To appear in, H. de Swarts, ed. *Proc. of the Internat. Conf. on Analytic Tableaux and Related Methods (TABLEAUX-98)*, LNAI, ©Springer Verlag. 1998

Simplification*

A general constraint propagation technique for propositional and modal tableaux

Fabio Massacci

Dipartimento di Informatica e Sistemistica,
Università di Roma "La Sapienza"
via Salaria 113, I-00198 Roma, Italy
massacci@dis.uniroma1.it

Abstract. Tableau and sequent calculi are the basis for most popular interactive theorem provers for formal verification. Yet, when it comes to automatic proof search, tableaux are often slower than Davis-Putnam, SAT procedures or other techniques. This is partly due to the absence of a bivalence principle (viz. the cut-rule) but there is another source of inefficiency: the lack of constraint propagation mechanisms.
This paper proposes an innovation in this direction: the rule of *simplification*, which plays for tableaux the role of subsumption for resolution and of unit for the Davis-Putnam procedure.
The simplicity and generality of simplification make possible its extension in a uniform way from propositional logic to a wide range of modal logics. This technique gives an unifying view of a number of tableaux-like calculi such as DPLL, KE, HARP, hyper-tableaux, BCP, KSAT.
We show its practical impact with experimental results for random 3-SAT and the industrial IFIP benchmarks for hardware verification.

1 Introduction

It is a widespread belief that methods based on the sequent calculus (such as tableaux) are hopeless for "real life" satisfiability and validity search. Even for decidable problems with a natural appeal for tableaux, such as modal or propositional logics, experimental results have shown that other algorithms outperform them by orders of magnitude [6, 12, 16, 27, 33].

The key question is whether such a gap is inherent to tableau methods or something is simply missing. The answer is not only of theoretical interest but has an extreme relevance for the development of formal verification tools.

Indeed, variants of the sequent calculus are the main techniques used by interactive theorem provers, such as Isabelle [22], PVS [26] or HOL [13]. Those provers have successfully tackled hardware and software verification and often require to prove some properties in decidable sub-theories such as propositional

* An comprehensive review of the various results on simplification, including the first order case can be found in [18].

logic (e.g. a N-bits binary adder) or fragments of arithmetics [26]. If tableaux are "hopeless by nature", then non sequent-based systems (outside the prover) should be consulted as oracles. For instance efficient algorithms for propositional logics as one-off "derived rules" for HOL have been proposed in [15].

To fill the computational gap, a number of works have improved the effectiveness of tableau method by switching from ground calculi to free variables versions [23, 11] with smart skolemisation techniques [1], adding ad-hoc rules for modus ponens and tollens [24, 7, 16, 21], imposing regularity conditions [7, 17], using controlled form of cut [17, 7], or factoring and merging [20, 32], exploiting universal variables [4, 3], incorporating features of hyper-resolution [2]. In some cases [6] the quest for an efficient implementation has lead from an original sequent calculus to a (seemingly different) Davis-Putnam procedure.

We propose a general technique, which subsumes a number of these approaches, and whose intuition is due to a comparison between tableaux and their "historical competitors".

Since the very beginning, the Davis-Putnam-Longeman-Loveland (DPLL for short) procedure [9, 8], and resolution [25] included rules for the simplification of the formulae to be proved unsatisfiable (e.g. unit or subsumption) *without* changing the basic inference mechanism.

The implementation of this simplification procedures is almost a research field in itself [31, 33] and difficult problems could hardly (if at all) be solved by either resolution or DPLL without using them. On the contrary, even in the modern texts on tableaux [11] everything is proved by "first principles".

We advocate that the lack of *rules for constraint propagation* is one of the main cause for the computational gap[1]. Hence we need a simple theoretical innovation: an operation that *plays for the sequent calculus the same role of unit for* DPLL *and subsumption for resolution*. We call it *simplification* and discuss its application to propositional, and modal logics.

The simplicity and flexibility of simplification makes it possible its application to a wide range of logics and logical formalism, as soon as there is sequent calculus. The only difficult feature is the enhancement of simplification as we "upgrade" the logic. Its effectiveness can enhance the computational power of interactive and automatic tableau based provers.

The introduction of simplification for the sequent calculus provides an *unifying perspective* of many (tableau based) deduction techniques and "explains away" the characteristic of the DPLL procedure (or KSAT-procedures of [12] which are based on DPLL) or KE. For instance the first order version in [8] can be "re-interpreted" as a tableau á la Smullyan "plus" propositional simplification. Such an interpretation can also explain the comparative inefficiency of first-order DPLL versus propositional DPLL (see further [18]).

We show the practical impact of the operation of simplification on the performance of tableau methods: a simple Prolog implementation shows essentially the same easy-hard-easy computation pattern for random 3-SAT shown by DPLL

[1] This has also been confirmed for modal logics by recent experimental studies [16] which clarified the gap between KSAT and tableau based procedures.

[6] even with a a simpler notion of simplification than the full fledged used by DPLL. An extensive experimental analysis has also been carried on the industrial IFIP (non clausal) benchmarks for hardware verification [5].

In the rest of the paper we present the intuitions behind the operation of simplification (2) and introduce some preliminary notation (§3). Next, we present the calculus for propositional logics (§4), show how other approaches are subsumed (§5) and present the extension to modal logics (§6). Finally, we discuss the experimental results (§7), and conclude (§8).

2 Principles and Intuitions

We assume a basic knowledge of the sequent calculus (see [28] for an introduction). In the sequel formulae are denoted by A, B, C, sets of formulae by Γ, Δ and sequents by $\Gamma \Longrightarrow \Delta$.

Simplification can be explained by comparing it with subsumption: bottom-up methods (resolution) explore the search space by generating new information and to simplify the search they *retain information* to discard (newly generated) irrelevant facts. Top-down methods (tableaux) work by breaking existing information and therefore the best way to delete irrelevant information should be *anticipating* the outcome of the search.

