Depth-first Proof Search without Backtracking for Free Variable Semantic Tableaux

Bernhard Beckert

University of Karlsruhe, Institute for Logic, Complexity and Deduction Systems, 76128 Karlsruhe, Germany. E-mail: beckert@ira.uka.de

Abstract. We analyse the problem of constructing a deterministic proof procedure for free variable tableaux that performs depth-first proof search without backtracking. We present a solution for the general case of arbitrary free variable calculi with certain properties making them (syntactically) well-behaved. The strategy is based on *weight orderings* and a new notion of *regularity* to ensure fairness and it employs *reconstruction steps* to handle the destructiveness of free variable calculi. First-order predicate logic (FOL) and a free variable tableau calculus for FOL are used as an example for the application of our method.

1 Introduction

In this paper, we analyse the problem of constructing a deterministic proof procedure for free variable tableau calculi that performs depth-first proof search and is complete without backtracking. We present a solution for the general case of arbitrary free variable calculi that enjoy certain properties making them (syntactically) *well-behaved*; in particular these calculi are proof confluent. No other assumptions are made; and the method is not restricted to calculi for certain logics or formulae in certain normal forms (such as clausal normal form). Throughout the paper, we use first-order predicate logic (FOL) and a free variable tableau calculus for FOL as an example for the application of our method.

The concept of free variables, which is well known from theorem proving in first-order predicate logic, is based on the observation that tableau expansion rules are often *stable* w.r.t. the replacement of terms in their premisses by other terms; that is, if some conclusion $C(t)$ can be derived from some premiss $\Pi(t)$, then $C(t')$ can be derived from $\Pi(t')$ for all terms t'. If a calculus has expansion rules that are stable for certain premisses (such as the standard tableau expansion rules for FOL) and certain rule applications allow the introduction of an arbitrary new term (such as the expansion rule for universal quantification in FOL) then—instead of guessing a term—a free variable can be introduced that is later instantiated "on demand" when a non-stable rule is applied (such as the branch closure rule in FOL calculi). Using free variables reduces the number of choice points in the construction of a tableau proof and thus the size of the search space.

Intuitively, a formula containing a free variable X stands for a single (but unknown) element of the set of all formulae that are the result of replacing X by some term. All occurrences of a free variable in a tableau have to be instantiated by the same term.

Thus, instantiating a free variable to make a non-stable expansion rule application possible can make other expansions impossible, which is particularly problematic if the calculus is not proof confluent.

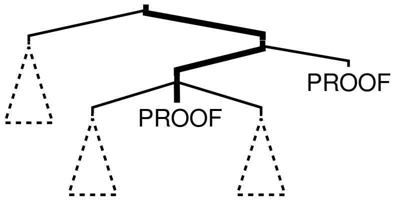


Fig. 1. A proof search tree.

Whether free variables are used or not, the development of an efficient tableau-based proof method for automated deduction can be separated into two phases: the design of a tableau calculus and the construction of a proof procedure based on that calculus. A tableau calculus is mainly characterised by a collection of deduction rules that may be employed to non-deterministically construct a tableau proof; a proof procedure is a description of how to search for a proof using a certain calculus. In the literature on tableau calculi, these two phases are often intermingled; refinements that are neither needed for soundness nor for completeness of a calculus but are intended to improve efficiency are made part of the calculus's definition. That is harmful because whatever properties the calculus is then shown to have are actually only properties of the refined calculus. Consequently, avoiding redundancy by making a calculus more deterministic and restricting the search space is mostly an issue when designing an efficient proof procedure (that notwithstanding, efficiency has to be considered when a calculus is designed).

The proof search space can be visualised as a search tree where each possible choice of the next expansion rule application to a tableaux T creates a node with as many successor nodes as T has different successor tableaux. The task of an automated deduction system is then to find a node in the search space that is labelled with a proof, i.e., a closed tableau (Fig. 1).

There are two main concepts for proof search: *breadth first* and *depth first* search. Depth first search requires that either there are no paths in the search tree that do not contain proofs or it is possible to avoid such paths using fairness strategies for the construction of tableaux.

As fairness strategies that allow depth first search are difficult to construct for free variable calculi, breadth first search is used by most automated deduction systems. Breadth first search allows to find shorter proofs than depth first search because all paths of the search tree are considered whereas, using depth first search, paths in the search tree that contain short proofs may be missed; fairness strategies only guarantee that some proof is found but it may not be the shortest one. However, the length of found proofs is not of great importance in automated deduction (the only advantage of short proofs is that they require less expansion rule applications and are thus easier to find); and breadth first search is very expensive as compared to depth first search because neighbouring paths in the search tree contain many similar or even identical tableaux that using breadth first search all have to be considered.

For all (practical) completion modes, i.e., (monotonic) functions m from $\mathbb{N}$ to sets of tableaux such that $\bigcup_{i \in \mathbb{N}} m(i)$ includes all constructible tableaux, the size $|m(i)|$ of the search tree grows exponentially in i. It is—even for small i—usually not possible to store all tableaux in $m(i)$ in the memory of a machine. Therefore, most implementations use *depth first iterative deepening* (DFID): the partial, finite search space consisting of all tableaux in $M(i) = \bigcup_{j \leq i} m(j)$ for some $i \in \mathbb{N}$ is searched for proofs in a depth first manner using backtracking, and if it turns out not to contain a proof, then i is increased (for example, the proof procedure described in [6] is of this type). Then, however, the tableaux in $M(i)$ are not available for the construction of the tableaux in $M(i+1)$; they have to be constructed again from scratch, which, however, merely causes polynomial overhead as compared to a breadth first search at the "right" level i because $M(i+1)$ is exponentially larger than $M(i)$. Although DFID search leads to acceptable performance of tableau-based automated theorem provers, it should be stressed that it is only a compromise used when no completeness preserving fairness strategy for depth first search is available.

The advantage of depth first proof search is that the information represented by the constructed tableaux increases at each proof step; no information is lost since there is no backtracking. In addition, considering similar tableaux or sequences of tableaux in different paths of the search tree is avoided.

In the case of non-destructive and proof confluent tableau calculi—such as the ground version of first-order tableaux that does not use free variables—it is relatively easy to use depth first proof search; it suffices to systematically add all possible conclusions until all branches of the constructed tableau are either fully expanded or closed. The situation is much more complicated in free variable calculi, which are destructive even if they are proof confluent. Applying a substitution may destroy formulae on a tableau that are needed for the proof such that they have to be deduced again from their premisses.

Up to now there was no practical solution to the problem of constructing deterministic proof procedures for free variable calculi that perform depth first search and are complete, i.e., that never fail to find a proof if there is one. Such procedures were only known for the special case of free variable calculi where branches are expanded without instantiating variables and only a single substitution is finally applied that is known to allow to close all branches simultaneously.

As a solution, we propose in this paper a deterministic search strategy that is based on *regularity* to make sure that there are no "cycles" in the search (it is not possible to deduce the same formulae or sub-tableau again and again), and *weight orderings*, i.e., each tableau formula is assigned a "weight" in such a way that there are only finitely many different formulae (up to variable renaming) of a certain weight; thus, if tableau formulae with lesser weight are deduced first, then sooner or later each conclusion is added to all branches containing its premiss, i.e., the strategy is *fair*. In addition, to handle the destructiveness of free variable calculi, our strategy employs *reconstruction steps*. Immediately after a tableau expansion that destroys formulae, the expansion steps that are needed to recreate the destroyed formulae are executed.

The main difficulty is to define a regularity condition that on the one hand is restrictive enough to avoid cycles in the proof construction and on the other hand is not too restrictive such that completeness is preserved. Baumgartner [1] suggests a regularity condition that turns out to be (just slightly) too strong, such that cycles are avoided but depth first search with *iterative deepening* has to be used to ensure completeness. Recently, Baumgartner et al. [2] described a depth first proof procedure for a connection calculus for FOL.

Our fairness strategy considers the whole tableau tree (and not only a single branch) both for checking regularity and for choosing a conclusion of minimal weight; a procedure based on this strategy may extend any branch of a tableau at any time. Note that this does not imply a large memory consumption; at least it is not worse than that of proof strategies where a "current" branch is extended until it is closed before other branches are considered and where DFID-based breadth first search is used to ensure completeness, as in that case all closed branches have to be stored for backtracking.

As said above, no *practical* deterministic proof procedures for free variable calculi were known up to now. There is trivially a (non-practical) deterministic proof procedure for all proof confluent calculi—namely a procedure that performs *breadth first* search in the background. "Practical" means that the computational complexity of deciding what the next rule application should be in each situation has to be reasonably low. In addition, the number of expansion steps that are necessary to find a proof has to be reasonably small as compared to the number of necessary steps when a breadth first search strategy is used.

If the fairness strategy we present in the following sections is used, then the complexity of deciding what the next expansion step should be is in the worst case quadratic in the size of the tableau to be expanded and its possible successor tableaux. In the average case the complexity is much lower as only those parts of a tableau have to be considered that are affected by one of the possible expansion rule applications. The size of the tableau proofs that are found (and thus the number of expansion steps) is at most that of the tableau proof constructed using DFID in the worst case (i.e., if coincidentally all paths in the search tree not containing a proof are considered first).

