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

Matti Järvisalo, Tommi A. Junttila, and Ilkka Niemelä

Laboratory for Theoretical Computer Science
Helsinki University of Technology
{matti.jarvisalo,tommi.junttila,ilkka.niemela}@hut.fi

**Abstract.** This paper studies the relative proof complexity of variations of a tableau method for Boolean circuit satisfiability checking obtained by restricting the use of the cut rule in several natural ways. The results show that the unrestricted cut rule can be exponentially more effective than any of the considered restrictions. Moreover, there are exponential differences between the restricted versions, too. The results also apply to the Davis-Putnam procedure for conjunctive normal form formulae obtained from Boolean circuits with a standard linear size translation.

## 1 Introduction

Propositional satisfiability checkers have been applied successfully to many interesting domains such as planning [10] and model checking of finite state systems [2, 1]. The success builds on recent significant advances in the performance of SAT checkers based both on stochastic local search algorithms and on complete systematic search, see e.g. [14, 11].

Most successful satisfiability checkers assume that the input formulae 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 formulae. However, using CNF makes efficient modeling of an application cumbersome. Therefore one usually employs a more general formula representation and then transforms the formula into CNF by using a standard translation. This translation introduces a new variable for each Boolean connective in the formula, resulting in a linear size CNF translation.

In this paper we study satisfiability checking methods for Boolean circuits. Boolean circuits are interesting because they allow for a compact and natural representation in many domains as the representation can be simplified by sharing common subexpressions and by preserving natural structures and concepts of the domain. A tableau method that works directly with Boolean circuits has been developed [9]. It can be seen as a lifting of the Davis-Putnam procedure for CNF to Boolean circuits. Instead of standard (cut free) tableau techniques it employs a direct cut rule combined with deterministic (non-branching) deduction rules. The aim is to achieve high performance and to avoid some computational problems in cut free tableaux [5].

The efficiency of a typical Davis-Putnam method based SAT checking system depends on (i) the applied search space pruning techniques (e.g. non-branching deduction rules, non-chronological backtracking and conflict-driven learning) and (ii) the splitting rule (i.e., on which gates the explicit cut is applied). In this paper, we focus on the splitting/cut rule. The research problem is: How do restrictions on the use of the cut rule effect the proof complexity in Boolean circuit satisfiability

[*]The financial support from Academy of Finland (project #53695) is gratefully acknowledged.

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 [15, 6]. However, our results show that doing so can, in the worst case, result in exponentially larger proofs compared to the unrestricted cut rule. In addition to the input gate restricted cuts, we study several other natural locality based restrictions of the cut rule, e.g., "top-down" cuts that are made only on the children of the already determined gates and "bottom-up" cuts that can be applied on input gates and on the parents of the already determined gates. Our results show that restricting the cut in any of the considered ways can result in exponentially larger proofs than by applying the unrestricted cut. In addition, we show that there are also exponential differences in the proof complexity between the restricted versions of the cut rule. Note that although this paper considers Boolean circuits and a tableau method for them in order to preserve and exploit the structure of the problem, the results directly apply to SAT checkers for CNF formulae in the case the formulae are obtained from circuits by using the standard linear size translation. This is because the Boolean propagation induced by the applied tableau rules corresponds to the propagation induced by the unit-literal rule of the Davis-Putnam method for the clauses generated by the translation.

## 2 Boolean Circuits

A *Boolean circuit* (see e.g. [12]) is an acyclic directed graph in which the nodes are called *gates*. The gates can be divided into three categories[1]: (i) *output gates* 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 with each output and intermediate gate. An example of a Boolean circuit is shown in Figure 1. In this circuit, $a$ and $b$ are input gates, $c, d, e, f, g$ and $h$ intermediate gates, and $v$ an output gate.