For example, consider the following α-rule:

$$\frac{\Gamma, A, B \Longrightarrow \Delta}{\Gamma, A \wedge B \Longrightarrow \Delta}\ (\alpha)$$

After this rule a tableau will continue to break down connectives until a branch closure rule can be applied. It looks for something like $\Gamma, C \Longrightarrow \Delta, C$.

Yet, after the α-step, there is some information that the calculus does not use to anticipate the outcome of its search: when all formulae in $\Gamma \cup \{A, B\}$ are true then A must be true, no matter what A is (equally for B). We can use this information about A to *simplify Γ and Δ before reducing other connectives.*

The simplest way is syntactic search: look for all exact[2] occurrences of A in Γ and replace it with the constant $\top$; next perform some boolean operations to eliminate $\top$ and $\bot$ from the formula; only afterwards continue with other rules.

The computational pay-off may seem doubtful: some time must be spent for scanning the formula replacing the occurrences with $\top$; eliminate boolean constant and so on. So what could be the gain?

Intuition 1. The sequent proof of $\Gamma \Longrightarrow \Delta$ is potentially of size $O(2^{|\Gamma \cup \Delta|})$ [30]. If can reduce $\Gamma \Longrightarrow \Delta$ to $\Gamma^* \Longrightarrow \Delta^*$ so that its size decreases, if only by 1, we would reduce the potential search space at least by half.

We trade off some polynomial processing for an exponential gain: scanning the formula and looking for "exact" copies can be done in polynomial time.

[2] With modal connectives or variables it is not so simple but this is just the intuition.

Hence, if there is at least another occurrence of A (or B) in Γ, it always pays off to replace this occurrence with $\top$: the resulting sequent will be smaller and thus the proof tree below will also be (exponentially) smaller.

We would like simplification to be

- *locally applicable* to single formulae;
- a *sound and complete* admissible rule for the underlying calculus;
- substantially a *rewriting rule*;
- a rule requiring *polynomial time*;
- such that *the resulting formulae are (potentially) smaller.*

Soundness is obvious. We need completeness only if we impose that simplification is applied before other rules. If rules can be applied in any order, incomplete simplification would be an "unsafe" rule over which we may backtrack [22].

Local applicability is also a key property and leads to rules like "simplify B using A". The advantage of this definition and the rules we propose is that when we "upgrade" the logic we do not have (in general) to change the rules but only to specify the behavior of the operation for the *new* connectives we have introduced. The flexibility and modularity makes it easy also to upgrade current implementations with these techniques.

3 Notation and Terminology

For an introduction to tableaux for propositional logic see [11, 28] and [10, 14] for propositional modal logics.

We construct propositional formulae A, B, C from a set of propositional atoms $p, q \in \mathcal{P}$ and the boolean constants $\top$ and $\bot$ with the logical connectives $\wedge$, $\vee$, $\neg$, $\supset$ etc. Propositional modal logics are obtained by adding the unary operators $\Box$ (necessity) and $\Diamond$ (possibility).

For sake of modularity we use signed formulae and the α, β, ν, π notation of Smullyan [28] and Fitting [10]. Thus a *signed formula*, denoted by φ and ψ, is a pair $\mathtt{t}.A$ or $\mathtt{f}.A$. In particular $\mathtt{t}.A$ ($\mathtt{f}.A$) is a formula with positive (resp. negative) sign. If A is a propositional letter we say that it is a signed atom. The intuitive interpretation is that A is assumed to be respectively true and false. The *conjugate* $\overline{\varphi}$ of a signed formula φ is obtained by switching $\mathtt{t.}$ with $\mathtt{f.}$ and vice versa. The uniform classification of formulae is given in Table 1.

Sets of formulae are represented by Γ and Δ and a *sequent*, written using Gentzen notation, is the pair $\Gamma \Longrightarrow \Delta$. With signed formula the sequent $\Gamma \Longrightarrow \Delta$ is represented as a set $\mathcal{S}$ of signed formulae as follows: $\mathcal{S} = \{\mathtt{t}.A \mid A \in \Gamma\} \cup \{\mathtt{f}.A \mid A \in \Delta\}$. We abbreviate $\mathcal{S} \cup \{\varphi\}$ as $\mathcal{S}, \varphi$. In this set-oriented framework the usual structural rules such as weakening and contraction become redundant.

4 Propositional Logic

The rules and axioms of the calculus are given in Fig. 1. The only new rule is denoted by (*simp*), and we call it *local simplification.* The use of signed formulae makes the definition of rule (*simp*) extremely compact.

α	α_1	α_2
$\mathtt{t}.A \wedge B$	$\mathtt{t}.A$	$\mathtt{t}.B$
$\mathtt{f}.A \vee B$	$\mathtt{f}.A$	$\mathtt{f}.B$
$\mathtt{f}.A \supset B$	$\mathtt{t}.A$	$\mathtt{f}.B$

α	α_1, α_2
$\mathtt{t}.\neg A$	$\mathtt{f}.A$
$\mathtt{f}.\neg A$	$\mathtt{t}.A$

β	β_1	β_2
$\mathtt{f}.A \wedge B$	$\mathtt{f}.A$	$\mathtt{f}.B$
$\mathtt{t}.A \vee B$	$\mathtt{t}.A$	$\mathtt{t}.B$
$\mathtt{t}.A \supset B$	$\mathtt{f}.A$	$\mathtt{t}.B$

