and Roberto Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas. In *Proceedings of the Italian National Conference on Artificial Intelligence*, pages 84–94. Springer, 2000.
14. Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the satisfiability (SAT) problem: a survey. In Dingzhu Du, Jun Gu, and Panos M. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 19–152. AMS, 1997.
15. Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2–3):297–308, 1985.
16. Stasys Jukna. *Extremal Combinatorics: with Applications in Computer Science.* Springer-Verlag, 2001.
17. Tommi A. Junttila. BCSat 0.3 – a satisfiability checker for Boolean circuits. Computer program, 2001. Available at `http://www.tcs.hut.fi/Software/`.
18. Tommi A. Junttila and Ilkka Niemelä. Towards an efficient tableau method for Boolean circuit satisfiability checking. In *Computational Logic – CL 2000; First International Conference*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 553–567, London, UK, 2000. Springer-Verlag.
19. Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359 – 363, 1992.
20. Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 1194–1201, 1996.
21. Andreas Kuehlmann, Malay K. Ganai, and Viresh Paruthi. Circuit-based boolean reasoning. In *Proceedings of the 38th Conference on Design Automation*, pages 232–237. ACM, 2001.
22. Tracy Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):6–22, January 1992.
23. João P. Marques-Silva and Karem A. Sakallah. GRASP: A new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*, pages 220–227, 1997.

24. Joo P. Marques-Silva and Lus Guerra e Silva. Solving satisfiability in combinational circuits. *IEEE Design & Test of Computers*, 20(4):16–21, 2003.
25. F. Massacci. Simplification — a general constraint propagation technique for propositional and modal tableaux. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 217–231. Springer, 1998.
26. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
27. David A. Plaisted and Steven A. Greenbaum. A structure–preserving clause form translation. *Journal of Symbolic Computation*, 2:193–304, 1986.
28. Ofer Shtrichman. Tuning SAT checkers for Bounded Model Checking. In *Computer Aided Verification – CAV 2000; 12th International Conference*, volume 1855 of *Lecture Notes in Computer Science*, pages 480–494. Springer-Verlag, 2000.
29. Christian Thiffault, Fahiem Bacchus, and Toby Walsh. Solving non-clausal formulas with DPLL search. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pages 663–678. Springer, 2004.
30. Grigori S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, 1983.
31. Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in boolean satisfiability solver. In *Proceedings of the International Conference on Computer Aided Design*, pages 279–285. IEEE Computer Society, 2001.
32. Lintao Zhang and Sharad Malik. The quest for efficient Boolean satisfiability solvers. In *Automated Deduction – CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 295–313. Springer-Verlag, 2002.