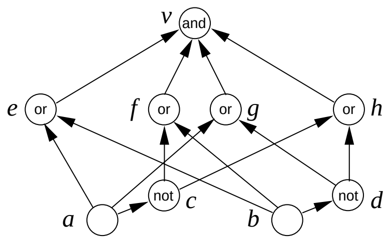


**Figure 1:** A Boolean circuit.

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 in the set of equations, each $v \in \mathcal{V}$ has at most one equation and that the equations are non-recursive. 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. For a Boolean circuit $\mathcal{C}$ with the set of gates $\mathcal{V}$, we denote the set of gates appearing in $\mathcal{C}$ by $V(\mathcal{C})$. The edge relation for a Boolean circuit $\mathcal{C}$ is defined as $E(\mathcal{C}) = \{\langle v, v' \rangle \mid v' = f(\ldots, v, \ldots) \in \mathcal{C}\}$.

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 *constrained Boolean circuit* $\langle \mathcal{C}, c^+, c^- \rangle$ is a Boolean circuit $\mathcal{C}$ with the restrictions that the gates in $c^+ \subseteq V(\mathcal{C})$ are $\mathtt{true}$ and those in $c^- \subseteq V(\mathcal{C})$ are $\mathtt{false}$. Here, we are interested in the *satisfaction problem* for constrained Boolean circuits: given a constrained Boolean circuit, is there a consistent truth assignment that respects the constraints? If such a consistent assignment exists, it is called a *satisfying truth assignment* and the circuit is *satisfiable*. Otherwise the circuit is *unsatisfiable*. The satisfaction problem for constrained Boolean circuits is obviously $\mathbf{NP}$-complete.

[1]We do not consider circuits in which there are trivial gates with no edges (neither incoming nor outgoing).

For simplicity, we consider the class of Boolean circuits in which the following three types of Boolean functions are allowed: (1) $\mathsf{not}(v) = \mathtt{true}$ iff $v$ is $\mathtt{false}$; (2) $\mathsf{or}(v_1, \ldots, v_k) = \mathtt{true}$ iff at least one $v_i$, $1 \le i \le k$, $k \ge 2$, is $\mathtt{true}$; and (3) $\mathsf{and}(v_1, \ldots, v_k) = \mathtt{true}$ iff all $v_i$, $1 \le i \le k$, $k \ge 2$, are $\mathtt{true}$. It is straightforward to extend this class with additional Boolean functions such as xor and equivalence.

## 3 A Tableau Method for Boolean Circuits

We concentrate on a tableau method for Boolean circuit satisfiability checking we call $\mathbf{BC}$. The $\mathbf{BC}$ method consists of the rules shown in Figure 2. It is a simplified version of the method introduced in [9], where rules e.g. for xor and equivalence are provided.

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

(a) The explicit cut rule

$$\frac{v = \top}{\mathbf{T}v} \qquad \frac{v = \bot}{\mathbf{F}v}$$

(b) Constant rules

$$\frac{\begin{array}{c} v = \mathsf{not}(v_1) \\ \mathbf{F}v_1 \end{array}}{\mathbf{T}v} \qquad \frac{\begin{array}{c} v = \mathsf{not}(v_1) \\ \mathbf{T}v_1 \end{array}}{\mathbf{F}v}$$

(c) "Up" rules for not

$$\frac{\begin{array}{c} v = \mathsf{not}(v_1) \\ \mathbf{F}v \end{array}}{\mathbf{T}v_1} \qquad \frac{\begin{array}{c} v = \mathsf{not}(v_1) \\ \mathbf{T}v \end{array}}{\mathbf{F}v_1}$$

(d) "Down" rules for not

$$\frac{\begin{array}{c} v = \mathsf{or}(v_1, \ldots, v_k) \\ \mathbf{F}v_1, \ldots, \mathbf{F}v_k \end{array}}{\mathbf{F}v} \qquad \frac{\begin{array}{c} v = \mathsf{or}(v_1, \ldots, v_k) \\ \mathbf{T}v_i, i \in \{1, \ldots, k\} \end{array}}{\mathbf{T}v}$$

"(e) Up" rules for or

$$\frac{\begin{array}{c} v = \mathsf{and}(v_1, \ldots, v_k) \\ \mathbf{T}v_1, \ldots, \mathbf{T}v_k \end{array}}{\mathbf{T}v} \qquad \frac{\begin{array}{c} v = \mathsf{and}(v_1, \ldots, v_k) \\ \mathbf{F}v_i, i \in \{1, \ldots, k\} \end{array}}{\mathbf{F}v}$$