The structure of the paper is as follows: We define the syntax of a *logic* in a very general way in Section 2 and describe FOL within this framework as an example. In Section 3 tableau calculi are defined in general, and syntactical properties of calculi that are of relevance in the following sections are described; a calculus for FOL with these properties is presented. After introducing our new notion of regularity in Section 4 and that of weight orderings in Section 5, our method for constructing deterministic proof procedures for free variable calculi is presented in Section 6. Due to space restrictions, all proofs are omitted; they can be found in [3].

2 Logical Systems

The general framework. In this section, we define the syntax of *logical systems* in a very general way; only indispensable properties are part of the definition without which a useful tableau calculus for the logic cannot exist. The semantics of a logic is not defined as it is not considered in the following sections.

Definition 1. *Associated with a* logical system *(a* logic *for short) is a set Sig of signatures. For each signature $\Sigma \in Sig$ the syntax of the logic is given by: (1) A set $Form(\Sigma)$ of* formulae *and a set $Atom(\Sigma) \subset Form(\Sigma)$ of* atomic formulae (atoms)*, where the sets $Atom(\Sigma)$ and $Form(\Sigma)$ are decidable.*

We do not further specify what a signature is; Sig can be seen as a set of indices for distinguishing different instances of the logic (which *usually* use different sets of symbols).

Often, formulae are used in tableau calculi that contain symbols from an extended signature (e.g., Skolem symbols): A signature $\Sigma^* \in Sig$ is an *extension* of a signature $\Sigma \in Sig$ iff $Form(\Sigma) \subset Form(\Sigma^*)$ and $Atom(\Sigma) \subset Atom(\Sigma^*)$.

There is an important class of logical systems, including all predicate logics, where formulae contain *terms*. Again we make as few restrictions as possible on what a term is. In particular we do not assume that terms are of the form $f(t_1, \ldots, t_n)$; instead, a set of terms may be any decidable set of words occurring in the formulae. The only condition, that in fact is used to define the notion of terms, is that the set of formulae is closed under the replacement of (sub-)terms occurring in a formula by other (sub-)terms (for a formal definition see [3]). This closure property allows the replacement of terms by *free variables*, which are place holders representing arbitrary terms. A formula containing a free variable X stands for a single (but unknown) element of the set of all formulae that are the result of replacing X by some term. In the following we use the set $Var = \{X_i \mid i \geq 1\}$ of free variables. A non-ground term is constructed by replacing an arbitrary number of subterms of a ground term by free variables; and a non-ground formula is constructed by replacing ground terms occurring in a ground formula by non-ground terms.

The notions of substitution and unification are defined as usual: A *substitution* is a mapping of free variables into (non-ground) terms that is extended to formulae, tableaux, etc.; and a unifier μ of terms t, t' is a substitution such that $t\mu = t'\mu$.

First-order predicate logic. As an example for a logical system we use first-order predicate logic FOL. A signature $\Sigma = \langle P(\Sigma), F(\Sigma), \alpha_\Sigma \rangle$ in Sig_{FOL} consists of a set $P(\Sigma)$ of predicate symbols, a non-empty set $F(\Sigma)$ of function symbols, and a function α_Σ that assigns an arity $n \geq 0$ to each predicate and each function symbol. A function symbol of arity 0 is called a *constant*. In addition to the predicate and function symbols in signatures there is an infinite set V of *object variables* (which is disjoint from the set Var of free variables). The *logical operators* are $\vee$ (disjunction), $\wedge$ (conjunction), $\rightarrow$ (implication), and $\neg$ (negation), and the quantifier symbols $\forall$ and $\exists$. Terms and formulae over a signature Σ are constructed as usual in FOL.

The set $Term_{\mathrm{FOL}}(\Sigma)$ consists of all terms; the set $Form_{\mathrm{FOL}}(\Sigma)$, however, contains only *sentences*, i.e., formulae in which all object variables are bound by a quantifier; and the set $Atom_{\mathrm{FOL}}(\Sigma)$ of atoms of FOL is the set of all atomic sentences.

3 Tableau Calculi

3.1 Syntax of Tableaux

To include as many different calculi as possible in our general definition of tableau calculi, labels and truth value signs[1] are attached to formulae in tableaux. Labels have

[1] Only the two truth value signs T and F are used, which however does not imply a restriction to two-valued logic; these signs represent the fact that a formula is resp. is not satisfied by

many uses; they allow to make information about formulae and about the relation between formulae explicit. Labels are particularly useful in calculi for non-classical logics (e.g., many-valued, modal, and intuitionistic logics). Tableau calculi using labels are in general more powerful than calculi that encode the information otherwise contained in labels into the structure of tableaux. Many tableau-like calculi using labelled formulae are described in [7].

Definition 2. *Given a signature Σ of some logic and a set Lab of labels, a* tableau formula $\mathsf{S}{:}\sigma{:}F$ *consists of a truth value sign* $\mathsf{S} \in \{\mathsf{T}, \mathsf{F}\}$*, a label* $\sigma \in Lab$*, and a formula* $F \in Form(\Sigma)$*; it is* atomic *if* $F \in Atom(\Sigma)$*. In addition, the symbol* $\perp$ *is a tableau formula (indicating branch closure). The set of all tableau formulae is denoted with* $TabForm(\Sigma)$*. The complement* $\overline{\phi}$ *of a tableau formula* ϕ *is defined by:* $\overline{\phi} = \mathsf{F}{:}\sigma{:}F$ *if* ϕ *is of the form* $\mathsf{T}{:}\sigma{:}F$*, and* $\overline{\phi} = \mathsf{T}{:}\sigma{:}F$ *if* ϕ *is of the form* $\mathsf{F}{:}\sigma{:}F$ *(the complement of* $\perp$ *is undefined).*

A tableau *is a finitely branching tree whose nodes are labelled with tableau formulae from* $TabForm(\Sigma)$*. A* branch *of a tableau* T *is a maximal path in* T*. A tableau branch is* closed *if one of its nodes is labelled with* $\perp$*. A tableau is* closed *if all its branches are closed.*

To keep the notion of tableau calculi as general as possible, any function that assigns to a tableau a set of possible successor tableaux is considered to be a *tableau rule*. A tableau rule can change a tableau to which it is applied in an arbitrary way (tableau *expansion* rules are considered a special case of tableau rules).

We allow formulae from an extended signature Σ^* to be used in a tableau proof. Only the tableau formulae that are tested for satisfiability have to be taken from the language of the non-extended signature Σ; they are put on the initial tableau. The possibility to use formulae over an extended signature in the proof is indispensable for many tableau calculi; it allows, for example, to introduce Skolem symbols that are known not to occur in the initial tableau.

Definition 3. *A tableau calculus* $\mathcal{C}$ *is, for each signature* $\Sigma \in Sig$*, specified by: (1) an extension* $\Sigma^* \in Sig$ *of the signature* Σ*; (2) a set* Lab *of labels and an initial label* $\sigma^0 \in Lab$*; (3) a tableau rule* $\mathcal{R}(\Sigma)$ *that assigns to each tableau* T *over* Σ^* *a set of tableaux over* Σ^**, which are the (possible)* successor tableaux *of* T*.*

We now have everything at hand to define what the tableaux for a set $\mathfrak{F}$ of formulae are. The construction of tableaux for $\mathfrak{F}$ is in general a non-deterministic process, since a tableau may have any—even an infinite—number of possible successor tableaux.

Definition 4. *Let* $\mathcal{C}$ *be a tableau calculus; and let* $\Sigma \in Sig$ *be a signature. The set of all tableaux for a finite set* $\Gamma \subset TabForm(\Sigma^*)$ *of tableau formulae is inductively defined by: (1) A linear tableau whose nodes are labelled with the formulae in* Γ *is a tableau for* Γ *(an* initial tableau*). (2) If* T *is a tableau for* Γ *and* $T' \in \mathcal{R}(\Sigma^*)(T)$*, i.e.,* T' *is a successor tableau of* T*, then* T' *is a tableau for* Γ*. A tableau* T *is a tableau for a finite*

a model. The truth-values of a many-valued logic can be encoded into the labels attached to tableau formulae.

set $\mathfrak{F} \subset Form(\Sigma)$ of formulae if it is a tableau for the set $\{\mathsf{T}{:}\sigma^0{:}F \mid F \in \mathfrak{F}\}$ of tableau formulae.

A tableau proof *for (the unsatisfiability of) a set $\mathfrak{F} \subset Form(\Sigma)$ of formulae is a finite sequence $T_0, \ldots, T_n$ ($n \geq 0$) of tableaux for $\mathfrak{F}$ such that (1) T_0 is an initial tableau for $\mathfrak{F}$, (2) T_i is a successor tableau of T_{i-1} ($1 \leq i \leq n$), and (3) T_n is closed.*

Some useful tableau calculi, by definition, start with an empty initial tableau and allow to (later on) extend tableau branches by formulae from the set $\mathfrak{F}$ whose unsatisfiability is to be proven. This can easily be achieved by extending the tableau rule such that the formulae in $\mathfrak{F}$ can be derived from the empty premiss.

The class of free variable tableau calculi is—syntactically—characterised only by the fact that they use tableau formulae containing free variables. The replacement of terms by free variables is not restricted to the formula part G of a tableau formula $\mathsf{S}{:}\sigma{:}G$, but they can as well be introduced into the label σ and the truth value sign S.

