Propositional and First-Order Logic Tableaux: Concept and Heuristics

Trouvain, Mira Toshiko (03710089)

June 2020

Contents

1 Introduction

When formal logics are worked with, the question whether or not a particular formula is a tautology (informally speaking: true in all cases) or in contrast unsatisfiable (false in all cases) is oftentimes of interest. One way to find an answer to that question is by use of the tableau calculus.

The basic concept of the so-called semantic tableau was first introduced by Evert William Beth in 1955 and further developed by Raymond Smullyan into analytic tableau. This paper will mostly be focused on the latter, as it is the most commonly used version of tableau and the method that the majority of tableau-based provers are based on.

Though not as popular as resolution/DPLL, with which the tableau method shares similarities, there have been many advancements concerning tableaux. Apart from propositional logic and first-order logic as presented here, the method has been adapted to work with other logics, particularly description logics or modal logics. Accordingly, the applications of tableaux are not just in theoretical fields, but in many other areas where automated reasoning and knowledge representation come into play, for example Computational Linguistics.

After presenting tableaux as a proof calculus in principle, more concrete approaches for constructing tableau proofs in propositional and first-order logic will be explained. Further, some flaws of the proof methods will be explained and some heuristics frequently used to combat those problems discussed.

2 Fundamentals of Tableaux

Similarly to the better known technique of resolution, tableaux are refutation systems applicable to various logics, i.e. they are used to search for a proof of validity for a given formula X by assuming its non-validity and deriving a contradiction from the negated formula $\neg X$.

A tableau proof is done by constructing a tree (called tableau) of which the nodes are labeled by formulas of the corresponding logic. In particular, every proof is begun with a tree consisting of a single node (labeled $\neg X$) and then expanded by applying so-called *Tableau Expansion Rules* that deduce simpler subformulas of X, possibly constructing multiple branches in the process. A branch in this context should be thought of as a path from root to leaf where every branch is set up to represent an alternate way of making the supplied formula true. Fitting in [4] also points out how more technically speaking, a branch can be thought of as a conjunction of the formulas on it, and the whole tableau as a disjunction of all of its branches.

When both a formula and its negation or alternatively the falsum ($\bot$) appear on the same branch, a contradiction has been found and that branch is called *closed*. A succesful tableau proof results in a *closed* tableau, which is simply defined as a tableau of which all branches are closed.

The validity of tableaux as a proof calculus depends on if it is both *sound*, i.e. proves tautologies only, and *complete*, i.e. proves all tautologies. In the name of simplicity, only conceptual/intuitive arguments will be given here - the more formal and detailed proofs from which the following have been derived can be found in [4] and [18].

Soundness. The proposition that X is true under any valuation or truth assignment v within the used logic (and therefore is a *tautology*) is equivalent to $\neg X$ being false under any v (hence: is not *satisfiable*). A tableau succeeds in showing the latter as it only closes if a contradiction is found on every branch, which based on the interpretation of branches given before means that satisfiability is not feasible.

Completeness. Completeness for tableau can be proven using contraposition, i.e. rather than showing "X tautology $\implies$ X has corresponding closed tableau", the proof shows that "X has no corresponding closed tabluea $\implies$ X is no tautology", and is largely based on the concept of *consistency*. In short, a formula without a closed tableau is called tableau consistent which in particular means that the formula is satisfiable. Intuitively, this seems sensible: a tableau is constructed by deriving subformulas, of which the satisfiability is based on the satisfiability of the original formula. As for the argument for completeness, a formula X having no tableau proof implies that $\neg X$ has no closed tableau and is therefore satisfiable, which means that X cannot be a tautology.

3 Propositional Tableaux

3.1 Propositional Logic

A propositional formula is either atomic or constructed from subformulas. An atomic formula is either a logical constant ($\top$, $\bot$) or a *propositional letter* ($p, q, \dots$) to which a truth value can be assigned. Valid compositions of propositional formulas may use negation ($\neg$) or binary connectives: conjunction ($\wedge$), disjunction ($\vee$), implication ($\supset$ / $\rightarrow$), etc with their usual semantics.