(g) "Up" rules for and

$$\frac{\begin{array}{c} v = \mathsf{or}(v_1, \ldots, v_k) \\ \mathbf{F}v \end{array}}{\mathbf{F}v_1, \ldots, \mathbf{F}v_k}$$

(f) "Down" rule for or

$$\frac{\begin{array}{c} v = \mathsf{and}(v_1, \ldots, v_k) \\ \mathbf{T}v \end{array}}{\mathbf{T}v_1, \ldots, \mathbf{T}v_k}$$

(h) "Down" rule for and

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

(i) "Last undetermined child" rules for or and and

Figure 2: Tableau method $\mathbf{BC}$ for Boolean circuits.

Given a constrained Boolean circuit $\langle \mathcal{C}, c^+, c^- \rangle$, a $\mathbf{BC}$-tableau for it is a tree such that the root of the tree consists of the equations in $\mathcal{C}$ and the constraints; for each gate $v \in c^+$, a $\mathbf{T}v$ entry is added, while for each $v \in c^-$, a $\mathbf{F}v$ entry is added. The other nodes in the tree are entries of the form $\mathbf{T}v$ or $\mathbf{F}v$, where $v \in V(\mathcal{C})$. The entries are generated by applying the rules in Figure 2 as in the standard tableau method [4].

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)–(i). A tableau is *finished* if all the branches of the tableau are complete. A finished tableau is *closed* if all of its branches are contradictory.

For each $v \in V(\mathcal{C})$, we say that the entry $\mathbf{T}v$ ($\mathbf{F}v$) can be *deduced* in the branch if the entry $\mathbf{T}v$ ($\mathbf{F}v$) can be generated by applying rules (b)–(i) only. A closed $\mathbf{BC}$-tableau for a constrained circuit is called a $\mathbf{BC}$-*refutation* for the circuit. As an example, for the circuit shown in Figure 1 with the constraint that the output gate $v$ is $\mathtt{true}$, a $\mathbf{BC}$-refutation is shown in Figure 3.

$$\begin{array}{ll}
\hline
1. & v = \mathsf{and}(e, f, g, h) \\
2. & e = \mathsf{or}(a, b) \\
3. & f = \mathsf{or}(b, c) \\
4. & g = \mathsf{or}(a, d) \\
5. & h = \mathsf{or}(c, d) \\
6. & c = \mathsf{not}(a) \\
7. & d = \mathsf{not}(b) \\
8. & \mathbf{T}v \\
\hline
\end{array}$$

$$\begin{array}{lll}
9. & \mathbf{T}e & (1, 8) \\
10. & \mathbf{T}f & (1, 8) \\
11. & \mathbf{T}g & (1, 8) \\
12. & \mathbf{T}h & (1, 8)
\end{array}$$

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

**Figure 3:** A $\mathbf{BC}$-refutation.

We study variations of $\mathbf{BC}$ in which we restrict the use of the explicit cut rule to certain types of gates. Let $\langle \mathcal{C}, c^+, c^- \rangle$ be a constrained Boolean circuit. The considered variations of $\mathbf{BC}$ are the following.

- $\mathbf{BC}_{\rm i}$: Use of explicit cut is restricted to input gates (we call such cuts *input cuts*).
- $\mathbf{BC}_{\rm td}$: Use of explicit cut is restricted to output gates and those gates $v$ for which there exists a $\mathbf{T}v'$ or a $\mathbf{F}v'$ entry in the branch and $\langle v, v' \rangle \in E(\mathcal{C})$ (*top-down cuts*).
- $\mathbf{BC}_{\rm bu}$: Use of explicit cut is restricted to input cuts and gates $v$ for which there exists a $\mathbf{T}v'$ or a $\mathbf{F}v'$ entry in the branch and $\langle v', v \rangle \in E(\mathcal{C})$ (*bottom-up cuts*).
- $\mathbf{BC}_{\rm i+td}$: Use of explicit cut is restricted to input and top-down cuts.
- $\mathbf{BC}_{\rm bu+td}$: Use of explicit cut is restricted to bottom-up and top-down cuts.