3.2 Syntactical Properties of Tableau Calculi

Non-destructive calculi. A tableau calculus is *non-destructive* if applications of its tableau rule do not alter or remove formulae already on a tableau but only add new formulae, i.e., all possible successor tableaux of a tableau T contain T as an initial subtree.

Most free variable tableau calculi are destructive, because they allow the instantiation of the free variables by terms when the tableau rule is applied.

Proof confluent calculi. A tableau calculus is *proof confluent* if there are no "dead ends" in the proof search. A certain tableau rule application may be useless for constructing a proof for a formula set $\mathfrak{F}$, but it cannot prevent the construction of a proof if the calculus is proof confluent.

Definition 5. *A tableau calculus $\mathcal{C}$ is* proof confluent *if each sequence $T_0, \ldots, T_k$ of tableaux for a set $\mathfrak{F}$ of formulae for which a tableau proof exists can be extended to a tableau proof $T_0, \ldots, T_k, \ldots, T_n$ for $\mathfrak{F}$.*

Calculi with expansion rule. The most important syntactical property of tableau calculi forcing them to be "well-behaved" is that rule applications have only *local* effects. That is, the tableau rule is non-destructive and it extends only a *single* branch of a tableau. What the possibilities for extending a branch B are, only depends on B itself; no additional pre-conditions are allowed such as, for example, the presence of formulae on other branches. Of course, in free variable tableau it has to be possible to instantiate free variables; but we demand that this is the only operation with non-local effects.

A calculus has this "local effect" property if its tableau rule can be described in form of a *tableau expansion rule* that derives conclusions $\langle C, \sigma \rangle$ from the formulae (and possibly the structure) of a tableau branch. A conclusion consists of a finite set C of branch extensions and a substitution σ that is applied to the tableau when the conclusion is used for expansion.

Definition 6. *A* branch extension *is a finite subset of tableau formulae over Σ^*. A* conclusion *is a pair $\langle C, \sigma \rangle$ consisting of a finite set C of branch extensions and a substitution σ such that $C = C\sigma$. An* expansion rule *$\mathcal{E}(\Sigma)$ is a function that assigns to each (finite) tableau branch whose nodes are labelled with formulae from $\mathit{TabForm}(\Sigma^*)$ a set $\mathcal{E}(\Sigma)(B)$ of (possible) conclusions.*

An expansion rule $\mathcal{E}(\Sigma)$ characterises *the tableau rule $\mathcal{R}(\Sigma)$ of a calculus $\mathcal{C}$ if for all tableaux T the following holds: a tableau T' is a successor tableau of T (i.e., $T' \in \mathcal{R}(\Sigma)(T)$) iff there are branches $B_1, \ldots, B_k$ ($k \geq 1$) of T and a conclusion $\langle C, \sigma \rangle$ in $\mathcal{E}(\Sigma)(B)$ such that the tableau T' can be constructed from T by (1) extending each of the branches $B_1, \ldots, B_k$ by a new sub-branch for each extension E in C where the nodes in that sub-branch are labelled with the elements of E, and (2) applying the substitution σ to the tableau.*

If the expansion rule $\mathcal{E}(\Sigma)$ characterises the tableau rule $\mathcal{R}(\Sigma)$ of $\mathcal{C}$ for all signatures Σ, then $\mathcal{E}$ is said to be *the* expansion rule of $\mathcal{C}$.

Note that the expansion of more than one branch using the same conclusion $\langle C, \sigma \rangle$ is considered to be a single tableau construction step. However, it is *not* required that *all* branches are expanded in that step for which $\langle C, \sigma \rangle$ is a possible conclusion.

A special *closure* rule is not needed. A branch is closed by extending it with the special tableau formula $\perp$. Thus, branch closure can be considered to be a special case of branch extension.

Monotonic calculi. A tableau calculus with expansion rule is *monotonic* if the set of possible conclusions for a branch B' that is an expansion of a branch B contains all possible conclusions for B. In that case, rule applications are permutable.

Definition 7. *A calculus with expansion rule $\mathcal{E}$ is* monotonic *if $\mathcal{E}(\Sigma)(B_1) \subset \mathcal{E}(\Sigma)(B_2)$ for all branches B_1, B_2 over a signature Σ^* such that B_1 is an initial sub-path of B_2.*

Non-structural calculi. A tableau calculus is *non-structural* if the order of formulae on a tableau branch B is irrelevant for the result of applying the expansion rule to B.

Definition 8. *A tableau calculus with expansion rule $\mathcal{E}$ is* non-structural *if, for all branches B_1, B_2 whose formula sets are identical, $\mathcal{E}(\Sigma)(B_1)$ and $\mathcal{E}(\Sigma)(B_2)$ contain the same conclusions.*

Calculi that are monotonic w.r.t. substitution. The intuition behind the substitution σ that is part of a conclusion $\langle C, \sigma \rangle$ is that its application is a pre-condition for the derivation of C from the formulae on the branch. If a calculus is monotonic (Def. 7), then the pre-condition for deriving a certain conclusion is the presence of certain formulae on the branch that is expanded—and not the absence of formulae. Accordingly a notion of monotonicity w.r.t. the application of substitutions can be defined: Only that a certain variable *is* resp. *can be* instantiated in a certain way is allowed as a pre-condition—and not that the variable is *not* instantiated that way. Different and incompatible instantiations, however, can very well prevent the derivation.

Definition 9. *A free variable calculus with expansion rule $\mathcal{E}$ is* monotonic w.r.t. substitution *if for all tableau branches B: if $\langle C, \sigma \rangle \in \mathcal{E}(B)$ and ρ, τ are substitutions such that $\sigma = \rho \circ \tau$, then $\langle C, \rho \rangle \in \mathcal{E}(B\tau)$.*

Example 1. Assume that $\langle \{\{\mathsf{T}{:}p(a,b)\}\}, \{X \mapsto a, Y \mapsto b\} \rangle$ is a possible conclusion for a branch $B(X,Y)$ in a calculus for FOL that is monotonic w.r.t. substitution. Then, $\langle \{\{\mathsf{T}{:}p(a,b)\}\}, \{Y \mapsto b\} \rangle$ is a conclusion for $B(a,Y)$, and $\langle \{\{\mathsf{T}{:}p(a,b)\}\}, id \rangle$ is a conclusion for $B(a,b)$.

Well-behaved tableau calculi. We call a tableau calculus *well-behaved* if it has all the syntactical properties defined in this section—with the exception of non-destructiveness since instantiating free variables is allowed.

Definition 10. *A free variable tableau calculus is* well-behaved *iff (a) it is a calculus with expansion rule (Def. 6), (b) it is monotonic (Def. 7), (c) it is non-structural (Def. 8), and (d) it is monotonic w.r.t. substitution (Def. 9).*

If a tableau calculus is well-behaved, then it is proof confluent; moreover, its expansion rule can be represented as a function on finite sets of tableau formulae (a function on *premisses*).

Proposition 1. *If a tableau calculus is well-behaved, then it is proof confluent.*

Proposition 2. *Let $\mathcal{C}$ be a well-behaved free variable tableau calculus with expansion rule $\mathcal{E}$. Then, for all signatures Σ, there is a (single) function $\tilde{\mathcal{E}}(\Sigma)$ that assigns to each (finite) premiss $\Pi \subset \mathit{TabForm}(\Sigma^*)$ a set $\tilde{\mathcal{E}}(\Sigma)(\Pi)$ of (possible) conclusions such that for all tableau branches B the sets $\mathcal{E}(\Sigma)(B)$ and $\tilde{\mathcal{E}}(\Sigma)(\Pi_B)$ are identical where Π_B is the set of all tableau formulae on B.*

Proposition 2 above implies that for well-behaved calculi the function $\tilde{\mathcal{E}}(\Sigma)$ on sets of tableau formulae (premisses) can be seen as an alternative characterisation of the expansion rule $\mathcal{E}$. Therefore, in the case of well-behaved calculi, we identify the two functions, denote them both with $\mathcal{E}$, and call them "expansion rule".

In practice, expansion rules of well-behaved calculi are often described by means of rule schemata (see Table 2 in Section 3.3 for an example). In these schemata, the elements of the minimal premiss, that have to be present to allow the deduction of a certain conclusion $\langle C, \sigma \rangle$, and the set C of extensions are separated by a horizontal bar, while vertical bars in the conclusion denote different extensions. An explanation has to be attached to the schemata of how to compute the substitution σ. The use of schemata for defining tableau expansion rules fits perfectly in our framework, with the exception that different rule schemata are usually considered to define (or to *be*) different rules whereas we consider them to define different sub-cases of one (single) rule.

Definition 11. *Let $\mathcal{E}$ be an expansion rule of a well-behaved free variable calculus. A set $\Pi \subset \mathit{TabForm}(\Sigma)$ is a* minimal premiss *of a conclusion $\langle C, \sigma \rangle$ if it is a premiss of $\langle C, \sigma \rangle$, i.e. $C \in \mathcal{E}(\Sigma)(\Pi)$, and there is no subset $\Pi' \subset \Pi$, $\Pi \neq \Pi'$ such that $\langle C, \sigma \rangle \in \mathcal{E}(\Sigma)(\Pi')$.*

Well-behaved calculi exist for most logics. There are, however, some non-classical logics whose inherent properties make it difficult or even impossible to define a well-behaved tableau calculus; these include non-monotonic logics (since their calculi are not monotonic) and logics with resource restrictions such as linear and relevance logic (as rule applications in calculi for these logics have non-local effects).