With his advance of the tableau method in [18], Smullyan introduced the concept of a *uniform* or *unified notation*. He categorized formulas that were constructed using binary connectives as either conjunctive or disjunctive and referred to them as α- and β-formulas respectively. This categorization is based on the fact that every formula $X \circ Y$ or $\neg(X \circ Y)$ is equivalent to either one of

$$\alpha \equiv \alpha_1 \wedge \alpha_2$$
$$\beta \equiv \beta_1 \vee \beta_2$$

in which the subformulas consist only of X/Y or their respective negations. In fact, the expressions on the right-hand side of these equations happen to be in *negation normal form* (NNF). This normal form is characterized by the fact that the negation is only applied to atomic formulas.

With the prospect of describing an algorithm that operates on propositional formulas, the unified notation allows for the declaration of rules that apply to formulas of either category in the same way, rather than explicitly stating rules for each connective.

3.2 Algorithm

A propositional tableau takes a set of propositional formulas in NNF as input. After initializing the tableau, the following *Tableau Expansion Rules* are applied to a selected branch θ and a non-literal formula X occurring on θ until all branches are closed:

- if $X = \neg\neg Z$: lengthen θ by Z
- if $X \in \{\neg\bot, \neg\top\}$: lengthen θ by $\top, \bot$
- if X is an α-formula: add α_1 and α_2 (in that order) to θ
- if X is a β-formula: add two children to node containing X and label them with $\{\beta_1\}$ (left) and $\{\beta_2\}$ (right)

One can see that the α- and β-rule as defined here do indeed comply with the reasoning behind tableau proofs. If an α-formula is assumed to be satisfiable, both α_1 and α_2 must be satisfiable as well, so if either one of them contradicts another part of the original formula, the assumption is proven to be unsatisfiable. Analogously for β-formulas: a satisfiable β-formula implies the satisfiability of at least β_1 or β_2. The closure condition of a tableau requires all branches to be closed, which specifically means that both β_1 and β_2 have been proven unsatisfiable.

The condition stating whether a branch is closed (either $\bot$ or both X and $\neg X$ for some X appear) may be restricted to atomic closure, i.e. either $\bot$ or X and $\neg X$ for some **atomic** X appear. This still allows for all tautologies to be proven by the tableau as proven in [5] while offering a termination condition that is easier to handle and verify, especially once the algorithm is actually implemented.

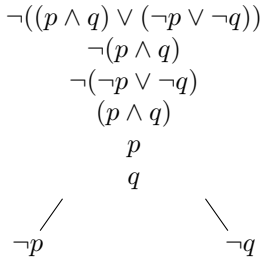
The given algorithm appears as more of a general description of what is possible, as for instance it does not specify what branch or formula a rule should be applied to. Because of the decidability of propositional logic, applying every rule to every formula possible eventually leads to a closed tableau if the formula in question was indeed a tautology. A corresponding proof can be found within the completeness proof in [5]. Still, this brute-force way is of course not the most efficient strategy and consequently, more consideration is needed on how to best approach a tableau proof, but that topic will be covered in 5.2.

3.3 Example

To better illustrate the described algorithm, a proof for

$$(p \wedge q) \vee \neg(p \wedge q) \quad \equiv \quad (p \wedge q) \vee (\neg p \vee \neg q)$$

will be given here. By convention, edges other than those signifying the splitting of branches are omitted.



In a first step, the α-rule is applied to derive $\neg(p \wedge q)$ and $\neg(\neg(p \wedge q))$. The double negation in the latter may be eliminated by the first Tableau Expansion Rule, adding $p \wedge q$ to the branch. Using the α-rule again gives us p and q. Now only $\neg(p \wedge q) \equiv (\neg p \vee \neg q)$ remains to be broken down further through application of the β-rule. Both operands are added on the left/right respectively and form a new branch, but as p and q have already been produced before, both new branches are now closed, and thus the tableau is closed.