ν	ν_0
$\mathtt{t}.\Box A$	$\mathtt{t}.A$
$\mathtt{f}.\Diamond A$	$\mathtt{f}.A$

π	π_0
$\mathtt{f}.\Box A$	$\mathtt{f}.A$
$\mathtt{t}.\Diamond A$	$\mathtt{t}.A$

Table 1. Uniform notation

$$\frac{\mathcal{S}, \alpha_1, \alpha_2}{\mathcal{S}, \alpha}\ (\alpha) \qquad \frac{\mathcal{S}, \beta_1 \quad \mathcal{S}, \beta_2}{\mathcal{S}, \beta}\ (\beta) \qquad \frac{\mathcal{S}, \psi[\varphi], \varphi}{\mathcal{S}, \psi, \varphi}\ (\mathit{simp}) \qquad \frac{\mathcal{S}}{\mathcal{S}, \varphi}\ (\mathit{thin})$$

$$\frac{}{\mathcal{S}, \mathtt{f}.\top}\ (\top) \qquad \frac{}{\mathcal{S}, \mathtt{t}.\bot}\ (\bot) \qquad \frac{}{\mathcal{S}, \varphi, \overline{\varphi}}\ (Ax) \qquad \frac{\mathcal{S}, \varphi \quad \mathcal{S}, \overline{\varphi}}{\mathcal{S}}\ (\mathit{cut})$$

Fig. 1. Propositional Sequent Rules with Simplification

Rule (*thin*) is not necessary and rule (*cut*) can be eliminated without hindering soundness and completeness [28, 11], although it may impact the computational complexity [7]. We introduce them to show that some particular forms of simplification are derived by using them in combination with (*simp*).

The definition of sequent tree and proof are standard [11, 28]: a *sequent tree* is a dyadic tree where each node is labelled by a sequent, the root is labelled with the sequent we are trying to prove and, for each node, its children are labelled with the corresponding consequents of a rule of the calculus. A *proof* is a sequent tree where each leaf is labelled with an axiom.

The next tool of our machinery, is the definition of the proper operation of simplification $\psi[\varphi]$. Signs will be left unchanged by simplification and we have just to specify the progression over formulae. In the following definition $\sharp$ is any binary (propositional) connective.

$$A[\varphi] \mapsto \begin{cases} \top & \text{if } \varphi = \mathtt{t}.A \\ \bot & \text{elseif } \varphi = \mathtt{f}.A \\ \neg(B[\varphi]) & \text{elseif } A = \neg B \\ B[\varphi] \,\sharp\, C[\varphi] & \text{elseif } A = B \sharp C \\ A & \text{otherwise} \end{cases} \tag{1}$$

By $A = B$ we mean, at this preliminary stage, syntactic identity. So $A \wedge B$ is different from $B \wedge A$. Further extensions, where the commutative properties of the connectives are considered, are described in the following subsections.

There is a number of observations worth making:

1. we do not only reduce branching, we get smaller and simpler formulae
2. the operation of simplification is parametric w.r.t. the connectives used;

$$
\dfrac{
\dfrac{A, (A \supset B) \vee (A \supset C), D \Longrightarrow A}{A, (A \supset B) \vee (A \supset C) \Longrightarrow D \supset A}\,\alpha_{\supset}
\qquad
\dfrac{\dfrac{\dfrac{A, B \Longrightarrow B, C}{A, A \supset B \Longrightarrow B, C}\,MP \quad \dfrac{A, C \Longrightarrow B, C}{A, A \supset C \Longrightarrow B, C}\,MP}{A, (A \supset B) \vee (A \supset C) \Longrightarrow B, C}\,\beta_{\vee}}{A, (A \supset B) \vee (A \supset C) \Longrightarrow B \vee C}\,\alpha_{\vee}
}{
\dfrac{A, (A \supset B) \vee (A \supset C) \Longrightarrow (D \supset A) \wedge (B \vee C)}{\dfrac{A \Longrightarrow (A \supset B) \vee (A \supset C) \supset (D \supset A) \wedge (B \vee C)}{\Longrightarrow A \supset ((A \supset B) \vee (A \supset C) \supset (D \supset A) \wedge (B \vee C))}\,\alpha_{\supset}}\,\alpha_{\supset}
}\,\beta_{\wedge}
$$

$$
\dfrac{\dfrac{\dfrac{A, B \vee C \Longrightarrow B \vee C}{A \Longrightarrow B \vee C \supset B \vee C}\,\alpha_{\supset}}{A \Longrightarrow ((A \supset B) \vee (A \supset C) \supset (D \supset A) \wedge (B \vee C))}\,simp_{\mathtt{t}.A}}{\Longrightarrow A \supset ((A \supset B) \vee (A \supset C) \supset (D \supset A) \wedge (B \vee C))}\,\alpha_{\supset}
$$

Fig. 2. HARP modus ponens versus simplification

3. the simplifying formula φ needs not be a literal nor have any normal form;
4. without restriction, simplification can proceed recursively down any level of nesting of propositional connectives.

The last step is one of the key differences with modus ponens and tollens applied in HARP [21], the KE β^c-rules [7] and the $\vee - simp_{0,1}$-rule used in [16]. Indeed we will show that they are particular forms of simplification where the recursion is stopped at the main connective of ψ.

Theorem 1. *Simplification (simp) is a sound and complete (invertible) rule for propositional logic.*

Proposition 1. *For propositional logic the size of $\psi[\varphi]$ is never larger than the size of ψ. It is strictly smaller whenever φ occurs as a subformula in ψ.*

An example is shown in Fig 2 where we compare simplification with the rule of Modus-Ponens in HARP. To simplify the notation we assume the usual associativity precedence of $\wedge$ over $\vee$ and of those two connectives over $\supset$ and use the $\Gamma \Longrightarrow \Delta$ notation in the obvious way.