The $\mathbf{BC}$ method and its variations are complete and sound proof systems for constrained Boolean circuits in the sense that a complete open branch in a tableau for a circuit gives a satisfying truth assignment for the circuit, and a closed tableau indicates that the circuit is unsatisfiable.

## 4 Proof Complexity and the Pigeon-Hole Principle

We use the notion of $p$-*simulation* [3] to study the relative efficiency of proof systems. Let $T$ be a proof system. The *proof complexity* (or *complexity* in short) of a proposition $x$ in $T$ is the minimum of $|P|$, where $P$ is a $T$-proof of $x$ and $|P|$ the *size*[2] of $P$. For any two proof systems $T$ and $T'$, we say that $T$ $p$-*simulates* $T'$, denoted by $T \succeq T'$, if there exists a polynomial $p(n)$ such that, for any $x$, if there exists $T'$-proof for $x$ of size $n$, then there exists a $T$-proof for $x$ of size $p(n)$. 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 $T \not\equiv T'$.

An example of a propositional formula with high proof complexity in many proof systems is the *pigeon-hole principle* $\mathrm{PHP}_n^m$ [8]. It can be formalized as a set of clauses as follows:

$$\mathrm{PHP}_n^m \stackrel{\text{def}}{=} \bigcup_{1 \le i \le m} \{P_i\} \cup \bigcup_{\substack{1 \le i \ne i' \le m, \\ 1 \le j \le n}} \{H_{i,i'}^j\},$$

where the clauses $P_i$ and $H_{i,i'}^j$ are defined as $P_i = \bigvee_{j=1}^n x_{i,j}$ and $H_{i,i'}^j = \neg x_{i,j} \vee \neg x_{i',j}$, and each $x_{i,j}$ is a Boolean variable $x_{i,j}$ with the interpretation "$x_{i,j} = \texttt{true}$ if and only if the $i^{\rm th}$ pigeon sits in the $j^{\rm th}$ hole". When $m > n$, $\mathrm{PHP}_n^m$ is obviously unsatisfiable.
For resolution [13], it was first proven by Haken [7] that the proof complexity of $\mathrm{PHP}_n^{n+1}$ is exponential w.r.t. $n$.
We define the size of a refutation in $\mathbf{BC}$ and its variations as the number of nodes in the closed tableau. As an example, the size of the $\mathbf{BC}$-refutation shown in Figure 3 is 14. The obvious ordering of $\mathbf{BC}$ and its variations based on the $p$-simulation relation is shown in Figure 4.

$$\begin{tikzcd}
 & \mathbf{BC}_{\rm bu} \arrow[dr, phantom, sloped, "\succeq"] & \\
\mathbf{BC} \succeq \mathbf{BC}_{\rm bu+td} \arrow[ur, phantom, sloped, "\succeq"] \arrow[dr, phantom, sloped, "\succeq"] & & \mathbf{BC}_{\rm i} \arrow[dl, phantom, sloped, "\succeq"] \\
 & \mathbf{BC}_{\rm i+td} \arrow[dr, phantom, sloped, "\succeq"] & \\
 & & \mathbf{BC}_{\rm td}
\end{tikzcd}$$

**Figure 4:** Obvious ordering of variations of $\mathbf{BC}$.

We denote the *canonical Boolean circuit representation*[3] of a set of clauses $\varphi$ by $C(\varphi)$. As an example, for $\mathrm{UNSAT}_{a,b} \stackrel{\text{def}}{=} \{a \vee b, a \vee \neg b, \neg a \vee b, \neg a \vee \neg b\}$, the circuit representation $C(\mathrm{UNSAT}_{a,b})$

[2]Defining the *size* of a proof depends highly on the type of a proof system considered.