Unfortunately, many calculi described in the literature and used in practice are "slightly" not well-behaved. Such calculi can often be repaired with minor changes. A typical example are calculi using expansion rules that introduce *new* symbols, i.e., symbols that must not occur on the branch or even the whole tableau. As the calculus for FOL defined in the following section shows, this type of rules can often be replaced by similar rules not violating monotonicity.

3.3 A Well-behaved Free Variable Calculus for FOL

In this section, a well-behaved free variable calculus $\mathcal{C}_{\mathrm{FOL}}$ for first-order predicate logic FOL is presented using the general framework from Definition 3.

Given a first-order signature Σ, the extended signature Σ^* used in proofs contains the free variables from Var as "constants" and, in addition, the set $F^{sko}(\Sigma)$ of Skolem function symbols containing infinitely many symbols of each arity $n \geq 0$.

α	$\alpha_1, \quad \alpha_2$
$\mathsf{T}{:}{*}{:}(F \wedge G)$	$\mathsf{T}{:}{*}{:}F,\ \mathsf{T}{:}{*}{:}G$
$\mathsf{F}{:}{*}{:}(F \vee G)$	$\mathsf{F}{:}{*}{:}F,\ \mathsf{F}{:}{*}{:}G$
$\mathsf{F}{:}{*}{:}(F \rightarrow G)$	$\mathsf{T}{:}{*}{:}F,\ \mathsf{F}{:}{*}{:}G$
$\mathsf{T}{:}{*}{:}\neg F$	$\mathsf{F}{:}{*}{:}F,\ \mathsf{F}{:}{*}{:}F$
$\mathsf{F}{:}{*}{:}\neg F$	$\mathsf{T}{:}{*}{:}F,\ \mathsf{T}{:}{*}{:}F$

β	$\beta_1, \quad \beta_2$
$\mathsf{T}{:}{*}{:}(F \vee G)$	$\mathsf{T}{:}{*}{:}F,\ \mathsf{T}{:}{*}{:}G$
$\mathsf{F}{:}{*}{:}(F \wedge G)$	$\mathsf{F}{:}{*}{:}F,\ \mathsf{F}{:}{*}{:}G$
$\mathsf{F}{:}{*}{:}(F \rightarrow G)$	$\mathsf{F}{:}{*}{:}F,\ \mathsf{T}{:}{*}{:}G$

$\gamma(x)$	$\gamma_1(x)$
$\mathsf{T}{:}{*}{:}(\forall x)(F(x))$	$\mathsf{T}{:}{*}{:}F(x)$
$\mathsf{F}{:}{*}{:}(\exists x)(F(x))$	$\mathsf{F}{:}{*}{:}F(x)$

$\delta(x)$	$\delta_1(x)$
$\mathsf{F}{:}{*}{:}(\forall x)(F(x))$	$\mathsf{F}{:}{*}{:}F(x)$
$\mathsf{T}{:}{*}{:}(\exists x)(F(x))$	$\mathsf{T}{:}{*}{:}F(x)$

Table 1. The four formula types of FOL.

Example 2. If a is a constant of the signatur Σ but it is not a Skolem constant, then the atom $p(a)$ is a formula over both Σ and the extended signature Σ^*. If c is a Skolem constant, then the atoms $p(c)$, $p(X)$, and $p(c, X)$ are formulae over Σ^* but not over Σ. Note that $p(c, x)$ is neither a formula over Σ nor over Σ^* as free occurrences of *object variables* are not allowed.

For $\mathcal{C}_{\mathrm{FOL}}$ we do not make use of the label mechanism; thus there is only one label $*$ that is attached to all tableau formulae, i.e., $Lab = \{*\}$ and $*$ is the initial label σ^0. To simplify notation, the abbreviation $\mathsf{S}{:}G$ is used for tableau formulae of $\mathcal{C}_{\mathrm{FOL}}$, i.e., the label $*$ is omitted.

To define the expansion rule of $\mathcal{C}_{\mathrm{FOL}}$, we use *unifying notation*. The set of non-atomic formulae in $\mathit{Form}_{\mathrm{FOL}}(\Sigma^*)$ is divided into four classes (Table 1): α-formulae (conjunctive), β-formulae (disjunctive), γ-formulae (universally quantified), and δ-formulae (existentially quantified).

$\dfrac{\alpha}{\alpha_1}$ α_2	$\dfrac{\beta}{\beta_1 \mid \beta_2}$	$\dfrac{\gamma(x)}{\gamma_1(X)}$ where X is any free variable	$\dfrac{\delta(x)}{\delta_1(t)}$ where $t = \mathit{sko}(\delta)$ (see Def. 12)	ϕ $\dfrac{\psi}{\bot}$ where ϕ and ψ are unifiable atoms; an MGU is applied

Table 2. Free variable rule schemata for first-order predicate logic.

The expansion rule schemata for α-, β-, and γ-formulae (Table 2) are stable w.r.t. replacement of terms by terms in the premiss. New free variables are introduced by rule applications to premisses containing γ-formulae. To make sure that the expansion rule is stable for δ-formulae and that the calculus is monotonic, we use a rule schema for δ-formulae that, for constructing the Skolem term, does not introduce a *new* Skolem function symbol. Rather, each equivalence class of δ-formulae identical up to variable renaming is assigned the same Skolem function symbol; this is an adaptation of the δ^{++}-rule described in [5].

Definition 12. *Given a signature $\Sigma \in \mathit{Sig}_{\mathrm{FOL}}$, a* Skolem term *of the form $\mathit{sko}(\phi) = f(X_1, \ldots, X_k)$ is assigned to each $\phi \in \mathit{TabForm}_{\mathrm{FOL}}(\Sigma^*)$ such that (1) $f \in F^{sko}(\Sigma)$, (2) $X_1, \ldots, X_k$ are the free variables occurring in ϕ; (3) for all $f' \in F^{sko}(\Sigma)$, if f' occurs in ϕ, then $f > f'$ where $>$ is an arbitrary but fixed ordering on $F^{sko}(\Sigma)$; and (4) for all δ-formulae $\psi \in \mathit{TabForm}_{\mathrm{FOL}}(\Sigma^*)$, $\mathit{sko}(\psi) = f(X'_1, \ldots X'_k)$ iff ϕ and ψ are identical up to renaming of bound object variables and up to replacing the free variables X_i by the X'_i.*

The expansion rule of $\mathcal{C}_{\mathrm{FOL}}$ is only unstable w.r.t. replacing terms by terms for premisses allowing to close a branch. The rule schema for closing branches is that a premiss $\Pi = \{\mathsf{T}{:}G,\ \mathsf{F}{:}G'\}$ allows the deduction of a conclusion $\langle\{\{\bot\}\}, \mu\rangle$ if μ is a unifier of G and G'. It is sufficient to only use conclusions $\langle\{\{\bot\}\}, \mu\rangle$ where μ is a *most general* unifier of G and G'.

In Table 2, the free variable expansion rule $\mathcal{E}_{\mathrm{FOL}}$ of $\mathcal{C}_{\mathrm{FOL}}$ is given schematically; and the following is its formal definition:

Definition 13. *The expansion rule $\mathcal{E}_{\mathrm{FOL}}$ of $\mathcal{C}_{\mathrm{FOL}}$ is for all $\Pi \subset \mathit{TabForm}_{\mathrm{FOL}}(\Sigma^*)$ defined by: $\mathcal{E}(\Sigma)_{\mathrm{FOL}}(\Pi)$ is the smallest set containing the following (possible) conclusions: (a) $\{\langle\{\alpha_1, \alpha_2\}, \mathit{id}\rangle\}$ for all $\alpha \in \Pi$; (b) $\{\langle\{\beta_1\}, \{\beta_2\}, \mathit{id}\rangle\}$ for all $\beta \in \Pi$; (c) $\{\langle\{\gamma_1(X)\}, \mathit{id}\rangle\}$ for all $\gamma \in \Pi$ and all free variables $X \in \mathit{Var}$; (d) $\{\langle\{\delta(t)\}, \mathit{id}\rangle\}$ for all $\delta \in \Pi$ where $t = \mathit{sko}(\delta)$ (Def. 12); (e) $\{\langle\{\bot\}, \mu\rangle\}$ if $\mathsf{T}{:}\sigma{:}G, \mathsf{F}{:}\sigma{:}G' \in \Pi$ such that $G, G' \in \mathit{Atom}_{\mathrm{FOL}}(\Sigma^*)$ are unifiable atoms, and μ is an MGU of G and G'.*

4 Regularity

The notion of *regularity* is well known from tableau calculi for classical logic in clausal normal form [11] and negation normal form [8, 9].

In the following, we define a concept of regularity for arbitrary *well-behaved* free variable calculi. Our concept differs from the classical notion in that it takes the whole tableau into concern and not only a single branch.