Of course one may argue that the pattern A and $(A \supset B) \vee (A \supset C)$ could be easily recognized but it is also easy to change it substantially; for instance as $(A \supset B) \vee ((A \wedge \neg B) \vee D \supset C)$. This formula can still be proved with local simplification without any branching whereas HARP's proof is harder than the one shown in Fig. 2. Moreover the point of simplification is that we do not want to remember a lot of particular patterns.

There is another advantage: with open branches *simplification leads to fewer and smaller counter-models*, i.e. it sets the truth value of fewer propositions. For example one may try $\mathtt{t}.p \wedge (r \wedge \neg(q \supset p) \supset q)$. "Normal" tableaux have four open branches: the first (left-to-rigth) is $\{\mathtt{t}.p, \mathtt{f}.r\}$, then $\{\mathtt{t}.p, \mathtt{f}.q\}$, followed by $\{\mathtt{t}.p\}$ and $\{\mathtt{t}.p, \mathtt{t}.q\}$. The application of simplification yields only the model $\{\mathtt{t}.p\}$.

5 Subsuming other approaches

A wide variety of rules used by tableau provers turn out to be restricted and shallow forms of simplification. Here we consider the case for the unit of DPLL [9], the β^c-rules of KE [7],the Modus ponens and tollens of HARP [21], hyper-tableau expansions [2] and boolean constraint propagation [19].

First consider DPLL unit: "let l be a unit clause, delete every clause which contains l and delete any occurrence of $\bar{l}$ from the remaining clauses" [8].

Suppose we have $\Gamma, l \Longrightarrow \emptyset$ where Γ is set of clauses and we apply simplification to all formulae of Γ. By applying the boolean reductions $\bot \vee l_1 \vee \ldots l_n \mapsto l_1 \vee \ldots \vee l_n$ and $\top \vee l_1 \vee \ldots l_n \mapsto \top$, and finally $\top \wedge C_1 \wedge \ldots \wedge C_n \mapsto C_1 \wedge \ldots \wedge C_n$ we obtain the same result of the unit rule.

Fact 1. *The* DPLL*-unit rule is derived by the sequent calculus with the* ($simp$)*-rule restricted to signed atoms, when* $\mathcal{S}$ *is a set of clauses with positive sign.*

Boolean constraint propagation is an inference system where the unit rule is the only rule of inference. Hence we subsume directly also the BCP system [19].

Then, we simply need to restrict the application of the cut rule to atomic proposition to make the following observation:

Fact 2. *The* DPLL *procedure is derived by the sequent calculus with rules* ($simp$), (cut) *and* ($thin$) *restricted to signed atoms, when* $\mathcal{S}$ *is a set of clauses with positive sign.*

The reason for the successes of the split-rule of DPLL or the folding operation with unit lemmata [17] is indeed their combination with (hidden) unit simplification: we add a unit literal (the negation of the left-hand side), we apply simplification (unit) and then use thinning to eliminate the literal. So *the size of the right subtree is never larger.*

Equally, the β^c-rules of KE from [7] are a generalization to signed formulae of the modus ponens and modus tollens of HARP [21] and the $\vee - simp_{0,1}$ in [16]. Reversed bottom-up from their original tableau-notation they are as follows:

$$\frac{\mathcal{S}, \beta_2, \overline{\beta_1}}{\mathcal{S}, \beta, \overline{\beta_1}}\ \beta_1^c \qquad \frac{\mathcal{S}, \beta_1, \overline{\beta_2}}{\mathcal{S}, \beta, \overline{\beta_2}}\ \beta_2^c$$

It can be easily shown, by cases, that they are extremely restricted form of simplification. For instance consider the following instance of β_1^c:

$$\frac{\Gamma, B \Longrightarrow \Delta, A}{\Gamma, A \vee B \Longrightarrow \Delta, A}\ \mathtt{t}.A^c \qquad \frac{\Gamma, (A \vee B)[\mathtt{f}.A] \Longrightarrow \Delta, A}{\Gamma, A \vee B \Longrightarrow \Delta, A}\ simp_{\mathtt{f}.A}$$

If we applied the full-fledge version of simplification we would have had:

$$(A \vee B)[\mathtt{f}.A] \mapsto A[\mathtt{f}.A] \vee B[\mathtt{f}.A] \mapsto \bot \vee B[\mathtt{f}.A] \mapsto B[\mathtt{f}.A]$$

Thus, while KE only yields B, the full fledged use of simplification would have given us $B[\mathtt{f}.A]$. If A occurred somewhere in B simplification would have reduced B and, in general, shortened the proof further.

Fact 3. *The β_i^c rules for* KE *are instances of the simplification rule, restricted to β-formulae and such that recursive simplification stops at the main connective.*

As a simpler corollary we get:

Fact 4. *Modus ponens, tollens etc. of* HARP *and the $\vee - simp_{0,1}$ rules in [16] are instances of the simplification rule ($simp$), when recursion stops at the main connective of $\mathcal{S}$ formulae and it is only applied to formulae of particular forms.*

The extension step of hyper-tableaux [2] can also be subsumed:

Fact 5. *The expansion rule of propositional hyper tableaux [2] is derived by the sequent calculus with ($simp$) restricted to atoms, a restriction on the application of the β-rules (β-formulae must not contains negated atoms), when $\mathcal{S}$ is a set of clauses with positive sign.*

The next restriction that is imposed on clausal tableau proofs or model elimination techniques is *regularity* [2, 7, 17, 32, 30]: a literal should never appear twice on a branch. In the setting of propositional logic and clausal theorem proving (such as [2, 17]) this is equivalent to saying that a clause is never selected for the extension of a branch in the tableau if the extension step (an n-ary β-rule) will yield a literal to appear twice. It is a system to constrain the search space and in particular the non-determinism in the selection of clauses for reduction.

