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

— Extended Abstract —

Bernhard Beckert
University of Karlsruhe, D-76128 Karlsruhe, Germany. Email: beckert@ira.uka.de

Abstract

We analyse the problem of constructing a deterministic proof procedure for free variable clausal tableaux that performs depth-first proof search without backtracking; and we present a solution based on a fairness strategy. That strategy uses weight orderings *and a notion of* tableau subsumption *to avoid proof cycles and it employs* reconstruction steps *to handle the destructiveness of free variable calculi.*

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. As an example, we present a solution for first-order clausal tableaux that is based on a fairness strategy. That strategy uses *weight orderings* and a notion of *tableau subsumption* to avoid proof cycles and it employs *reconstruction steps* to handle the destructiveness of clausal tableaux.

First-order clausal tableaux are *proof-confluent*, i.e., every tableau for an unsatisfiable clause set can be completed to a proof. They are, however, a *destructive* calculus because all occurrences of a (free) variable in a tableau have to be instantiated by the same term and, thus, a rule application can make another rule application impossible.

The proof search space can be visualised as a search tree where each possible choice of the next rule application to a tableaux T creates a node with as many successor nodes as T has different successor tableaux (Fig. 1). Since we use a proof-confluent calculus, all path are either infinite or end in a node that is labelled with a proof, i.e., a closed tableau.

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.

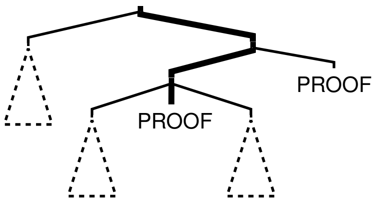

Figure 1. A proof search tree.

As fairness strategies that allow depth-first search are difficult to construct for first-order clausal tableaux, most automated deduction systems use breadth-first search. It 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 their construction requires less rule applications and are thus easier to find); and breadth-first search is "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. Even for small i, it is 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 initial, partial search space consisting of all the 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 [4] 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.

Figure 2 shows how the different search strategies traverse the search space. The coloured part has to be searched before a proof is found. The form of the search space visualises its exponential growth.

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 clausal tableau calculi, which are *destructive* (even if they are proof-confluent). Applying a substitution may destroy literals on a tableau that are needed for the proof, such that they have to be deduced again.

Up to now there was no practical solution to the problem of constructing a deterministic proof procedure for free variable clauses tableaux that performs depth-first search and is complete, i.e., that never fails to find a proof if there is one. Such procedures were only known for the special case where tableaux are expanded without instantiating variables and only a single substitution is finally applied that is known to allow to close all branches simultaneously. Solving a similar problem, Baumgartner et al. [1] recently described a depth-first proof procedure for a *connection calculus*.

We propose in this paper a deterministic search strategy that is based upon:

- A *tableau subsumption* relation to detect "cycles" in the search (i.e., to make sure that it is not possible to deduce the same literals or sub-tableaux again and again).
- *Weight orderings* that assign each literal a "weight" in such a way that there are only finitely many different literals (up to variable renaming) of a certain weight; thus, if literals with lesser weight are deduced first, then sooner or later each possible conclusion is added to all branches containing its premiss.
- *Reconstruction steps* to handle the destructiveness of free variable clausal tableaux. Immediately after a rule application that destroys literals, the construction steps that are needed to recreate the destroyed sub-tableaux are executed.

The main difficulty is to define a tableau subsumption relation 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.

Our fairness strategy considers the whole tableau tree (and not only a single branch) both for the subsumption check 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 clausal tableaux were known up to now. There is trivially a (non-practical) deterministic proof procedure for all proof-confluent calculi, namely a procedure performing a *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 construction 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 tableau rule applications. The size of the proofs that are found (and thus the number of construction steps) is at most that of the proofs 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: In Section 2, we describe the calculus of clausal tableaux. After introducing our notion of tableau subsumption in Section 3 and that of weight orderings in Section 4, our method for constructing deterministic proof procedures for free variable clausal tableaux is presented in Section 5.

Due to space restrictions, all proofs are omitted; they can be found in [2].