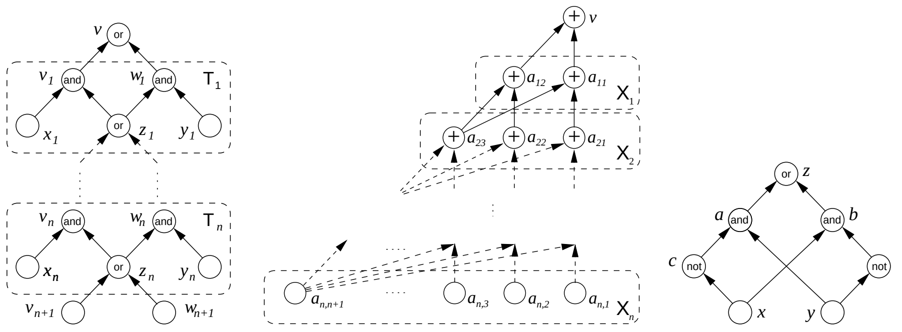
[3]The canonical representation is obvious; each variable induces an input gate, each negated variable and each clause an intermediate not/or gate, and the set of clauses itself an output and gate.

is shown in Figure 1. By $C_{\top}(\varphi)$ we denote $C(\varphi)$ with the constraint that the output gate of $C(\varphi)$ is `true`. It is not hard to show that, given any set of clauses $\varphi$ that is unsatisfiable and a $\mathbf{BC}$-refutation $T$ for $C_{\top}(\varphi)$ of size $t$, we can construct a tree-like resolution refutation for $\varphi$ of size $\mathcal{O}(t)$. Thus we have the following theorem, which obviously applies to the restricted variations of $\mathbf{BC}$ as well.

**Theorem 4.1** *The complexity of* $C_{\top}(\mathrm{PHP}_n^{n+1})$ *for* $\mathbf{BC}$ *is exponential w.r.t.* $n$.

## 5 Relative Efficiency of Restricted Cuts

In this section we prove negative $p$-simulation results between $\mathbf{BC}$ and its variations. First we define some circuit constructs we call *gadgets* that are used in the proofs. We call $C(\mathrm{UNSAT}_{a,b})$ shown in Figure 1 the UNSAT gadget. Similarly, we call $C(\mathrm{PHP}_n^{n+1})$ the $\mathrm{PHP}_n^{n+1}$ gadget. Obviously, $C_{\top}(\mathrm{UNSAT}_{a,b})$ and $C_{\top}(\mathrm{PHP}_n^{n+1})$ are unsatisfiable. The structure of the TD$_n$ gadget is shown in Figure 5(a), and the structure of the XOR$_n$ gadget in Figure 5(b). In the XOR$_n$ gadget we denote by $\oplus$ the construct shown in Figure 5(c), i.e., the logical xor function.



(a) (b) (c)

Figure 5: (a) The TD$_n$ gadget, (b) the XOR$_n$ gadget, and (c) the logical xor function as a Boolean circuit.

We now show that each pair of the $p$-simulation orderings of $\mathbf{BC}$ and its restricted variations, shown in Figure 4, is actually proper. The following theorems yield a complete classification of these systems based on relative proof complexity.

**Theorem 5.1** $\mathbf{BC}_{\mathrm{bu}} \succ \mathbf{BC}_{\mathrm{i}}$.

**Proof sketch.** Consider the family of circuits shown in Figure 6 with the output gate $v$ constrained to `true`. Any such circuit is obviously unsatisfiable. For $\mathbf{BC}_{\mathrm{bu}}$ we can construct a constant size refutation w.r.t. $n$ by applying the cut rule first on both of the input gates representing variables in some clause of type $H^{j}_{i,i'}$ in the leftmost $\mathrm{PHP}_n^{n+1}$ gadget and then on gate $a$, after which every branch can be closed as seen in Figure 3.

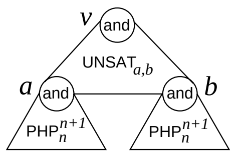


**Figure 6:** The circuit for Theorems 5.1 and 5.2.

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 refutation without having an entry for some of the gates in the UNSAT gadget in the tableau. From $\mathbf{T}v$ we can deduce $\mathbf{T}e$, $\mathbf{T}f$, $\mathbf{T}g$, and $\mathbf{T}h$, but nothing else. In $\mathbf{BC}_{\mathrm{i}}$, when applying the cut rule contradiction can be achieved only in the PHP part or by deducing $\mathbf{F}e$, $\mathbf{F}f$, $\mathbf{F}g$, or $\mathbf{F}h$ by starting from the input gates and propagating the deduction through the PHP part. As $\mathrm{PHP}_n^{n+1}$ is unsatisfiable, it is impossible to deduce $\mathbf{T}a$ or $\mathbf{T}b$. Now assume that there exists a $\mathbf{BC}_{\mathrm{i}}$-refutation of polynomial size w.r.t. $n$ for this circuit. Given such a refutation, it would be easy to construct a $\mathbf{BC}_{\mathrm{i}}$-refutation of polynomial size w.r.t. $n$ for $C_{\top}(\mathrm{PHP}_n^{n+1})$. This contradicts Theorem 4.1. □