4 First-Order Tableaux

4.1 First-Order Logic

The formalities of first-order logic formulas can be easily extended from those of propositional formulas. The main characteristic of FOL is that it allows for the quantification of formulas over a specified domain, hence the introduction of universal ($\forall$) and existential ($\exists$) quantifiers. In particular, a formula ϕ in which a variable x occurs is quantified using $\forall x$ to express that the formula will evaluate to true if x was substituted by any element of the domain. Analogously, $\exists x$ expresses that the formula will evaluate to true if x was substituted by some element of the domain. Furthermore, FOL supports predicates (conventionally denoted with $P, Q, R, \dots$) and functions ($f, g, \dots$) of some arity $n \in \mathbb{N}$ that may be applied to either n variables (denoted $x, y, z, \dots$) or n constants (denoted $a, b, c, \dots$) in the realm of said domain to express relations or characteristics.

In contrast to propositional logic, first-order logic formulas are more complex concerning their semantics. To give meaning to a first-order logic formula, a *model* must be specified that consists of both a *domain* $\mathbf{D}$ in which the variables and constants lie and an *interpretation* $\mathbf{I}$ for constants, functions and predicates. Depending on the model associated with a specific formula, the truth value of that formula may be true or false albeit keeping its syntactic structure. As an example,

$$(\forall x)P(x)$$

evaluates to true if $\mathbf{D} = \mathbb{N}_+$ and $P(x)$ was to be interpreted as *"x is larger than 0"*, but evaluates to false if the meaning of $P(x)$ is changed to *"x is prime"*. This ambiguity that is inherent to FOL formulas is a root for problems in FOL tableau, as the tableau calculus is designed to analyze formulas based on their syntactic appearance only.

Before specifying the algorithm for FOL, the unified notation is expanded to include universal (γ) and existential (δ) formulas: formulas of the form $(\forall x\Phi)$ and $\neg(\exists x)\Phi$ are called universal formulas or γ-formulas, whereas formulas constructed like $(\exists x)\Phi$ and $\neg(\forall x)\Phi$ are named existential or δ-formulas.

Universal		Existential	
γ	$\gamma(t)$	δ	$\delta(t)$
$(\forall x)\Phi$	$\Phi\{x/t\}$	$(\exists x)\Phi$	$\Phi\{x/t\}$
$\neg(\exists x)\Phi$	$\neg\Phi\{x/t\}$	$\neg(\forall x)\Phi$	$\neg\Phi\{x/t\}$

Figure 1: γ-/δ-formulas, taken from [4]

To comply with the idea of braking formulas down into subformulas, quantified formulas can be instantiated for concrete constants. Here, the expression $\Phi\{x/t\}$ indicates a substitution of every x in Φ with t.

4.2 Algorithm

The tableaux algorithm for formulas of first-order logic is largely similar to the one for propositional formulas. As the set of possible formulas has only been expanded by γ- and δ-formulas, giving Tableau Expansion Rules for those cases suffices for adapting the existing algorithm.

For a selected formula X on a branch θ, the rules are as follows:

- if X is a γ-formula, lengthen θ by $\gamma(t)$ for a closed (existing) term t
- if X is a δ-formula, lengthen θ by $\delta(p)$ for some newly introduced constant p

These rules are also easily justified. An universal formula is per definition true for any substitution, and hence for any substitution chosen within the proof. On the other hand, an existential formula is not necessarily true for any particular substitution but an unknown one, which is why a new constant is used as parameter when instantiating a δ-formula. The latter procedure is a simplified version of a technique known as *Skolemization*, in which existentially quantified variables are replaced by a term that depends on the universally quantified variables in the relevant scope.

Once again, the rules are non-deterministic and in FOL can even lead to non-termination, especially if every possible rule application is executed. This is especially true as in the propositional case, the validity of a formula depends only on the truth assignments to the propositional letters or more precisely, the definite values held by the atomic formulas. Hence, it can be determined solely from the syntactic structure of the formula. FOL formulas however depend in that respect on underlying models of the predicates and their domains as well, and therefore all possible substitutions are implicitly considered. Theoretically, the δ-rule can be used to introduce infinitely many new parameters that can then be used as substitute in any γ-formulas present, creating a potentially endless tableau, as for example:

$$
\begin{array}{c}
(\exists x)P(x) \\
(\forall y)Q(y) \\
P(a) \\
Q(a) \\
P(b) \\
Q(b) \\
\vdots
\end{array}
$$

This problem can diminished, e.g. by using *Unification* (in short: substituting variables in two or more FOL formulas in a way that makes the formulas equivalent) or the *Universal Formulae* heuristic presented in 5.3. Moreover, it is often reasonable to apply δ-formulas first and then use the new parameters as substitutes in universal formulas.