2 First-order Clausal Tableaux

The notions of *free* and *bound variable*, *term*, *atom*, *literal*, and *substitution* are defined as usual. We use x, y, z etc. to denote quantified variables and X, Y, Z etc. to denote free variables. The logical constants $\top$ (true) and $\perp$ (false) are considered to be literals (but not atoms). The complement of a literal L is denoted with $\overline{L}$. A *variable renaming* is a substitution that replaces all variables by distinct variables that are "new" w.r.t. the context.

A *clause* C is a first-order formula of the form

$$(\forall x_1) \cdots (\forall x_n)(L_1 \vee \cdots \vee L_r)$$

where the L_i are literals and $x_1, \ldots, x_n$ are all variables occurring in $L_1, \ldots . L_r$. A *new instance* of C is a formula $(L_1 \vee \cdots \vee L_r)\sigma$ where σ is a variable renaming.

We use the *weak* connectedness condition where a clause used for expansion must have a link into the branch being expanded (the strong connectedness condition, where the clause must be linked to the leaf of the branch, is not used as it destroys proof confluence).

A *clausal tableau* for a set S of clauses is built by a sequence of applications of the following construction rules. Each rule has a premiss (a set of literals) and a conclusion (consisting of a set of literals and a substitution).

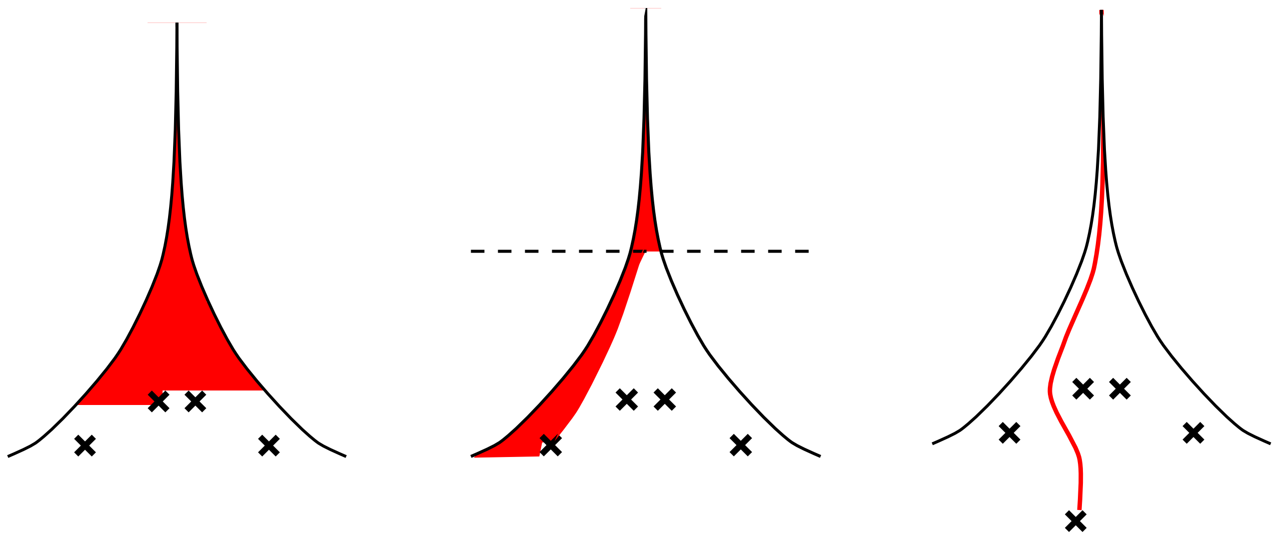

Figure 2. Comparison of proof search strategies.

Initialisation: If $L_1 \vee \ldots \vee L_r$ is a new instance of a clause in S, then the tree is a tableau for S that consists of the root node $\top$ and r sub-branches with the single nodes $L_1, \ldots, L_r$. (In this case, the premiss is empty and the conclusion is $\langle \{L_1, \ldots, L_r\}, id \rangle$.)