Since we use sequents as sets, and are not restricted to clausal normal form we rephrase it in more general terms: a proof is regular iff all subformulae introduced by α, β rules (i.e. $\beta_1, \beta_2, \alpha_1, \alpha_2$) do not occur already in the sequent $\mathcal{S}$ before the application of the rule.

Fact 6. *If simplification is given precedence over other rules then all propositional tableaux proofs are regular.*

We can devise more powerful forms of simplification if we replace $\wedge$ and $\vee$, which are commutative, associative and idempotent, with set-oriented versions $\bigwedge\{\}$ and $\bigvee\{\}$. For instance $A \wedge A \wedge B \wedge (C \vee D \vee E) \mapsto \bigwedge\{A, B, \bigvee\{C, D, E\}\}$. This is already a simplification since we eliminate duplicates. This topic is described in more details in [18]. In the experiments mentioned in §7 this is obtained by using prolog lexicographically ordered lists (without repetitions).

It has been sometimes argued that the use of the name TABLEAU in [6] was somehow misleading since it was an efficient implementation of DPLL. In the light of the results of this paper the choice of the name was indeed correct.

6 Modal Logics from K to S5

A wide range of tableau and sequent calculi have been proposed for modal logics (see e.g. [10]). For simplicity, we use the Gentzen-like formulation without adding extra-logical symbols. These techniques can be adapted to prefixed systems [10].

The rules are shown in Fig. 3. Different logics can be obtained by changing the operation $\mathcal{S}^*$. The same figure list the cases for the major modal logics

Modal Rules	Logic	Composition of S^*	Logic	Composition of S^*
$\dfrac{\mathcal{S}^*, \pi_0}{\mathcal{S}, \pi}\ \pi$	K	$\{\nu_0 \mid \nu \in \mathcal{S}\}$	T	$\{\nu_0 \mid \nu \in \mathcal{S}\}$
	K4	$\{\nu_0, \nu \mid \nu \in \mathcal{S}\}$	S4	$\{\nu \mid \nu \in \mathcal{S}\}$
	K45	$\{\nu_0, \nu, \pi \mid \nu, \pi \in \mathcal{S}\}$	S5	$\{\nu, \pi \mid \nu, \pi \in \mathcal{S}\}$
$\dfrac{\mathcal{S}, \nu_0}{\mathcal{S}, \nu}\ \nu$	Rule ν is only used for logics T, S4, S5			

Fig. 3. Modal Sequent Rules

of knowledge and belief[3] [10, 14]. Deontic variants are obtained by waiving the requirement to have a π-formula (and the corresponding π_0) in the π-rule.

We do not need to change our general framework nor to impose clausal normal form, as done in KSAT [12] to apply propositional DPLL. We inherit all propositional rules including simplification. We simply need to specify how $A[\varphi]$ will behave in presence of $\Box$ and $\Diamond$.

The simplest way is to consider $\Box B$ or $\Diamond B$ as atoms ("modal atoms") and apply boolean simplification. This is already sufficient for a speed-up [12, 16].

Fact 7. *Basic* KSAT *is an instance of the modal sequent calculus for logic* K *with propositional simplification,* $\mathcal{S}$ *contains only modal-CNF formulae with positive sign and with only the* $\Box$ *operator.*

This restriction is such that KSAT may not even simplify directly $\Box p \wedge \Diamond \neg p$ without reduction to normal form.

To enhance simplification we look inside modal atoms. We skip boolean rules as they remain unchanged. The new (interesting) cases in the simplification of $A[\varphi]$ are those where φ is a modal formula i.e. either a ν formula or a π-formula. The uniform notation gives extremely elegant rules.

*Modal-*K *simplification* is the simplest:

$$A[\nu] \mapsto \begin{cases} \Box(B[\nu_0]) \text{ if } A = \Box B \\ \Diamond(B[\nu_0]) \text{ if } A = \Diamond B \\ \ldots \qquad\qquad \text{as for propositional logic} \end{cases}$$

The idea is that if we want to simplify $\Box A$ with $[\mathtt{t}.\Box C]$ (a ν-formula) then we can exploit the properties of the modal operators and transform this operation in the simplification of A with $[\mathtt{t}.C]$ (the corresponding ν_0-formula).

In a nutshell, while propositional connectives leave the simplifying formula unchanged in the recursive steps (see §4 and reduction (1)), the semantics of the modal operators is such that we must do the recursion on the subformulae of both the simplified and the simplifying formula.

[3] With this sequent calculus á la Fitch, analytic cut is not eliminable for strong completeness of the logic S5. One has to resort to prefixed systems to avoid it [10].

For instance suppose we have $\Diamond(\Box p \supset \Box(p \supset r) \vee \Diamond\neg r)$ and want to simplify it with the ν-formula $\mathtt{t}.\Box\Box p$. Then we get the following computation:

$$
\begin{array}{ll}
(\Diamond(\Box p \supset \Box(p \supset r) \vee \Diamond\neg r))[\mathtt{t}.\Box\Box p] & \\
\quad \mapsto_K & \Diamond((\Box p \supset \Box(p \supset r) \vee \Diamond\neg r)[\mathtt{t}.\Box p]) \\
\quad \mapsto_{bool} & \Diamond((\Box p)[\mathtt{t}.\Box p] \supset (\Box(p \supset r))[\mathtt{t}.\Box p] \vee (\Diamond\neg r)[\mathtt{t}.\Box p]) \\
\quad \mapsto_K & \Diamond(\top \supset \Box((p \supset r)[\mathtt{t}.p] \vee \Diamond(\neg r[\mathtt{t}.p]) \\
\quad \mapsto_{bool} & \Diamond(\Box r \vee \Diamond\neg r)
\end{array}
$$

Modal-K4 *simplification* exploits directly the properties of transitivity (represented by the axiom $\Box A \supset \Box\Box A$):

$$
A[\nu] \mapsto \begin{cases} \Box(B[\nu][\nu_0]) \text{ if } A = \Box B \\ \Diamond(B[\nu][\nu_0]) \text{ if } A = \Diamond B \\ \ldots \qquad\qquad\quad \text{as for propositional logic} \end{cases}
$$

This means that for any logic containing the transitivity axiom we can first simplify B with ν and then simplify the result with ν_0. The presence of the transitivity adds a further level of simplification with respect to logic K.