Apart from that, FOL in contrast to propositional logic is only semi-decidable, so the tableaux method may not terminate even when the mentioned restrictions are imposed. [2] In particular, it is possible that no proof is found for either X or $\neg X$.

4.3 Example

Again, an exemplary proof of $((\forall x)P(x) \wedge R(x)) \rightarrow (\neg(\exists y)\neg P(y))$ will be demonstrated.

$$\neg(((\forall x)P(x) \wedge R(x)) \rightarrow (\neg(\exists y)\neg P(y)))$$
$$(\forall x)P(x) \wedge R(x)$$
$$\neg\neg(\exists y)\neg P(y)$$
$$(\exists y)\neg P(y)$$
$$\neg P(a)$$
$$P(a) \wedge R(a)$$
$$P(a)$$
$$R(a)$$

First, the α-rule is applied to split the implication into its premise and conclusion. After eliminating the double negation, a new parameter a is introduced to apply the γ-rule to $(\exists y)\neg P(y)$, deriving $\neg P(a)$. Now, the new parameter can be used in applying the δ-rule to $(\forall x)P(x) \wedge R(x)$. Using the α-rule on the result derives $P(a)$ and $R(a)$, which means that the tableau is closed as $\neg P(a)$ has been stated before.

5 Improvements and heuristics

5.1 Problems

As mentioned earlier, the tableau algorithms for both propositional and first-order logic have some faults concerning their efficiency and determinism. Tableau Expansion Rules may be applied deliberately, which leaves the possibility for very efficient, but also very inefficient or in the worst case even non-terminating proofs.

Another aspect to consider is the branching of the tableau. A large number of branches leads to a larger search space, as every branch must be worked on individually to check for closure. Additionally, branches permit redundancies as described in 5.2.

Apart from the branching mechanic, the rules themselves allow for redundant or repeated appearances of the same formula. Technically, one could re-apply the same rule to the same formula persistently. Of course, this can be easily avoided by common sense if a proof is done by hand, but an automated version of the algorithm needs more explicit restrictions.

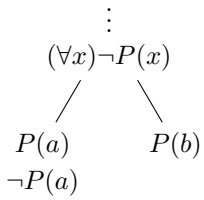
Nevertheless, the tableau proof method can be improved significantly with regards to the aforementioned points. In the following sections, some simpler heuristics as well as approaches known from DPLL (or SAT-solvers in general) that are commonly applied to tableaux will be discussed. The given examples are mostly supposed to illustrate the approach of the heuristics in principle, the benefits of the described methods are of course much more obvious in proofs of a larger scale.

5.2 Basic improvements

A very basic first improvement that may be introduced are some heuristics concerning the order in which the Tableau Expansion Rules are applied or, more accurately, which kinds of formula should be handled first in tableau proofs. Obviously, branching and especially redundancy between branches should be reduced to a minimum, which is why Fitting [6] proposes to exert the α-rule where possible before turning to disjunctive formulas. This avoids scenarios in which the α-rule is applied to a conjunctive formula above the branching point and α_1/α_2 have to be added to both branches respectively, which is both redundant and space-inefficient, as seen in the following examples:

Concerning FOL-tableaux, Fitting [6] advises to prioritize the use of propositional rules and further deal with existential formulas before turning to the universal ones. The latter is intuitively sensible, because as mentioned earlier, contradictions can often be discovered by substituting the variable in an universal formula with a parameter that was introduced using the δ-rule. This technique has e.g. been implemented in the HARP system [15].