Expansion: If T is a tableau for S, B is a branch of T, L is a literal on B, $L_1 \vee \cdots \vee L_r$ is a new instance of a clause in S, and $L, \overline{L_j}$ are unifiable (for some $1 \leq j \leq r$), then a tableau T' is a tableau for S if obtained by extending B with r nodes $L_1, \ldots, L_r$. (In this case, the premiss is $\{L\}$ and the conclusion is $\langle \{L_1, \ldots, L_r\}, id \rangle$,)

Closure: If T is a tableau for S, B is a branch of T, L, L' are literals on B, and $L, \overline{L'}$ are unifiable with MGU σ, then T' is a tableau for S if obtained by appending $\perp$ to B and applying σ to each node of T. (In this case, the premiss is $\{L, L'\}$ and the conclusion is $\langle \{\perp\}, \sigma \rangle$.)

Note, that a branch is closed by adding the special literal $\perp$; therefore, branch closure can be considered to be a special kind of branch expansion.

A tableau T is *closed* if all its branches are closed, i.e., contain $\perp$. A *tableau proof* for (the unsatisfiability of) a clause set S is a tableau for S that is closed.

Clausal tableaux as defined above are a complete and proof-confluent calculus.

We use a slightly non-standard definition of the notion of successor tableau: A tableau T' is a *successor tableau* of a tableau T if it is constructed from T by one *or more* "identical" rule applications, i.e., there are (1) different branches $B_1, \ldots, B_n$ ($n \geq 1$) of T, (2) premisses Π_i on the B_i that are identical up to variable renaming, (3) a (single) conclusion $\langle C, \sigma \rangle$ such that $\Pi_i \sigma = \Pi_j \sigma$ ($1 \leq i, j \leq n$), and T' is constructed from T by extending each of the branches B_i with the literals in C and applying the substitution σ to T.

3 Tableau Subsumption Relation

Assume that a sequence $T_1, \ldots, T_n$ of tableaux has already been constructed. A rule application to T_n is forbidden if the successor tableau T_{n+1} is *subsumed* by one of the predecessor tableaux T_j—in particular, if T_{n+1} is subsumed by 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 subsume a tableau T_{n+1} iff each branch of T_j subsumes 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 subsumes a branch of T_{n+1}.

When does a tableau *branch* subsume another branch? A first approximate answer to that question is: A branch B subsumes a branch B' if B contains a variant of each literal occurring on B'. That, however, is an over-simplification; three additional aspects have to be taken into concern.

First additional aspect. For a branch B to subsume a branch B', it is in general *not* sufficient if the branch B contains *one* variant of each literal L occurring in B', namely in case B' contains two variants of L that are all both needed to close the branch. However, since the premiss for a single rule application contains at most two literals, it is sufficient if B contains a variant of each set of (at most) two literals occurring on B'. This implies that at most two variants of each literal on B' are needed on B (where however, as described below, literals may have to be considered to be effectively different although they are variants of each other on first sight).

Example 1. If the literals $\neg p(X)$, $p(f(X))$, $\neg p(X')$, and $p(f(X'))$ occur on B' whereas the branch B only contains

$\neg p(X)$ and $p(f(X))$ (and B and B' are otherwise identical), then B contains a variant of each literal 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.

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

Definition 1. Literals L and L' are *associated* if there is a variable occurring in both L and L'. The set of all literals in a tableau T that are associated with a literal L, excluding L itself, is denoted with $Assoc(T, L)$. Accordingly, if Φ is a set of literals, then $Assoc(T, \Phi)$ is the variable set $\left(\bigcup_{L \in \Phi} Assoc(T, L)\right) \setminus \Phi$.

Associated literals play a role because the ordering of tableau rule applications used by a deterministic proof procedure as described in Section 5 has to take all literals into account that are generated by an application. So, if $L(X)$ is a premiss for a certain tableau rule application that leads to the instantiation of X with a term t and there is a literal $L'(X)$ on the tableau, then that application will generate the new literal $L'(t)$; and the form of $L'(t)$—and thus the form of the associated literal $L'(X)$—affects the choice of the application.