Assume that a sequence $T_1, \ldots, T_n$ of tableaux has already been constructed. We define an expansion rule application to T_n to be irregular if the successor tableau T_{n+1} is *contained* in one of the predecessor tableaux T_j—in particular, if T_{n+1} is contained in T_n. In that case, the sequence $T_j, \ldots, T_{n+1}$ constitutes a cycle in the proof search because T_{n+1} does not contain any information that is not already in T_j.

We define a tableau T_j to contain a tableau T_{n+1} iff each branch of T_j contains one of the branches of T_{n+1}. Intuitively, the tableau T_{n+1} is in that case redundant because, if closed sub-tableaux can be constructed below all branches of T_{n+1}, it is possible to construct closed sub-tableaux below all branches of T_j as each of them contains a branch from T_{n+1}.

Before formally defining the notion of a tableau *branch* B containing a branch B', we discuss two important aspects relevant for that definition. First, for B to contain B', it is in general *not* sufficient if the branch B contains *one* variant of each tableau formula ϕ occurring in B', namely in case B' contains several variants of ϕ that are all needed for a certain derivation. However, if there is a finite limit $k \in \mathbb{N}$ on the size of minimal premisses in a calculus, then it is sufficient if B contains a variant of each set of (at most) k formulae occurring on B'. This implies that at most k variants of each formula on B' are needed on B (where however, as described below, formulae may have to be considered to be effectively different although they are variants of each other on first sight). In the following we only consider calculi where there is a finite upper bound $k \in \mathbb{N}$ on the size of minimal premisses (the techniques described in the following can easily be extended to arbitrary well-behaved calculi by iteratively increasing the limit k during proof search, for the details see [3]).

Example 3. Minimal premisses in the calculus $\mathcal{C}_{\mathrm{FOL}}$ from Section 3.3 consist of at most $k = 2$ formulae. A branch B does therefore in general *not* contain a branch B' if it only contains *one* variant of each formula on B'. If, for example, the two formulae $\phi = \mathsf{T}{:}(\neg p(X) \wedge p(f(X)))$ and $\phi' = \mathsf{T}{:}(\neg p(X') \wedge p(f(X')))$ occur on B' whereas the branch B only contains ϕ (and B and B' are otherwise identical), then B contains a variant of each formula on B'. Nevertheless, the transition from B to B' is definitely not a cycle in proof search because—contrary to B—the branch B' can be closed.

The second important aspect is that not only the formulae on B and B' have to be considered but also *associated* formulae on other branches that have free variables in common with B and B'.

Definition 14. *Tableau formulae ϕ and ψ are* associated *if there is a free variable occurring in both ϕ and ψ. The set of all formulae in a tableau T that are associated with a formula ϕ, excluding ϕ itself, is denoted with $Assoc(T, \phi)$. Accordingly, if Φ is a set of tableau formulae, then $Assoc(T, \Phi) = \left(\bigcup_{\phi\in\Phi} Assoc(T, \phi)\right) \setminus \Phi$.*

Associated formulae play a role because the ordering of tableau rule applications used by a deterministic proof procedure as described in Section 6 has to take all formulae into account that are generated by an application. So, if $\phi(X)$ is a premiss for a certain tableau rule application that leads to the instantiation of X with a term t and there is a formula $\psi(X)$ on the tableau, then that application will generate the new formula $\psi(t)$; and the form of $\psi(t)$—and thus the form of the associated formula $\psi(X)$—affects the choice of the application.

We proceed to formally define when a branch B contains a branch B' and when a tableau T contains a tableau T'. These relations are transitive and reflexive.

Definition 15. *Let T and T' be tableaux over a signature Σ. A branch B of T k-*contains *a branch B' of T' for some $k \in \mathbb{N}$ iff for each set Φ' of formulae on B' with at most k elements there is a set Φ of formulae on B and a free variable renaming π such that: (1) $\Phi\pi = \Phi'$. (2) For each formula in $Assoc(T, \Phi)$ there is a formula ψ' in $Assoc(T', \Phi')$ that is identical to $\psi\pi$ up to the renaming of free variables* not *occurring in $\Phi\pi$ or Φ'. The tableau T k-*contains *the tableau T' iff each branch of T k-contains a branch of T'.*

In addition, if T and T' have no free variables in common, then T k-contains T' if there is a renaming ρ of free variables such that T k-contains $T\rho$ as defined above.

Example 4. Let $\Phi = \{\phi(X)\}$ and $\Phi' = \{\phi(X')\}$; moreover let $Assoc(T, \Phi)$ consist of the formulae $\psi(X, Y_1)$ and $\psi(X, Y_2)$. Then, Condition (2) in Definition 15 is, for example, satisfied if $Assoc(T', \Phi') = \{\psi(X', Y')\}$. It is *not* satisfied if $Assoc(T', \Phi') = \emptyset$; it is also not satisfied if $Assoc(T', \Phi') = \{\psi(Y', X')\}$.

Example 5. The tableaux T_1, T_2 on the right both 1-contain all of T_1' T_2', T_3'. The tableau T_1 also 2-contains them; but T_2 2-contains only T_1'.

$$\begin{array}{ccccc}
\mathsf{T}{:}\phi(X_1) & \mathsf{T}{:}\phi(X_1) & \mathsf{T}{:}\phi(Y_1) & \mathsf{T}{:}\phi(Y_1) & \mathsf{T}{:}\phi(Y_1) \\
\mathsf{T}{:}\phi(X_2) & & & \mathsf{T}{:}\phi(Y_2) & \mathsf{T}{:}\phi(Y_2) \\
 & & & & \mathsf{T}{:}\phi(Y_3) \\
T_1 & T_2 & T_1' & T_2' & T_3'
\end{array}$$

Example 6. Neither of the two tableaux on the right contains the other one; T_1 does not contain T_2 because the (single) branch of T_2 contains an additional formula; and, although a variant of each formula set on T_1 occurs on T_2, the tableau T_2 does not contain T_1 since for $\mathsf{T}{:}r(X') \in Assoc(T_2, \mathsf{T}{:}q(X'))$ there is no corresponding element in $Assoc(T_1, \mathsf{T}{:}q(X))$ and, thus, Condition (2) in Definition 15 is not satisfied.

$$\begin{array}{cc}
\mathsf{T}{:}p & \mathsf{T}{:}p \\
\mathsf{T}{:}q(X) & \mathsf{T}{:}q(X') \\
 & \mathsf{T}{:}r(X') \\
T_1 & T_2
\end{array}$$

Example 7. The tableaux T_1 on the right 1-contains the tableau T_2 on the far right. However, T_2 does *not* 1-contain T_1 because the formulae $\mathsf{T}{:}p(X')$ and $\mathsf{T}{:}q(X')$ in T_2 are associated. Intuitively, a transition from T_2 to T_1 does not constitute a cycle in proof search because the tableau T_1 can be closed whereas T_2 cannot be closed.

$$\begin{array}{cc}
\begin{array}{c}
\mathsf{F}{:}p(a) \\
\mathsf{F}{:}q(b) \\
\mathsf{T}{:}p(X) \quad \mathsf{T}{:}q(Y) \\
T_1
\end{array}
&
\begin{array}{c}
\mathsf{F}{:}p(a) \\
\mathsf{F}{:}q(b) \\
\mathsf{T}{:}p(X') \quad \mathsf{T}{:}q(X') \\
T_2
\end{array}
\end{array}$$

Based on the concept of a tableau k-containing another tableau, we can now define our notion of regularity.

Definition 16. *A (finite or infinite) sequence $(T_i)_{i\geq 0}$ of tableaux (and in particular a tableau proof) for a set $\mathfrak{F}$ of formulae is* k-regular *(for some $k \in \mathbb{N}$) if it does* not *contain tableaux T_j and T_i where $j < i$ such that T_j k-contains T_i (Def. 15). A sequence of tableaux that is not k-regular is* k-irregular.

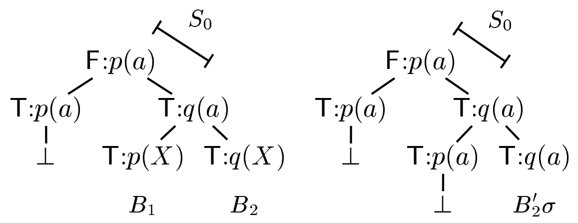
If a sequence $T_1, \ldots, T_n$ of tableaux is k-regular, T_{n+1} is a successor tableau of T_n, and the sequence $T_1, \ldots, T_n, T_{n+1}$ is k-irregular, then the expansion rule application that is used to construct T_{n+1} from T_n is said to be k-irregular (because it causes the irregularity). To check whether an expansion rule application is regular (which depends on the context in which it is used), it is sufficient to only consider those parts of the expanded tableau that are affected, i.e., the expanded branch and the formulae on the tableau that are associated with it. Checking whether an expansion rule application is regular or not does not involve unifiability tests because free variables may only be renamed but not instantiated with terms, i.e., checking regularity is not as complex as the problem of checking whether a tableau T_j *subsumes* T_{n+1} (which is NP-complete). Note also that associated formulae have to be considered but their position in the tableau is irrelevant—and thus (to a certain extend) the structure of the tableau.

If the notion of regularity as defined above is used to restrict proof search by only allowing regular sequences of tableaux, then completeness is preserved; i.e., our notion of regularity is not too restrictive.