Even with this guidelines about the order of rule application in place, the algorithm and its rules themselves still allow for non-termination as they only specify what may be done with some specific kind of formula, but do not give any restrictions concerning how to proceed with a tableau proof on a larger scale and, in the worst case, allows the proof to run in circles. To address this, Fitting introduces the concept of *strictness* [7]. A *strict* tableau is characterized by the fact that on every branch, every formula is processed by the application of a Tableau Expansion Rule once at most. In practice, this may be realized by maintaining a list of formulas for every branch from which a formula is removed once it has a rule applied to it. In the propositional case, this approach is particularly convenient, as strictness may be imposed without any restrictions. Additionally, propositional tableaux as mentioned before are guaranteed to result in closure if a proof exists and every possible formula is applied, so strict tableaux are in fact a decision procedure for propositional logic. For FOL tableaux, that is not exactly true, because proofs may rely on repeated applications of the γ-rule where the universal formula is applied to different terms. An example for this situation is the following:



When strictness is imposed to the γ-rule, the right branch cannot be closed as $\neg P(b)$ would have to be derived, but $(\forall x)\neg P(x)$ has already been used to add $\neg P(a)$ and can therefore not be used again. With exception of that rule however, FOL tableau proofs can still be constructed to be strict and benefit from that approach to some extent.

Lastly, the proof method also benefits from simplifying and normalizing any derived formulas. In the case of non-atomic closure, a contradiction is found only if a formula X appears alongside $\neg X$ where both X are exactly identical. This approach leads to the problem that even if a formula appears that is equivalent to $\neg X$, it is not identified as contradicting X. As an example: $p \vee q$ and $\neg p \wedge \neg q$ appear on the same branch in a tableau. The two formulas propose a direct contradiction as $\neg p \wedge \neg q \equiv \neg(p \vee q)$, but are not identified as doing so. Thus, the branch they appear on will be built further, increasing the size of the tableau unnecessarily. Furthermore, formulas may appear that can be simplified to either reduce branching ($p \vee (p \vee q) \equiv p \vee q$) or find closure earlier ($p \wedge \neg p$). Horrocks gives a number

of appropriate rules for simplifying/normalizing formulas of the description logic $\mathcal{ALC}$ [12], but in principle, analogous rules are applicable to propositional and first-order logic as well.

5.3 Universal Formulae

The restriction of strictness described in 5.2 cannot be applied to universally quantified formulas. As such, there may be several applications of the γ-rule to the same formula with different (or the same) substitutions, which significantly increases the search space. To combat this problem, a now commonly used heuristic named *Universal Formulae* was first suggested by Beckert [1]. Its approach is to identify formulas that are universally quantified or, in other words, in which a variable x may be substituted by any constant (*universal formula* with respect to x). A placeholder X is then used to instantiate the formula for x, signifying that if necessary, the formula may be treated as if instead of X, any appropriate constant occurred in the formula. Beckert notes that while this is in general an undecidable problem, many universal formulas may be found using the following rules:

- a formula that results from an application of the γ-rule is universal with respect to the variable x if the outermost bound variable in the original formula was substituted by x
  e.g.: $(\exists y)P(X) \rightarrow \neg P(y)$ that was derived from $(\forall x)(\exists y)P(x) \rightarrow \neg P(y)$ is universal with respect to X

- a formula that results from an application of either the α-rule or the γ-rule to a formula that is universal with respect to x is also universal with respect to the same x
  e.g.: with $P(X) \wedge P(y)$ being universal with respect to X, the same is true for $P(X)$ as it is a result of applying the α-rule

Once such a formula has been found, it can be used analogously to the version with the substituted variable, i.e. further Tableau Expansion Rules may be applied and closing branches recognized, but without generating specified formulas for every time it is used.

As an example, here is a proof for $(\forall x)P(x) \rightarrow (P(a) \wedge P(b))$:

$$\begin{array}{c} \neg((\forall x)P(x) \rightarrow (P(a) \wedge P(b))) \\ (\forall x)P(x) \\ \neg(P(a) \wedge P(b)) \\ P(X) \\ \diagup \qquad \diagdown \\ \neg P(a) \qquad\quad \neg P(b) \end{array}$$

In this case, the X in $P(X)$ signifies that $P(X)$ is an universal formula where X may be implicitly substituted with existing terms as needed. That way, $P(X)$ can be used with both $\neg P(a)$ and $\neg P(b)$ to close the respective branches.