Third additional aspect. As said above, a tableau T subsumes a tableau T' if for each branch B in T there is a branch B' in T' such that B subsumes B'. That includes the possibility that two different branches B_1 and B_2 of T are assigned the same branch B'. In that case there is for each set Φ' (of at most two literals on B') a literal set Φ_1 on B_1 and a literal set Φ_2 on B_2 that are variants of Φ'. The basic idea behind the definition of our subsumption relation implies that every possible rule application on branch B' with the premiss Φ' can as well be applied—simultaneously—on the branches subsuming B' with the premisses Φ_1 resp. Φ_2. That, however, requires the two variable renamings constructing Φ' from Φ_1 resp. Φ_2 to be compatible. The same holds if B' is assigned to more than two branche in T.

Formal definition of the subsumption relation. We now formally define our tableau subsumption relation. It is transitive und reflexive.

Definition 2. Let T and T' be tableaux that do not have any variables in common. The tableau T *subsumes* the tableau T' if

i. each branch B of T can be assigned a branch B' of T'

ii. and then—for each pair B, B' respectively—each set Φ' of at most two literals on B' can be assigned a set Φ of literals B and a variable renaming π

such that:

1. The following holds for each of the Φ, Φ' and π:
   (a) $\Phi\pi = \Phi'$;
   (b) for each of the literals L in $Assoc(T, \Phi)$ there is (at least) one literal L' in $Assoc(T', \Phi')$ such that $L\pi$ and L' are identical up the renaming of variables *not* occurring in $\Phi\pi$ resp. Φ'.
2. If a branch B' of T' is assigned to different branches $B_1, \ldots, B_s$ of T ($s \geq 2$), then, for all Φ' on B', the variable renamings $\pi_1, \ldots, \pi_s$ assigned to Φ' in connection with $B_1, \ldots, B_s$ are compatible in the following way: there is a substitution π such that the restriction of π to the variables occurring in $\Phi \cup Assoc(T, \Phi)$ is identical to π_i ($1 \leq i \leq s$).

Now, let T and T' be tableaux that have variables in common; and let ρ be a variable renaming such that T und $T'\rho$ do *not* have any variables in common. Then, T *subsumes* T' iff T subsumes $T'\rho$.

If a tableau T subsumes a tableau T', then each branch B of T is assigned a branch B' of T'. In that case, we say that B subsumes B'.

Completeness of clausal tableaux is preserved if the tableau subsumption relation is used for restricting the search space: Given a partial proof $T_0, \ldots, T_i$ it is forbidden to derive a successor tableau T_{i+1} from T_i that is subsumed by any of the tableau $T_0, \ldots, T_i$. On the other hand, this restriction is strong enough to ensure that every sequence of tableaux built accordingly, i.e., every tableau sequence not containing a tableau that is subsumed by one of its predecessors, has the following property: If the sequence is infinite, then it contains infinitely many different literals or, equivalently, if the sequence only contains finitely many different literals (up to the renaming of variables) then it is finite.

To check whether a tableau T subsumes one of its successor tableaux T' and, thus, whether the rule application deriving T' from T is allowed, it is sufficient to only consider those parts of the tableaux that are affected, i.e., the expanded branch and the formulae on the tableaux that are associated with it. The check does not involve unifiability tests because free variables may only be renamed but not instantiated with terms.

Example 2. Let $\Phi = \{p(X)\}$ and $\Phi' = \{p(X')\}$; moreover let $Assoc(T, \Phi)$ consist of $q(X, Y_1)$ and $q(X, Y_2)$. Then, Condition 1 (a) in Definition 2 is, for example, satisfied if $Assoc(T', \Phi') = \{q(X', Y')\}$. But it is neither satisfied if $Assoc(T', \Phi') = \emptyset$ nor if $Assoc(T', \Phi') = \{q(Y', X')\}$ (because to make $q(X', Y_1)$ and $q(Y', X')$ identical would require to rename the variable X' that occurs in Φ'.

Example 3. The tableau T_1 in Figure 3 subsumes each of the tableau T_1', T_2', T_3'. The tableau T_2 subsumes only T_1'.

Example 4. Neither of the two tableaux in Figure 4 subsumes the other one. The tableau T_1 on the left does not subsume the tableau T_2 on the right because the (single) branch of T_2 contains an additional literal; and, although a variant of each literal set on T_1 occurs on T_2, the tableau T_2 does not subsume T_1 since for $r(X') \in Assoc(T_2, q(X'))$ there is no corresponding element in $Assoc(T_1, q(X))$ and, thus, Condition 1 (a) in Definition 2 is not satisfied.