Theorem 1. *Let $\mathcal{C}$ be a well-behaved free variable calculus; and let $k \in \mathbb{N}$ be the maximal size of the minimal premisses of all possible conclusions using some signature Σ. If there is a tableau proof for a set $\mathfrak{F} \subset Form(\Sigma^*)$ of formulae, then there is a regular tableau proof for $\mathfrak{F}$.*

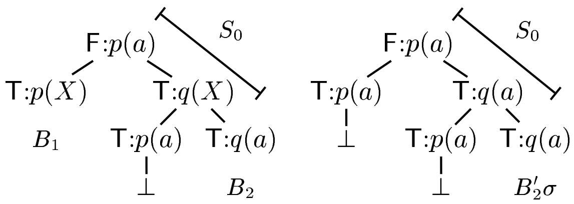
An important class of irregular expansion steps is the following: Assume that a branch B_1 of a tableau T is extended using a free variable conclusion $\langle C, \sigma \rangle$, and a branch $B_2'\sigma$ in the resulting tableau T' is contained in *all* branches B of T affected by the expansion, i.e., the branch B_1 (which is extended) and all other branches containing free variables that are instantiated by σ. This is in particular the case if $B_2'\sigma$ is contained in an initial sub-branch S_0 of T that ends above the first occurrence of any free variable in the domain of σ.

Example 8. Assume that the expansion rule of the calculus for FOL is used for closing the branch B_1 of the tableau T shown on the right; the tableau T' shown on the far right is constructed using the premiss consisting of the two literals $\mathsf{F}{:}p(a)$ and $\mathsf{T}{:}p(X)$ to derive the conclusion $\langle \{\{\bot\}\}, \{X \mapsto a\} \rangle$. This application belongs to the class of easy to detect irregular applications described above.



The right branch $B'_2\sigma$ of T' whose nodes are labelled with the formulae $\mathsf{F}{:}p(a)$ and twice $\mathsf{T}{:}q(a)$ is contained in the initial sub-branch S_0 of T whose nodes are labelled with $\mathsf{F}{:}p(a)$ and $\mathsf{T}{:}q(a)$; and S_0 ends above the first occurrence of X in T which is the only variable instantiated by σ. Intuitively, the application is useless because any closed sub-tableau that can be constructed below $B'_2\sigma$ can be constructed as well below both B_1 and B_2. Such an expansion rule application is irregular according to the definition of regularity that is usually given in the literature (e.g. [4]) since the branch $B'_2\sigma$ contains the same branch extension multiply.

Example 9. This example illustrates the difference between our and the classical notion of regularity. The situation is similar to the one in the previous example; but now, the sub-branch S_0 of T containing the branch $B'_2\sigma$ of the tableau T' does *not* end above the first occurrence of the variable X, which is instantiated by σ. Thus, this expansion rule application does not belong to the class of easy to detect irregular applications; moreover, it is indeed *regular* according to our definition, because the branch B_1 of T does *not* contain any of the branches of T'. According to the classical definition of regularity, however, this expansion rule application is *irregular* because the branch $B'_2\sigma$ of T' contains the extension $\{\mathsf{T}{:}q(a)\}$ twice.



As the above example demonstrates, the classical notion of regularity is in some cases more restrictive than ours. Whether using the classical, more restrictive notion destroys proof confluence depends on which other restrictions are imposed; however, to my knowledge nobody has been able to define a deterministic proof procedure for free variable tableau calculi using the classical notion. And the reason may very well be that it is too restrictive to preserve proof confluence when it is combined with other restrictions for ensuring fairness of the strategy (such as weight orderings). For calculi without free variables there is no difference between the two notions of regularity.

The following lemma formalises the consequences of regularity. An infinite regular sequence of tableaux contains infinitely many different formulae or, equivalently, if a regular sequence of tableaux only contains (up to renaming of free variables) finitely many different tableau formulae then it is finite.

Lemma 1. *Let $\mathcal{C}$ be a well-behaved free variable calculus; and let Γ be a finite set of tableau formulae. Then there is no infinite k-regular sequence $(T_i)_{i\geq 0}$ of tableaux ($k \in \mathbb{N}$) such that, for all $i \geq 0$, the tableau formulae in T_i are variants of tableau formulae in Γ.*

5 Weight Orderings

Weight orderings are the second important concept (besides regularity) on which our fairness strategy is based. The properties an ordering on tableau formulae for ensuring fairness must have are: (1) It is a well-ordering on the set of tableau formulae (up to renaming of free variables), i.e., it is well founded and there are only finitely many tableau formulae that are incomparable to a given tableau formula. (2) Proper instances of a tableau formula ϕ have a higher weight than ϕ. Intuitively, these are typical properties of orderings on tableau formulae that are defined by assigning a "weight" to the symbols of a signature (which is why we call them *weight* orderings).

Definition 17. *A* weight function w *assigns to each tableau formula a natural number (its* weight*) such that the following conditions are satisfied: (1) Given any tableau formula ϕ, there is* no infinite *set Ψ of tableau formulae such that (a) Ψ does not contain any ψ, ψ' that are identical up to renaming of free variables and (b) $w(\psi) \leq w(\phi)$ for all $\psi \in \Psi$. (2) If $X \in$ Var occurs in a tableau formula ϕ and the term t is not a free variable, then $w(\phi) < w(\phi\{X \mapsto t\})$. Given a weight function w, the* weight ordering $\leq_w$ *on tableau formulae (that is induced by w) is defined by $\phi \leq_w \psi$ iff $w(\phi) \leq w(\psi)$.*

The purpose of Condition 1 in the definition of weight functions is obvious; it ensure that if infinitely many different formulae are added to a tableau branch, then the maximal weight of formulae on the branch will sooner or later reach any value. Condition 2 might need some more explanation. It ensures that the weight increases in case no new formulae are appended to a branch but it is changed again and again by instantiating free variables; note that such a sequence of applications does not violate regularity because a new formula is created by each instantiation.

A weight ordering is extended to *sets* of tableau formulae by comparing the *maximal* weight of the formulae they contain. This extension is a well-ordering as well, provided the sets that are compared are only allowed to contain a certain number of variants of each tableau formula.

For every FOL-signature Σ a weight function can be defined by assigning a positive weight to each function and each predicate symbol in such a way that only a finite number of function and/or predicate symbols have the same weight (free variables are implicitly assigned the weight 0). The weight of a tableau formula ϕ is then defined to be the sum of the weights of all function and predicate symbols occurring in ϕ (multiple occurrences of the same symbol count multiply).

6 Deterministic Proof Procedures for Free Variable Calculi

In this section, we show that complete deterministic proof procedures can be defined for arbitrary well-behaved free variable calculi; such proof procedures can be used to perform depth first search for tableau proofs without backtracking. They are constructed using the notions of regularity and weight orderings as described in Sections 4 and 5.

It is indispensable that the calculi are well-behaved. They have to be non-structural because otherwise the order in which formulae are added to the tableau is not irrelevant, and completeness may be lost when formulae are added in the order of their weight. The calculi have to be monotonic since a deterministic procedure may execute redundant proof steps between useful ones, and the redundant formulae that are added must not prevent useful expansion rule applications later on. Analogously, the calculus must be monotonic w.r.t. substitution because there may be redundant instantiations of free variables. As said before, we only consider calculi in which minimal premisses are not of arbitrary size, i.e., do not contain more than k formulae for some fixed $k \in \mathbb{N}$.

To ensure that a deterministic proof procedure is complete, i.e., a proof is found if there is one, we demand that the constructed sequence of tableaux satisfies the following two conditions: (1) The sequence must be k-regular, i.e., the creation of a tableau that is k-contained in one of its predecessors is forbidden. (2) At each step, from all possible rule applications not violating Condition (1), an application is chosen that creates a successor tableau in which the maximal weight of formulae is as small as possible (i.e., successor tableaux are compared according to the maximal weight of the formulae they contain). If several rule applications satisfy these conditions, arbitrary heuristics may be employed to choose one of them; for example, rule applications creating less new sub-branches may be preferred.

Note that conclusions are not necessarily added to a tableau branch in the order defined by the maximal weight of their formulae, because a conclusion can only be added when its premiss is present on the branch and the weight of the formulae in the premiss may be higher than that of the conclusion.

To comply with the condition that *all* expansion rule applications adding formulae of less weight have to be executed before formulae of higher weight are added to a tableau, it may be necessary to expand branches that are already closed. That is not always redundant, because closed branches still contain useful information and can influence other branches by the substitutions that are applied when they are expanded (the first substitution that is applied to close a branch is not necessarily the "right one" that allows to complete the proof). If a closed branch has no free variables in common with other branches, it needs not be further expanded.

Unfortunately, the regularity condition as described above is difficult to implement; it requires to compare a tableau T_{n+1} with *all* its predecessors $T_1, \ldots, T_n$ and not only with the tableau T_n from which it is constructed. Such a regularity check is prohibitively expensive w.r.t. both space and time. Moreover, if an irregularity is encountered, i.e., if T_{n+1} is contained in one of the predecessor tableaux T_j, then other successor tableaux of T_j (besides T_{j+1}) have to be considered, which in a certain sense amounts to backtracking. The reason for this is the following: A tableau T_{n+1} that is contained in a tableau T_j does not have to be considered for proof search because all the proofs that may be constructed from T_{n+1} can be constructed from T_j. Now, if $j = n$, then we can just exclude the successor tableau T_{n+1} and be sure that if there is a proof derivable from T_{n+1} is is derivable from T_n without considering T_{n+1}. If, however, $j \neq n$, then the tableau proof that is known to be derivable from T_{n+1} and thus from T_j may not involve T_n but require to procceed with an alternative successor tableau T'_{j+1} different from T_{j+1}.