**Theorem 5.2** $\mathbf{BC}_{\mathrm{i+td}} \succ \mathbf{BC}_{\mathrm{i}}$.

**Proof sketch.** Consider the family of circuits shown in Figure 6 with the constraint that the output gate $v$ is `true`, making any such circuit unsatisfiable. For $\mathbf{BC}_{\mathrm{i+td}}$ we can construct a constant size refutation by applying the cut rule on gate $e$ and then on gate $a$. For $\mathbf{BC}_{\mathrm{i}}$, all $\mathbf{BC}_{\mathrm{i}}$-refutations will be of exponential size w.r.t. $n$ as shown in the proof of Theorem 5.1. □

**Theorem 5.3** $\mathbf{BC}_{\mathrm{i+td}} \succ \mathbf{BC}_{\mathrm{td}}$.

**Proof sketch.** Consider the family of circuits shown in Figure 7 with the constraint that the output gate $v$ is `true`. Any circuit in the family is obviously unsatisfiable. For $\mathbf{BC}_{\mathrm{i+td}}$ we can construct a refutation of size $\mathcal{O}(n)$ by first applying the cut rule on gates $a_1, b_1, a_2$ and $b_2$. Then we can deduce $\mathbf{F}v_{n+1}$ and $\mathbf{F}w_{n+1}$ in every branch, after which we can deduce $\mathbf{F}v$ in a straightforward manner.

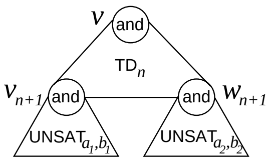


**Figure 7:** The circuit for Theorem 5.3.

Notice that to generate a refutation we need to reach the UNSAT gadgets, as in the proof of Theorem 5.1. $\mathbf{T}v$ 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 $v_1$. Consider the branch in which we have $\mathbf{F}v_1$ (notice the symmetry). 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 and we deduce $\mathbf{T}x_1$ and $\mathbf{T}z_1$. Nothing else can be deduced. By induction on $i$, we must use the cut rule on either $v_i$ or $w_i$. By increasing the number of TD gadgets $n$ by one, the number of times the cut rule is applied also increases by one, doubling the number of branches in the tableau. After reaching $z_n$, from $\mathbf{T}z_n$ we will have $\mathbf{T}v_{n+1}$ or $\mathbf{T}w_{n+1}$ in every branch. This leads to contradiction in one of the UNSAT gadgets (this is shown in Figure 3). Every $\mathbf{BC}_{\mathrm{td}}$-refutation will thus be of exponential size w.r.t. $n$. □

**Theorem 5.4** $\mathbf{BC}_{\mathrm{bu+td}} \succ \mathbf{BC}_{\mathrm{i+td}}$.

**Proof sketch.** Consider the family of circuits shown in Figure 8 with the constraint that the output gate $v$ is `true`, making any such circuit unsatisfiable. For $\mathbf{BC}_{\mathrm{bu+td}}$ we can construct a refutation of polynomial size w.r.t. $n$ by applying the cut rule first on both of the input gates representing variables in some clause of type $H^j_{i,i'}$ in each of the $\mathrm{PHP}_n^{n+1}$ gadgets, and then on either one of the gates $a_i$, $b_i$, where $1 \le i \le 3$, in each of the UNSAT gadgets.