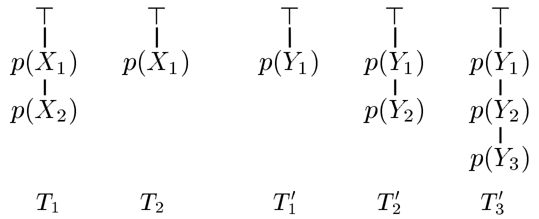

Figure 3. The tableaux from Example 3.

Figure 4. The tableaux from Example 4.

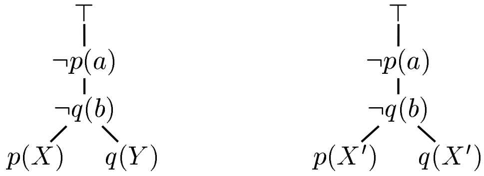

Figure 5. The tableaux from Example 5.

Example 5. The tableau T_1 in Figure 5 on the left subsumes the tableau T_2 on the right. But T_2 does *not* subsume T_1 because the literals $p(X')$ and $q(X')$ in T_2 are associated, whereas the corresponding literals in T_1 are *not* associated.

Indeed, 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.

Example 6. The tableau T in Figure 6 on the upper left does *not* subsume the tableau T' on the right. That would only be possible if the branches B_1, B_2 of T would *both* subsume the single branch of T'. Both B_1 and B_2 contain a variant $p(X,Y)$ resp. $p(Y,X)$ of the (single) literal $p(U,V)$ on the branch of T'. But the required variable renamings $\{X \mapsto U, Y \mapsto V\}$ and $\{X \mapsto V, Y \mapsto U\}$ are not compatible, which violates Condition 1 (a) in the definition of the subsumption relation (Def. 2).

This problem does not occur with the tableau shown in Figure 6 on the lower left. It subsumes T' because the two required variable renamings $\{X_1 \mapsto U, Y_1 \mapsto V\}$ and $\{X_2 \mapsto U, Y_2 \mapsto V\}$ are compatible.

Example 7. Consider the tableau T_1 shown on the left in Figure 7. The rule application that derives the conclusion $\langle\{\bot\}, id\rangle$ from the premiss $\{\neg p, p\}$ can be used to close both of its branches. Closing the tableau requires two consecutive applications. However, the intermediate tableau that results from closing the left branch (in the middle of Figure 7) is subsumed by T_1 because both branches of T_1 subsume the right (not yet expanded) branch of T_2. Thus, this first rule application is not allowed. The tableau T_3, however, that results from closing both branches (shown in the right in Figure 7) is neither subsumed by T_2 nor by T_1.

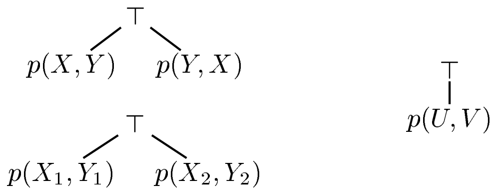

Figure 6. Tableaux from Example 6.

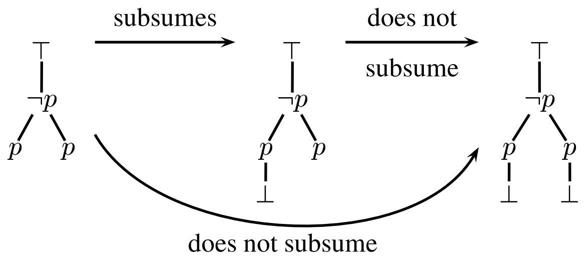

Figure 7. Tableau from Example 7.

Indeed, since both rule applications use the same premiss and conclusion, T_3 is by definition a successor tableau of T_1 (without considering the intermediate step), and deriving T_3 from T_1 is an allowed rule application.