Notice that the recursive call to simplification always terminates since the nesting of modal operators decreases in the simplified formula. For instance one may try the previous example with $\mathtt{t}.\Box p$ as the simplifying formula.

Modal-K5 *simplification* is the following

$$
\begin{array}{l}
A[\nu] \mapsto \Box(B[\nu_0]) \text{ if } A = \Box B \\
A[\nu] \mapsto \Diamond(B[\nu_0]) \text{ if } A = \Diamond B \\
A[\pi] \mapsto \Box(B[\pi]) \text{ if } A = \Box B \\
A[\pi] \mapsto \Diamond(B[\pi]) \text{ if } A = \Diamond B
\end{array}
$$

The rule can be understood by considering the corresponding axiom, $\Diamond A \supset \Box\Diamond A$: it tells that possibility-like formulae can propagate inside modalities.

The calculus for K45 uses the simplification procedures of both K5 and K4:

$$
A[\varphi] \mapsto \begin{cases} \Box(B[\nu][\nu_0]) \text{ if } A = \Box B \text{ and } \varphi = \nu \\ \Diamond(B[\nu][\nu_0]) \text{ if } A = \Diamond B \text{ and } \varphi = \nu \\ \Box(B[\pi]) \qquad \text{if } A = \Box B \text{ and } \varphi = \pi \\ \Diamond(B[\pi]) \qquad \text{if } A = \Diamond B \text{ and } \varphi = \pi \\ \ldots \qquad\qquad\quad \text{as for propositional logic} \end{cases}
$$

Theorem 2. *The rules for* L*-Modal simplification, where* L *is one of* K,T, K4, S4, K45, S5, *is a sound and complete (invertible) rule.*

Proposition 2. *For the propositional modal logics* K,T, K4, S4, K45, S5, *the size of* $\psi[\varphi]$ *is never larger than the size of* ψ.

7 Experimental Analysis

We performed experiments for both propositional and modal logics For propositional logics we have used the random 3-SAT and the IFIP benchmarks[4].

[4] The benchmarks are available at `http://www.dis.uniroma1.it/~massacci/ifip` both in Isabelle and prolog format.

We have used a tableau prover (Beatrix) implemented in `sicstus prolog`. Beatrix has been designed in the spirit of lean tableau theorem proving [4] with few modifications. At first, we used set oriented connectives implemented with prolog ordered lists without repetitions, e.g. the formula $a_1 \wedge a_2 \wedge (b_1 \vee b_2)$ is represented by the prolog list `[conj,a1,a2,[disj,b1,b2]]`. The benchmarks are easier to represent in this way. Second we used prolog lexicographic ordering among formulae, reduced conjunctions before disjunctions and small formulae before large ones. Finally, before applying an α or β rule to a formula φ we use φ to simplify all other formulae in the current sequent. Formula are reduced off-line in negation normal form (without any optimizations) as in leanTAP. The machine used was a workstation Sun SuperSPARK (time is in seconds).

To make the comparison more interesting we used two β-rules: the standard rule of Fig. 1 and the asymmetric rule for a limited form of cut [7, 17]:

$$\frac{\mathcal{S}, \beta_1 \quad \mathcal{S}, \beta_2}{\mathcal{S}, \beta}\ Dir \qquad \frac{\mathcal{S}, \beta_1 \quad \mathcal{S}, \overline{\beta_1}, \beta_2}{\mathcal{S}, \beta}\ Lem$$

A simple implementation is already sufficient for interesting conclusions wrt the relative gains of simplification and lemmaizing. In the following tables the normal β rule is denoted by "Dir" and the lemmaizing version by "Lem".

Remark 1. None of the problems listed here could be solved by Beatrix without simplification in any reasonable amount of time (one night), *no matter the branching rule.* So analytic cut does not necessarily helps for hard problems.

Thus, we do not show the benchmark results for Beatrix without simplification (in practice an enhanced version of leanTAP) since it would be an empty column except for the easiest formulae (first line of the table) of the 3-SAT benchmark.

Table 2 shows the result for the standard random distribution of 3-SAT[5]: $3-sat(V, C)$ means that samples had C clauses, with 3 literals selected uniformly among V variables and each literal negated with probability 0.5.

A number of experimental studies [6, 12] have shown that DPLL (and indeed most complete satisfiability checking procedures) exhibits an easy-hard-easy computation pattern as the ratio between the number of clauses and variables increases, with the hardest formulae around $C/V = 4.2 - 4.3$. This phenomena has been sometimes attributed to the "semantical" branching described by the splitting rule of DPLL. On the contrary we have found out the following:

Fact 8. *For random 3-CNF local simplification produces already the easy-hard-easy pattern. Only in the transition region lemmaizing improves the performance.*

Apart from the CNF benchmark we have tested our implementation on the IFIP benchmarks for hardware verification [5]. These benchmarks are known to be hard for DPLL [29] and are formulated in non clausal form, with an extensive use of equivalences ($\equiv$) and exclusive or ($\oplus$). Each problem requires to prove

[5] After $C = 10 \times V$ a linear relation between C and running time is due to inefficient list management by `prolog`.