All these problems stem from the fact that a tableau T_j is not necessarily contained in its successor tableau T_{j+1} because free variable calculi are destructive and formulae occurring in T_j may not occur in T_{j+1} any more. Indeed, an irregular infinite sequence of tableaux can contain a cycle consisting of infinitely many tableaux T_{n_i} that all contain each other, where none of the irregular expansion steps is easy to detect, i.e., there is no tableau in the sequence that is contained in its immediate predecessor.

Example 10. Figure 2 shows one cycle in an infinite irregular sequence of tableaux that is constructed using the free variable calculus $\mathcal{C}_{\mathrm{FOL}}$ from Section 3.3. To construct this sequence, the following four steps are repeated again and again: (1) A conclusion of the form $\langle\{\{\mathsf{T}{:}r(Y)\},\{\mathsf{T}{:}s(Y)\}\}, id\rangle$ is derived from $\{\mathsf{T}{:}(\forall y)(r(y) \vee s(y))\}$ and is used to expand the rightmost branch of the tableau (for example, to derive tableau T_6 from tableau T_5). (2) The third branch from the right is closed and a variable X is instantiated with a (for example, to derive T_7 from T_6). (3) A conclusion of the form $\langle\{\{\mathsf{T}{:}p(X)\},\{\mathsf{T}{:}q(X)\}\}, id\rangle$ is derived from $\{\mathsf{T}{:}(\forall x)(p(x) \vee q(x))\}$ and is used to expand the rightmost branch of the tableau (for example, to derive T_8 from T_7). (4) The third branch from the right is closed and a variable Y is instantiated with b (for example, to derive T_9 from T_8). This completes the cycle; each tableau T_i in the sequence contains the tableau T_{i+4} for $i \geq 5$. No tableau in the sequence contains its immediate successor tableau, such that the cycle is difficult to detect. Note that for each expansion rule application, the most specific premiss is used.

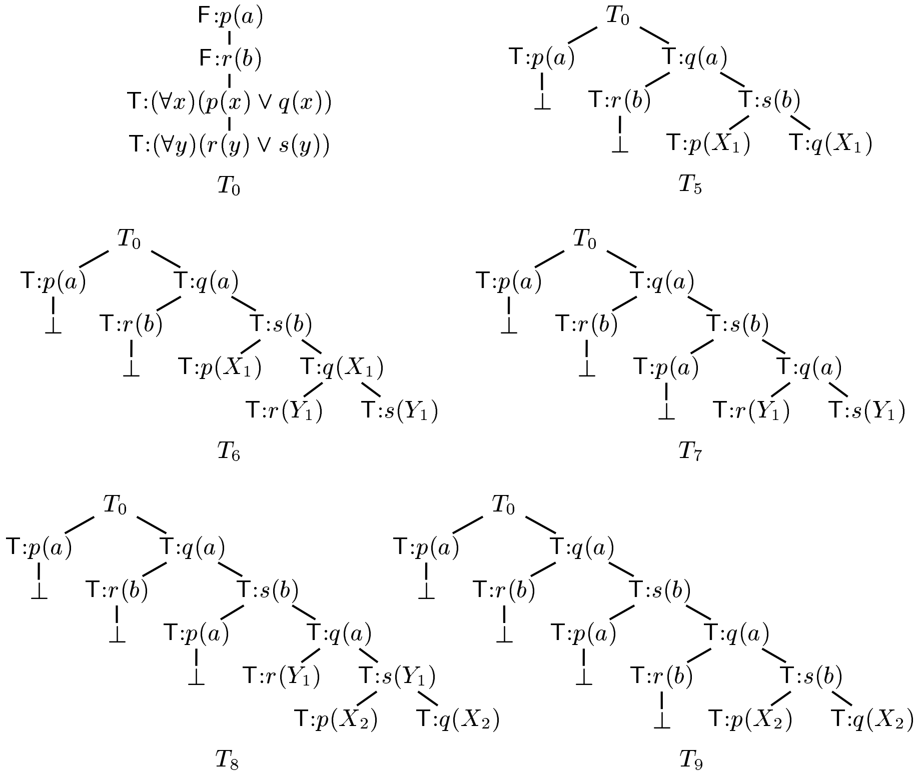


Fig. 2. An irregular sequence of tableaux containing a cycle; see Example 10.

The problems discussed above that all result from the fact that a tableau T_{i+1} may not contain its predecessor tableau T_i, can be solved by making the proof procedure less destructive in the following sense: We demand that immediately after an expansion step that destroys formulae, the expansion steps that are needed to recreate the destroyed formulae are executed. In the worst case, a new copy of the sub-tableau that was affected by the instantiation is created and appended to all sub-branches that have been affected. The result is a tableau T^+_{i+1} that contains both T_i and T_{i+1} and all the tableaux that occur as intermediate results during the reconstruction.

Example 11. The left branch of the tableau T_i in Figure 3 (a) is closed using the conclusion $\{\{\bot\}\}, \{X \mapsto a\}$. The result is the tableau T_{i+1} in Figure 3 (b), in which all formulae containing the free variable X have been destroyed. They are reconstructed by appending a copy of the sub-tableau $S(X)$ that consists of all formulae in T_i in which X occurs to all the branches in T_{i+1} from which formulae are missing; the resulting tableau T^+_{i+1} shown in Figure 3 (c) contains both T_i and T_{i+1}.

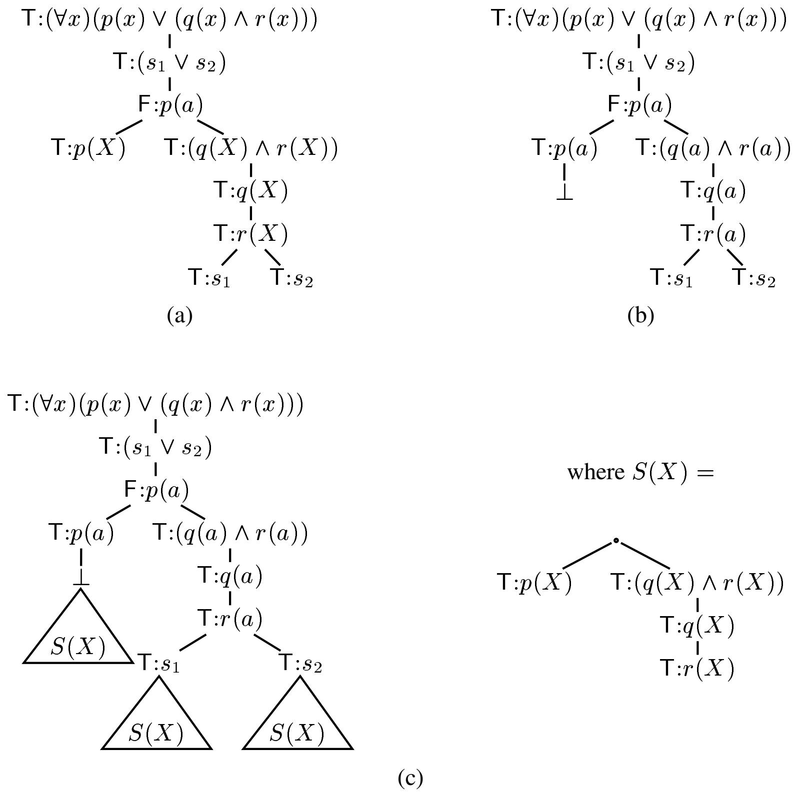


Fig. 3. A tableau reconstruction step (Example 11).

If a deterministic proof procedure executes a reconstruction step after each expansion rule application, then a sequence $T^+_1, T^+_2, \ldots$ of tableaux is constructed where

T_{i+1}^+ is constructed from T_i^+ by applying a regular expansion step and then reconstructing the destroyed formulae. To check whether such a sequence is regular, it is sufficient to test whether the immediate successor tableau T_{i+1} of T_i^+ is contained in T_i^+. The earlier predecessors do not have to be considered as they are all contained in T_i^+. Theorem 2 below states completeness of such a proof procedure; it is the main theorem of this paper.