Example 8. An important type of tableau construction steps that generate a tableau T' subsumed by it predecessor T and that are, therefore, forbidden, is the following: Assume that a branch B_1 of T is extended using a conclusion $\langle C, \sigma\rangle$, and a branch $B_2'\sigma$ in the resulting tableau T' is subsumed by *all* branches B of T affected by the rule application, i.e., the branch B_1 (which is extended) and all other branches containing variables that are instantiated by σ. This is in particular the case if $B_2'\sigma$ is "contained" in an initial sub-branch R_0 of T that ends above the first occurrence of any free variable in the domain of σ.

As an example consider the tableau T shown in Figure 8 on the left, and assume that its branch B_1 is closed using the premiss consisting of the two literals $p(a)$ and $p(X)$ to derive the conclusion $\langle\{\bot\}, \{X \mapsto a\}\rangle$. The right branch $B_2'\sigma$ of the resulting tableau T' (shown in Figure 8 on the right) whose nodes are labelled with the literals $p(a)$ and twice $q(a)$ is "contained" in the sub-branch R_0 of T whose nodes are labelled with $p(a)$ and $q(a)$; and R_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.

A forbidden rule application as described above is irregular according to the definition of regularity that is usually given in the literature (e.g. [3]) since the branch $B_2'\sigma$ contains the same branch extension multiply.

4 Weight Orderings

Weight orderings are the second important concept (besides the concept of tableau subsumption) on which our fairness strategy is based. The properties an ordering on literals for ensuring fairness must have are: (1) It is a well-ordering

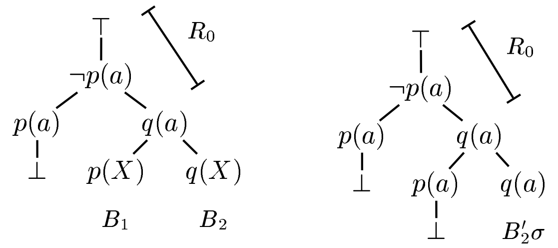

Figure 8. Tableaux from Example 8.

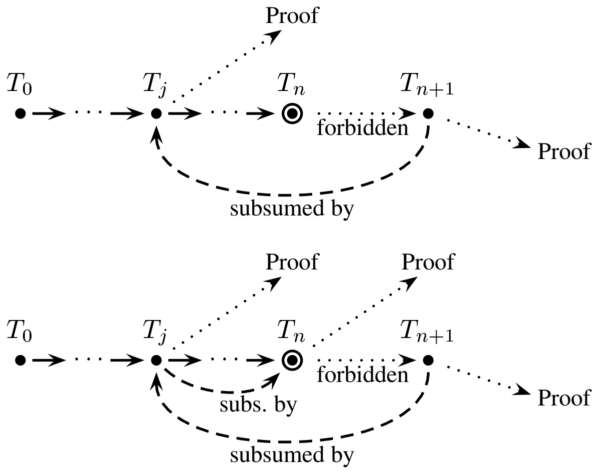

Figure 9. Proof search with a destructive calculus (top) and a non-destructive calculus (bottom).

on the set of literals (up to renaming of free variables), i.e., it is well-founded and there are only finitely many literals that are incomparable to a given literal. (2) Proper instances of a literal L have a higher weight than L. (3) Literals that are identical up to variable renaming have the same weight. Intuitively, these are typical properties of orderings on literals that are defined by assigning a "weight" to the symbols of a signature (which is why we call them *weight* orderings).

A weight ordering is extended to *sets* of literals by comparing the *maximal* weight of the literals 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 literal.

5 Deterministic Proof Procedures for Clausal Tableaux

In this section, we define a (class of) complete deterministic proof procedure(s) for clausal tableaux; this proof procedure can be used to perform depth-first search for proofs without backtracking. It is constructed using the notions of subsumption and weight orderings as described in Sections 3 and 4.

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 creation of a tableau that is subsumed by 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 literals is as small as possible (i.e., successor tableaux are compared according to the maximal weight of the literals 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 literals because a literal L can only be added if the necessary premiss Π is present on the branch; and the weight of the literals in Π may be higher than that of L. Also, when a conclusion is added, is controlled by its literal with the highest weight such that literals with a lower weight that can only be added as part of a conclusion containing other literals of higher weight are added to the tableau later.