Table 2. Benchmark on Random 3-SAT

Problem	Dir	Lem	Problem	Dir	Lem
3-sat(32,96)	.3	.2	3-sat(64,192)	1.4	1.0
3-sat(32,128)	3.9	1.2	3-sat(64,256)	334.6	38.4
3-sat(32,136)	6.1	1.8	3-sat(64,272)	554.3	56.4
3-sat(32,144)	6.9	2.1	3-sat(64,288)	1,050.9	72.0
3-sat(32,160)	8.2	2.4	3-sat(64,320)	568.6	60.0
3-sat(32,192)	7.7	2.6	3-sat(64,384)	240.3	39.4
3-sat(32,224)	6.0	2.2	3-sat(64,448)	139.3	30.6
3-sat(32,256)	5.3	2.3	3-sat(64,512)	74.3	23.5
3-sat(32,288)	4.9	2.3	3-sat(64,576)	69.6	21.8
3-sat(32,320)	4.8	2.4	3-sat(64,640)	59.6	20.4
3-sat(32,640)	5.9	3.7	3-sat(64,1280)	38.2	19.8
3-sat(32,1280)	10.5	6.5	3-sat(64,2560)	52.3	24.9

$$Spec = \begin{cases} car_1 \equiv (a_1 \wedge b_1), \\ som_1^s \equiv (a_1 \oplus b_1), \\ som_2^s \equiv (a_2 \oplus b_2 \oplus car_1), \\ cout^s \equiv (((a_2 \vee b_2) \wedge car_1) \vee (a_2 \wedge b_2)), \end{cases}$$

$$Impl = \begin{cases} cout_1 \equiv (b_1 \wedge a_1)), \\ som_1^i \equiv \neg((\neg a_1 \wedge \neg b_1) \vee (a_1 \wedge b_1)), \\ som_2^i \equiv \neg((((\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge b_2)) \wedge \neg cout_1) \vee \\ \qquad\qquad \vee (cout_1 \wedge \neg((\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge b_2)))), \\ cout^i \equiv ((a_2 \wedge cout_1) \vee (b_2 \wedge cout_1) \vee (a_2 \wedge b_2)) \end{cases}$$

$$Spec \cup Impl \Longrightarrow (som_1^s \equiv som_1^i) \wedge (som_2^s \equiv som_2^i) \wedge (cout^s \equiv cout^i)$$

Fig. 4. A two-bit ripple adder `rip02.be`

that the outputs of two circuits are equivalent i.e. the specification is equivalent to the implementation.

An example is shown in Fig. 4 as a sequent to be proved. It is a simple 2-bits binary ripple adder: it takes four bits of input (a_i and b_i) and returns three bits of output: the sums som_i and the carry $cout$. The two circuits are described by two sets of equivalences, each equivalence defining the output of a sub-circuit. The "upper" part in Fig. 4 describes the specification and the "lower" part describes the implementation. Using this description we must derive a (sequent) proof of the equivalence of the respective outputs.

In practice all have been solved except large multipliers, which are hard also for BDDs, timed out after 10 minutes (for a comparison between DPLL and OBDDs see [29]). Satisfiable formulae (marked by *) are solved within few milliseconds. What is more interesting here is the comparison with a standard

Table 3. Simplification versus Isabelle

Problem	fast_tac	ifip_tac
ex2	13.1	1.3
transp	43.8	0.2
risc	-	9.8
counter	-	68.8
hostint1	-	96.5
mul	-	130.9

Problem	Dir	Lem
ex2	0.0	0.0
transp	0.0	0.0
risc	0.4	0.6
counter	0.1	0.1
hostint1	0.3	0.2
mul	0.4	0.2

Prob.	fast_tac	ifip_tac
rip02	2848.2	1.6
rip04	-	994.5
rip06	-	-
rip08	-	-

Prob.	Dir	Lem
rip02	0.0	0.0
rip04	0.5	0.6
rip06	3.0	3.0
rip08	18.2	18.4

Table 4. Run times on IFIP Benchmarks

Problem	Dir	Lem	Problem	Dir	Lem	Problem	Dir	Lem	Problem	Dir	Lem
counter	0.1	0.2	ztwaalf1	0.8	0.8	add1	24.4	12.2	alupla20	784.0	618.1
d3(*)	0.1	0.1	ztwaalf2	0.8	0.4	alu	21.3	7.1	dc2	149.7	12.5
dk27	2.2	2.3	mjcg_no (*)	0.0	0.0	dk17	19.8	3.0	sqn	297.7	11.2
ex2	0.0	0.0	mjcg_yes	1.8	1.1	f51m	21.4	5.7	z9sym	166.8	9.8
hostint1	0.2	0.2	mp2d	3.7	1.1	table	13.9	2.8	mul03	-	20.1
misg	0.7	1.0	rip02	0.0	0.0	vg2	12.8	7.0	pitch	-	5.7
mul	0.2	0.2	rip04	0.4	0.5	x1dn	13.4	7.2	rd73	-	30.4
risc	0.4	0.5	rip06	2.9	2.9	z4	2.9	2.3	rom2	-	2.5
transp	0.0	0.0	rip08	18.2	18.4	z5xpl	11.5	4.1	root	-	33.7

interactive prover such as Isabelle [22]. This is shown in Table 3, where $fast\_tac$ is the standard automatic tactic used in Isabelle, and $ifip\_tac$ applies $fast\_tac$ after a pre-processing step which eliminates the equivalences by substituting each defined proposition with the corresponding formula (sub-circuit). Notice that also this preprocessing step, which eliminates abbreviations and leaves only "natural" (in)equivalences (at the price of an increase in size), is not enough.

The scaling factor for problems such as *rip0n.be* is more interesting than the absolute values. Absolute values simply tell us about the relative efficiency of lean proving in prolog vs big provers in ML, somehow strengthening the argument of [4]. The gap in the scaling factor (a factor of 1000 for Isabelle in the passage from $rip0n$ to $rip0n + 2$ is matched by a factor of 10 for Beatrix) tells that the use of simplification can substantial increase the tractability threshold for tableau calculi. Further benchmarks are shown in Table 4.