Theorem 2. *Let $\mathcal{C}$ be a free variable calculus that is well-behaved (Def. 10); let $k \in \mathbb{N}$ be the maximal size of the minimal premisses of all possible conclusions in $\mathcal{C}$; and let w be a weight function (Def. 17).*

If there is a tableau proof for a set $\mathfrak{F} \subset Form(\Sigma^*)$ of formulae, then every sequence $(T_i^+)_{i\geq 1}$ of tableaux for $\mathfrak{F}$ constructed as described below contains a closed tableau T_n^+ ($n \in \mathbb{N}$). The tableau T_1^+ is an initial tableau for $\mathfrak{F}$ and for all $i > 1$ the tableau T_{i+1}^+ is constructed as follows:

1. *T_{i+1} is a successor tableau of T_i^+ such that (a) T_{i+1} is not k-contained in T_i^+, and (b) there is no successor tableau T'_{i+1} of T_i^+ that satisfies condition (a) and has a smaller maximal formula weight then T_{i+1} (w.r.t. the weight function w).*
2. *Let $\langle C_i, \tau_i \rangle$ be the conclusion derivable from some premiss on T_i^+ that is used to construct T_{i+1}, and let S_i be the minimal sub-tableau of T_{i+1} that contains all occurrences of free variables instantiated by τ_i. The tableau T_{i+1}^+ is constructed from T_{i+1} by repeatedly executing all expansion rule applications needed to construct the sub-tableau S_i and thus appending S_i at the end of each branch going through the sub-tableau in T_{i+1} that results from applying τ_i to S_i.*

Example 12. An example for a tableau proof construction as described in this section is shown in Figure 4. The construction starts with the initial tableau T_1. At this point only one rule application is possible, which results in the tableau T_2. Then there are several possibilities to proceed; the left branch of T_2 can be closed instantiating X_1 with either a or b and the right branch can be closed instantiating X_1 with b. We assume that according to the weight ordering, $p(a) \leq_w p(b)$ and $q(a) \leq_w q(b)$. Consequently, the "bad" instantiation $\{X_1 \mapsto a\}$ is preferred and the tableau T_3 is constructed, because the maximal weight of its formulae is less than that of the formulae in the alternative tableaux. Since the free variable X_1 is instantiated, a reconstruction step is required; the result of that step is the tableau T_4. Now there are again several possibilities. If the weight of formulae were the only criterion, then the tableau T'_5 would have to be derived from T_4, repeating the useless instantiation of a free variable with a. However, deriving T'_5 form T_4 would be an irregular expansion step as T'_5 is contained in T_4 (it is easy to check that each branch of T_4 contains one of the branches of T'_5). Therefore, the tableau T_5 is derived instead of T'_5; and the free variable X_3 is instantiated with b instead of a. Again, a reconstruction step is required, which results in the tableau T_6. From T_6 the closed tableau T_7 can easily be constructed (in two steps).

A proof procedure satisfying the conditions of Theorem 2 constructs a regular sequence $T_1^+, T_2^+, \ldots$ of tableaux such that $T_i^+ \subseteq_k T_{i+1}^+$ for all $i \geq 1$. Such a procedure simulates—in a certain sense—a depth first iterative deepening search (as described in the introduction). The weight of the formulae that can occur in the tableaux increases

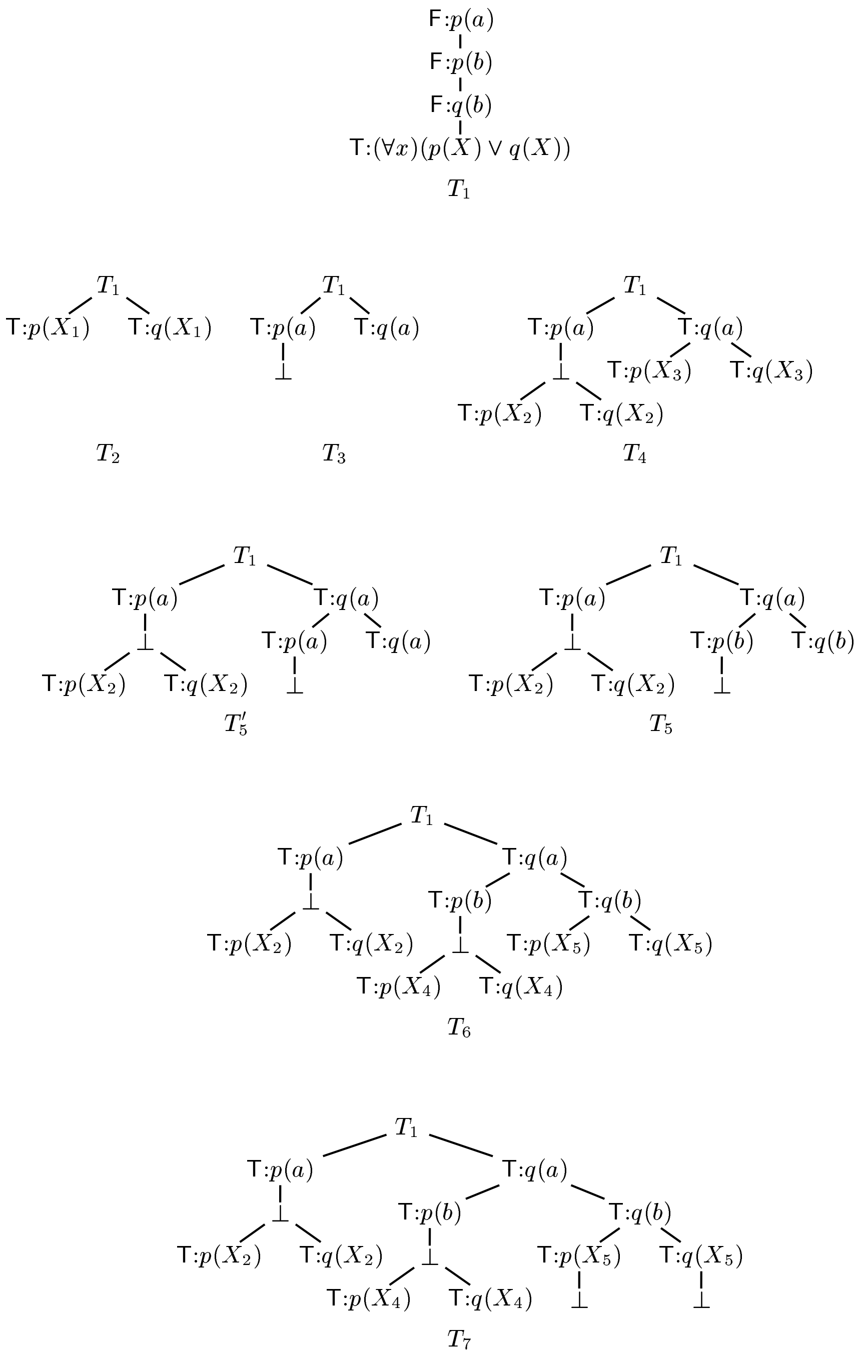


Fig. 4. The tableau proof described in Example 12.

stepwise. If some (possibly irregular) tableau proof exists that does not contain formulae of weight bigger than $w_{\max}$, then there is a closed tableau T_n^+ that is the last in the constructed sequence not containing formulae of weight bigger than some $w_{\max}^+ \in \mathbb{N}$. It contains all tableaux that can be constructed from formulae ϕ of weight $w(\phi) \leq w_{\max}$. The big advantage of this simulated DFID over classical DFID search based on backtracking is that the tableau T_n^+ is a very compact representation of the search space. All the information that is contained in tableaux whose formulae are of weight less than $w_{\max}$ is present in the single structure T_n^+; and all the tableaux in the search space that are identical or in some way symmetrical to each other are represented by only one sub-tableau of T_n^+. Since no backtracking occurs, no information that has been derived is ever lost. There may be parts of the tableau T_n^+ that represent redundant information and are therefore useless (i.e., non closed-subtableau that should not have been created); but these are not harmful as they can be removed using the *pruning* technique (see [4]).

The method of constructing deterministic proof procedures for free variable calculi described in this paper is compatible with all search space restrictions with which the calculus remains well-behaved. Important examples for such restrictions include selection functions [8, 10] and the weak connectedness condition.

References

1. P. Baumgartner. Fairness strategies in hyper tableaux. Talk given as part of the Tele Seminar *Automated Theorem Proving and Applications* jointly held at the University of Karlsruhe and the University of Koblenz, 1998.
2. P. Baumgartner, N. Eisinger, and U. Furbach. A confluent connection calculus. Draft, 1998.
3. B. Beckert. *Integration and Unifying Methods of Tableau-based Theorem Proving*. PhD thesis, Universität Karlsruhe, Fakultät für Informatik, July 1998.
4. B. Beckert and R. Hähnle. Analytic tableaux. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction — A Basis for Applications*, volume I: Foundations. Kluwer, 1998.
5. B. Beckert, R. Hähnle, and P. H. Schmitt. The even more liberalized δ-rule in free variable semantic tableaux. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Proceedings, 3rd Kurt Gödel Colloquium (KGC), Brno, Czech Republic*, LNCS 713. Springer, 1993.
6. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, New York, second edition, 1996.
7. D. Gabbay. *Labelled Deductive Systems*. Number 33 in Oxford Logic Guides. Oxford University Press, 1996.
8. R. Hähnle and S. Klingenbeck. A-ordered tableaux. *Journal of Logic and Computation*, 6(6):819–834, 1996.
9. R. Hähnle, N. Murray, and E. Rosenthal. Completeness for linear regular negation normal form inference systems. In *Proc., Int. Symp. on Methodologies for Intelligent Systems (ISMIS), Charlotte, NC, USA*, LNCS. Springer, 1997.
10. R. Hähnle and C. Pape. Ordered tableaux: Extensions and applications. In *Proceedings, TABLEAUX, Pont-á-Mousson, France*, LNCS 1227, pages 173–187. Springer, 1997.
11. R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

Acknowledgement I thank Reiner Hähnle and Peter H. Schmitt for fruitful discussions and comments on earlier versions of this paper.