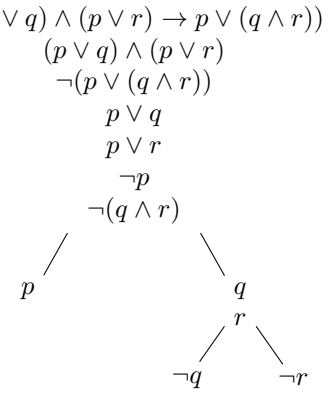
Posegga and Schmitt [16] compare the performance of a tableau-based prover written in Prolog with and without a simple implementation of Universal Formulae. They came to the conclusion that while in some cases, the version with the heuristic is admittedly slower and creates an overhead, it may also yield exponentially faster proofs or permit proofs that are not possible altogether without the heuristic.

5.4 Semantic Branching

Both the heuristics concerning the order of Tableau Expansion Rule applications and the Universal Formulae heuristic reduce redundant formulas to an extent. Even so, redundancies occur on a more abstract level based on the semantics of disjunctions, as pointed out by Giunchiglia and Sebastiani [9]. The two subformulas β_1 and β_2 of any β-formula are *mutually consistent*, i.e. satisfiability of one does not necessarily imply unsatisfiability of the other. For that reason, testing both for satisfiability is likely to implicitly examine truth assignments multiple times.

The basic tableau procedure performs so-called *syntactic branching* by adding either operand of the disjunction as a child node to the tableau. Instead of the traditional β-rule, Guinchiglia/Sebastiani propose the use of *semantic branching*, in which after the left branch with β_1 has lead to closure, $\neg\beta_1$ is assumed. This method is based on a similar idea in the DPLL algorithm, where variables are assigned a truth value and if the resulting formula proves to be unsatisfiable, are assigned the opposite truth value.

To illustrate this heuristic, a proof for $(p \vee q) \wedge (p \vee r) \rightarrow p \vee (q \wedge r)$ will be discussed:



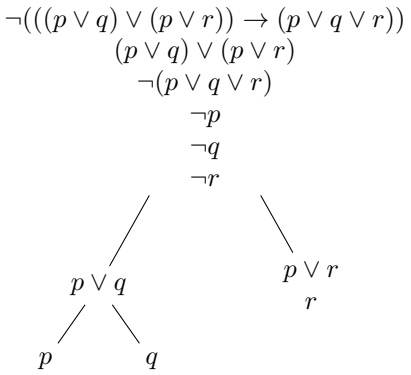
The main point in this proof is that after p lead to closure on the left-most branch and $p \vee r$ would be added to the right branch, $p = \bot$ is already assumed. Instead of branching under q in the right branch into p and r from the formula $p \vee r$ above, r can simply be added as $r = \top$ is now the only possibility for $p \vee r$ to be satisfiable.

5.5 Boolean Constraint Propagation

Similarly to semantic branching, *Boolean Constraint Propagation* (BCP) or *unit propagation* is a technique that was adapted from DPLL to optimize tableau proofs. In the DPLL algorithm ([3]), the so-called *one-literal rule* states that the last remaining literal in a clause (or disjunction) of which all other literals have been removed (i.e. been assigned the truth value $\bot$) may automatically assigned the truth value $\top$.

Freeman described the use of this approach for propositional tableaux [8] and it has since been employed in different tableau-based reasoning systems such as FaCT [11] or SAGA [10]. With tableaux, unnecessary branching is avoided by screening every disjunction for formulas that have already been proven unsatisfiable. If all but one operand of the disjunction are unsatisfiable, the corresponding branch is lengthened by said operand only. The β-rule is now only needed if more than one operand of the disjunction is still possibly satisfiable.

As for determining whether a formula is proven unsatisfiable, this can be derived from the semantics behind tableau proofs. An atomic formula is deemed unsatisfiable if its negation occurs on the same branch as itself and is thus directly contradicted. α- and β-formulas on the other hand are often contradicted only implicitly, as in: both of their subformulas have been contradicted/proven unsatisfiable, and thus their composition.