The experiments on modal logics lead to the same results noted in [16] on the gap between "standard" tableaux and KSAT: after the introduction of (limited) simplification, the gap claimed by Giunchiglia & Sebastiani [12] disappears.

Intuition 2. Simplification is a must (and a win) for efficient tableau provers, no matter if you use cut or not.

8 Conclusion

In this paper we have shown how the introduction of a simple technique called simplification may boost the deduction capabilities of tableau methods. It plays the same role of unit for DPLL and subsumption for resolution and provides a uniform framework which subsumes a number of techniques for the improvement of tableau-based methods.

Its flexibility and its simplicity make it possible its incorporation into sequent based interactive theorem provers used for hardware and software verification. Moreover there is no need of black-boxes since the computational effectiveness of DPLL and KSAT can now be compiled into standard sequent-based tactics.

While propositional and modal logics can be uniformly treated without much changes, the effective use of simplification for first order logic requires more advanced techniques. This topic is further discussed in [18].

At the end, there is a criticism we may have to face: after adding all these simplifications to sequent calculi are not we abandoning the calculus itself?

The answer is a question: is resolution with subsumption no longer resolution?

Acknowledgments

This work have been developed while the author was visiting the Computer Laboratory at the University of Cambridge.

I would like to acknowledge many fruitful discussions with L. Paulson, L. Carlucci Aiello, F. Donini, F. Giunchiglia, R. Goré, J. Harrison, F. Pirri and R. Sebastiani, the members of the AI group in Rome and the Automatic Reasoning group in Cambridge. Comments from the referees were helpful.

This work has been financially supported by ASI, CNR, and MURST grants.

References

1. M. Baaz and C. Fermüller. Non-elementary speedups between different versions of tableaux. In *Proc. of the 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, pages 217–230. Springer-Verlag, 1995.
2. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper tableaux. In *Proc. JELIA-96*, LNAI 1126, pages 1–17. Springer-Verlag, 1996.
3. B. Beckert. Semantic tableaux with equality. *Journal of Logic and Computation*, 7(1):38–58, 1997.
4. B. Beckert and J. Possega. `leanTAP`: Lean tableau-based deduction. *JAR*, 15(3):339–358, 1995.
5. L. J. Claesen, editor. *Formal VLSI Correctness Verification: VLSI Design Methods*, volume II. Elsevier Science, 1990.
6. J. Crawford and L. Auton. Experimental results on the crossover point in satisfiability problems. In *Proc. of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 21–27. AAAI Press/The MIT Press, 1993.
7. M. D'Agostino and M. Mondadori. The taming of the cut. *JLC*, 4(3):285–319, 1994.

8. M. Davis, G. Longemann, and D. Loveland. A machine program for theorem-proving. *Comm. of the ACM*, 5(7):394–397, 1962.
9. M. Davis and H. Putnam. A computing procedure for quantificational theory. *JACM*, 7(3):201–215, 1960.
10. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* Reidel, 1983.
11. M. Fitting. *First Order Logic and Automated Theorem Proving.* Springer-Verlag, 1990.
12. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. CADE-96*, LNAI 1104, pages 583–597. Springer-Verlag, 1996.
13. M. Gordon and T. Melham. *Introduction to HOL.* Cambridge University Press, 1993.
14. J. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *AIJ*, 54:319–379, 1992.
15. J. Harrison. Binary decision diagrams as a HOL derived rule. *The Computer Journal*, 38:162–170, 1995.
16. U. Hustadt and R. Schmidt. On evaluating decision procedure for modal logic. In *Proc. of IJCAI-97*, 1997.
17. R. Letz, K. Mayr, and C. Goller. Controlled integration of the cut rule into connection tableau calculi. *JAR*, 13(3):297–337, 1994.
18. F. Massacci. Simplification with renaming: A general proof technique for tableau and sequent-based provers. Tech. Rep. 424, Computer Laboratory, Univ. of Cambridge (UK), 1997.
19. D. McAllester, Truth maintenance. In *Proc. of AAAI-90*, pages 1109-1116, 1990.
20. N. V. Murray and E. Rosenthal. On the computational intractability of analytic tableau methods. *Bulletin of the IGPL*, 2(2):205–228, 1994.
21. F. Oppacher and E. Suen. HARP: a tableau-based theorem prover. *JAR*, 4:69–100, 1988.
22. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
23. D. Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960.
24. W. Quine. *Methods of Logic.* Harvard Univ. Press, iv ed., 1983.
25. J. A. Robinson. A machine oriented logic based on the resolution principle. *JACM*, 12(1):23–41, 1965.
26. J. Rushby. Automated deduction and formal methods. In *Proc. of CAV-96*, LNCS 1102, pages 169–183. Springer-Verlag, 1996.
27. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of AAAI-92*, 1992.
28. R. M. Smullyan. *First Order Logic.* Springer-Verlag, 1968. Republished by Dover, 1995.
29. T. Uribe and M. E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In *Proc. of the 1st Internat. Conf. on Constraints in Computational Logics*, LNCS 845, Springer-Verlag, 1994.
30. A. Urquhart. The complexity of propositional proofs. *Bull. of Symbolic Logic*, 1(4):425–467, Dec 1995.
31. A. Voronkov. The anatomy of vampire (implementing bottom-up procedures with code trees). *JAR*, 15(2):237–265, 1995.
32. K. Wallace and G. Wrightson. Regressive merging in model elimination tableau-based theorem provers. *Bull. of the IGPL*, 3(6):921–937, 1995.
33. H. Zhang and M. E. Stickel. An efficient algorithm for unit-propagation. In *Proc. of AI-MATH-96*, 1996.