Again, to generate a refutation we need to reach the UNSAT gadgets. With input cuts as the only cuts, this results in a refutation of exponential size w.r.t. $n$, as in the proof of Theorem 5.1. With top-down cuts as the only cuts, we are forced to apply the cut rule on at least one of the gates on every level of the $\mathrm{XOR}_n$ gadgets and inductively dou-

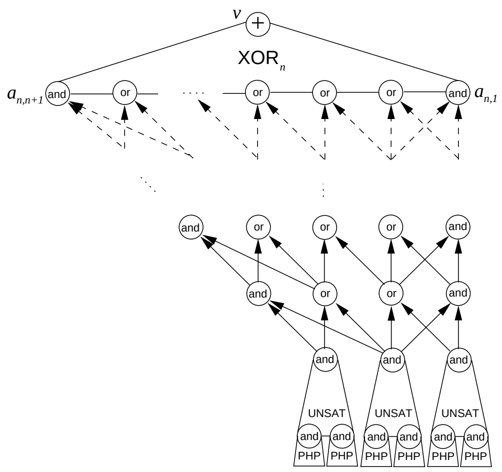


**Figure 8:** The circuit for Theorem 5.4.

ble the number of open branches on each level. This also results in a refutation of exponential size w.r.t. $n$. □

**Theorem 5.5** $\mathbf{BC}_{\mathrm{bu+td}} \succ \mathbf{BC}_{\mathrm{bu}}$.

**Proof sketch.** Consider the family of circuits shown in Figure 9 with the output gate $v$ constrained to `true`, making any such circuit unsatisfiable. For $\mathbf{BC}_{\mathrm{bu+td}}$ we can construct a constant size refutation by applying the cut rule top-down as described in the proof of Theorem 5.2.

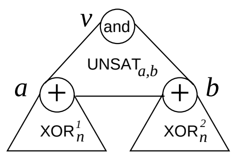


**Figure 9:** The circuit for Theorem 5.5.

It is impossible to generate a refutation without reaching the UNSAT gadgets, as in the previous proofs. With bottom-up cut as the only cuts, after applying the cut rule on a single $a_{n,j}$ we cannot deduce anything (by symmetry we only discuss one of the two XOR gadgets). Applying the cut rule on two gates on level $n$ we can deduce an entry for at most one gate on level $n - 1$. Inductively, applying the cut rule on $k$ gates on level $n$ we can deduce an entry for at most $k - 1$ gates on level $n - 1$. This approach results in a refutation of exponential size w.r.t. $n$.
After applying the cut rule on a single $a_{n,j}$ on level $n$, we could also apply the cut rule on any of the gates on level $n - 1$ that is the parent of $a_{n,j}$. Then we can deduce an entry for the other child of $a_{n-1,i}$, but nothing else. Now we could again apply the cut on any of the gates $a_{n-2,k}$ on level $n - 2$ that is the parent of $a_{n-1,i}$. Again, we can deduce an entry for the other child of $a_{n-2,k}$, but nothing else. Inductively, this approach also results in a refutation of exponential size w.r.t. $n$.
All other approaches to climbing up the levels of the XOR gadget are combinations of these two approaches, and will also results in refutations of exponential size w.r.t. $n$. □

**Theorem 5.6** $\mathbf{BC} \succ \mathbf{BC}_{\mathrm{bu+td}}$.

**Proof sketch.** Consider the family of circuits shown in Figure 10 with the output gate $v$ constrained to `true`. Any such circuit is unsatisfiable. For $\mathbf{BC}$ we can construct a refutation of polynomial size w.r.t. $n$ by applying the cut rule on $a_{n,i}$, $a_i$, and $b_i$, $1 \leq i \leq n + 1$, inductively on $i$.
Again, to generate a refutation we need to reach the UNSAT gadgets. With bottom-up cuts as the only cuts this results in a refutation of exponential size w.r.t. $n$, as in the proof of Theorem 5.5. With top-down cuts, this yields an exponential size refutation w.r.t. $n$, similarly to the proof of Theorem 5.4. □

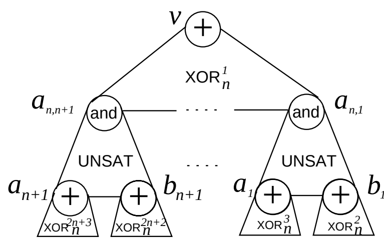


**Figure 10:** The circuit for Theorem 5.6.

Finally, we have the following theorem for which the proofs are omitted. In the theorem, (a) is established using the ideas in the proofs of Theorems 5.2 and 5.3, (b) results from the approach in the proofs of Theorems 5.3 and 5.5, and (c) from the approach in the proofs of Theorem 5.5 and from a circuit which consists of the circuit shown in Figure 7 with PHP gadgets hanging from all of the input gates of the UNSAT gadgets.