In the case of this example, BCP is successfully applied to the disjunction $p \vee r$. p has already been used to close the left branch as it contradicts $\neg p$ and can therefore be considered unsatisfiable. This leaves only r to be added to the branch instead of splitting it again.

5.6 MOMS

The MOMS heuristic is yet another heuristic commonly used for SAT procedures in combination with BCP. After being first proposed by Pretolani [17], it was adopted for tableaux by Freeman [8] also. The description given here is mainly based on those of Horrocks ([12] and [13]).

MOMS seeks to increase the effectivity of BCP by pruning the search space beforehand. As mentioned by Hladik [10], it implicitly requires semantic branching.

When considering which disjunct to branch on when multiple disjunctions are present, this heuristic chooses those with *Maximum Occurrences in clauses of Minimum Size* – hence the name. That way, if the chosen disjunct turns out to be unsatisfiable, BCP can be used to deduct as many formulas as possible from the disjunctions.

In practice, this decision is done by counting the occurrences of each disjunct, refining the result e.g. by weighing them with the size of the respective disjunction ([14]).

After choosing a concrete disjunct C, the one with less occurrences of $\{C, \neg C\}$ is branched on first. This is done to increase the benefits of BCP in that first branch, but leads to a slower calculation if contradictions cannot be found. Furthermore, MOMS can hinder the benefits of *backjumping*, a method in which rather than *backtracking* and therefore visiting every single branching point before the contradiction in order, the most suitable point to jump back to is calculated. Hence, some systems (e.g. FaCT) forgo it.

6 Conclusion and outlook

All in all, the tableaux calculus is a proof method that has rather simple syntax-based rules and can be extended to first-order logic easily. Although it has some faults in principle,

these can be mostly avoided in the propositional case and at least somewhat relieved with FOL through heuristics, which can oftentimes be adapted from other SAT procedures.

Here only the basics of tableaux and simpler heuristics were covered, but of course much more advanced methods have been developed as well. Many of those can be found in the proceedings of the annual *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*.

References

[1] Bernhard Beckert. Adding equality to semantic tableaux. In *Proceedings, 3rd Workshop on Theorem Proving with Analytic Tableaux and Related Methods, Abingdon*, pages 29–41, 1994.

[2] Ricardo Caferra. *Logic for computer science and artificial intelligence*, chapter 5.3. John Wiley & Sons, 2013.

[3] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

[4] Melvin Fitting. *First-order logic and automated theorem proving.* Springer Science & Business Media, 2012.

[5] Melvin Fitting. *First-order logic and automated theorem proving*, pages 69–70. Springer Science & Business Media, 2012.

[6] Melvin Fitting. *First-order logic and automated theorem proving*, page 139. Springer Science & Business Media, 2012.

[7] Melvin Fitting. *First-order logic and automated theorem proving*, pages 44–46,139. Springer Science & Business Media, 2012.

[8] Jon William Freeman. *Improvements to propositional satisfiability search algorithms.* PhD thesis, Citeseer, 1995.

[9] Fausto Giunchiglia and Roberto Sebastiani. A sat-based decision procedure for alc. *KR*, 96:304–314, 1996.

[10] Jan Hladik. Implementation and optimisation of a tableau algorithm for the guarded fragment. pages 287–347, 12 2001.

[11] Ian Horrocks. The fact system. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 307–312. Springer, 1998.

[12] Ian Horrocks. Implementation and optimization techniques. In *The description logic handbook: theory, implementation, and applications*, pages 306–346. 2003.

[13] Ian Robert Horrocks. *Optimising tableaux decision procedures for description logics.* Citeseer, 1997.

[14] Robert G Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of mathematics and Artificial Intelligence*, 1(1-4):167–187, 1990.

[15] Franz Oppacher and E Suen. Harp: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4(1):69–100, 1988.

[16] Joachim Posegga and Peter H Schmitt. Implementing semantic tableaux. In *Handbook of Tableau Methods*, pages 581–629. Springer, 1999.

[17] Daniele Pretolani. Efficiency and stability of hypergraph sat algorithms. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:479–498, 1996.

[18] Raymond M Smullyan. *First-order logic.* Courier Corporation, 1995.