**Theorem 5.7** (a) $\mathbf{BC}_{\mathrm{i}} \not\equiv \mathbf{BC}_{\mathrm{td}}$. (b) $\mathbf{BC}_{\mathrm{bu}} \not\equiv \mathbf{BC}_{\mathrm{td}}$. (c) $\mathbf{BC}_{\mathrm{bu}} \not\equiv \mathbf{BC}_{\mathrm{i+td}}$.

By transitivity of $\succeq$, the resulting ordering of $\mathbf{BC}$ and its restricted variations based on the $p$-simulation relation is shown in Figure 11.

## 6 Conclusions

In this paper we study the relative proof complexity of several restricted versions of a tableau method for Boolean circuits. Especially, we consider the natural cases (and their combinations) in which (i) the cut rule can only be applied on the input gates or (ii) the cut rule is applied in a locality based manner either "top-down" starting from the constraints or "bottom-up" starting from the input gates. We show that the unrestricted use of the cut rule may yield exponentially smaller proofs than any of the restrictions considered. Moreover, there are exponential differences in the efficiency between the restricted versions, too. The results indicate that it is preferable to use less restricted cut rules in order to have as small proofs as possible. Obviously, the results hold for extended classes of Boolean circuits if the set of rules involving and, or, and not gates remains unchanged in the tableau method. The results also apply to the Davis-Putnam method for conjunctive normal form formulae that are obtained from Boolean circuits by the standard linear size translation. For instance, our results are in contradiction with a common intuition that it is, in general, beneficial to restrict the splittings in a Davis-Putnam procedure to the variables corresponding to the input gates only.

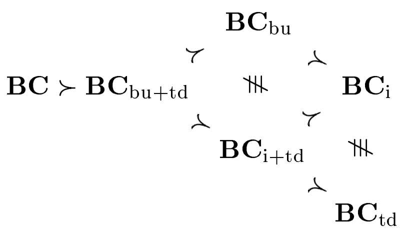

$\mathbf{BC}_{\mathrm{bu}}$
$\mathbf{BC} \succ \mathbf{BC}_{\mathrm{bu+td}}$
$\mathbf{BC}_{\mathrm{i}}$
$\mathbf{BC}_{\mathrm{i+td}}$
$\mathbf{BC}_{\mathrm{td}}$

**Figure 11:** Summary of $p$-simulation relations between $\mathbf{BC}$ and its restricted variations. The relation $\mathbf{BC}_{\mathrm{bu}} \not\equiv \mathbf{BC}_{\mathrm{td}}$ is omitted from the picture for clarity.

## References

[1] P. Bjesse, T. Leonard, and A. Mokkedem. Finding bugs in an Alpha microprocessor using satisfiability solvers. In *Proceedings of the 13th International Conference of Computer-Aided Verification*, volume 2102 of *LNCS*, pages 454–464. Springer, 2001.

[2] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, July 2001.

[3] S. A. Cook and R. A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

[4] M. D'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, editors. *Handbook of Tableau Methods*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[5] M. D'Agostino and M. Mondadori. The taming of the cut: Classical refutations with analytic cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[6] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 948–953. AAAI Press, 1998.

[7] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2–3):297–308, 1985.

[8] S. Jukna. *Extremal Combinatorics: with Applications in Computer Science*. Springer, 2001.

[9] T. A. Junttila and I. Niemelä. Towards an efficient tableau method for Boolean circuit satisfiability checking. In *Computational Logic – CL 2000*, volume 1861 of *LNAI*, pages 553–567. Springer, 2000.

[10] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence*, Portland, Oregon, July 1996.

[11] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, pages 530–535. ACM, 2001.

[12] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, USA, 1994.

[13] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[14] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:521–532, 1996.

[15] O. Shtrichman. Tuning SAT checkers for bounded model checking. In *Computer Aided Verification, CAV 2000*, volume 1855 of *LNCS*, pages 480–494. Springer, 2000.