To comply with the condition that *all* rule applications adding literals of less weight have to be executed before literals 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 restriction of the search space 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 derived. Such a subsumption check is prohibitively expensive w.r.t. both space and time. Moreover, if a subsumption is encountered, i.e., if T_{n+1} is subsumed by 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 subsumed by 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} then it 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}. This situation is shown schematically in Figure 9 (top).

All these problems stem from the fact that a tableau T_j is not necessarily subsumed by its successor tableau T_{j+1} because the clausal tableau calculus is destructive and literals occurring in T_j may not occur in T_{j+1} any more. However, if we make the calculus *weakly non-destructive* in the sense that a tableau is always subsumed by all its successor tableaux, then we have the situation shown in Figure 9 (bottom). Now, the tableau T_j is subsumed by the tableau T_n ensuring that every proof that can be constructed from T_{n+1} can as well be constructed from T_n—without deriving T_{n+1}

as an intermediate result. In a certain sense, a (weakly) non-destructive calculus is proof-confluent w.r.t. the restricted search space (where no tableaux subsumed by a predecessor are allowed).

To make clausal tableaux weakly non-destructive, i.e., to make sure that a tableau T_{i+1} always subsumes its predecessor tableau T_i, we impose the following additional restriction on the proof construction: Immediately after a tableau construction step destroying literals, the construction steps that are needed to recreate the destroyed literals must be executed. In the worst case, a new copy of the sub-tableau that was affected by the variable instantiation is created and appended to all sub-branches that have been affected. The result is a tableau T_{i+1}^+ that subsumes both T_i and T_{i+1} and all the tableaux that occur as intermediate results during the reconstruction.

Example 9. Consider the clause set S consisting of the clauses $(\forall x)(p(x) \vee q(x))$, $(\forall x)(\neg q(x) \vee r(x))$, and $s_1 \vee s_2$. Figure 10 (a) shows a tableau T_i for S. The left branch of T_i is closed using the conclusion $\langle\{\bot\}, \{X \mapsto a\}\rangle$. The result is the tableau T_{i+1} in Figure 10 (b), in which all literals containing the free variable X have been destroyed. They are reconstructed by appending a copy of the sub-tableau $R(X)$ that consists of all literals in T_i in which X occurs to all the branches in T_{i+1} from which literals are missing; the resulting tableau T_{i+1}^+ (shown in Figure 10 (c)) subsumes both T_i and T_{i+1}.

If a deterministic proof procedure executes a reconstruction step after each tableau rule application, then a sequence $T_1^+, T_2^+, \ldots$ of tableaux is constructed where T_{i+1}^+ is derived from T_i^+ by executing a construction step (that does not lead to a tableau subsumed by its predecessor) and then reconstructing the destroyed literals. To ensure that such a sequence meets all conditions, it is sufficient to test whether the immediate successor tableau T_{i+1} of T_i^+ is subsumed by T_i^+. The earlier predecessors do not have to be considered as they are all subsumed by T_i^+. Theorem 3 below states completeness of such a proof procedure; it is the main theorem of this paper.

Theorem 3. *If a clause set S is unsatisfiable, then every sequence $(T_i^+)_{i\geq 1}$ of tableaux for S that is constructed as described below contains a closed tableau T_n^+ ($n \in \mathbb{N}$).*

The tableau T_1^+ is an initial tableau for S. And for all $i > 1$ the following holds:

1. *T_{i+1} is a successor tableau of T_i^+ (see Sect. 2) such that (a) T_i^+ does not subsume T_{i+1} and (b) there is no successor tableau T'_{i+1} of T_i^+ that satisfies Condition (a) and has a smaller maximal literals weight than T_{i+1} (w.r.t. an arbitrary but fixed weight ordering).*

2. *Let $\langle C_i, \tau_i\rangle$ be the conclusion (derived from some premiss on T_i^+) that is used to construct T_{i+1}; and let R_i be the minimal sub-tableau of T_{i+1} that contains all occurrences of the variables instantiated by τ_i. The tableau T_{i+1}^+ is constructed from T_{i+1} by (repeatedly) executing all rule applications that are necessary to generate R_i; R_i is appended to all branches that go through the sub-tableau of T_{i+1} corresponding to R_i (which results from applying τ_i to R_i).*

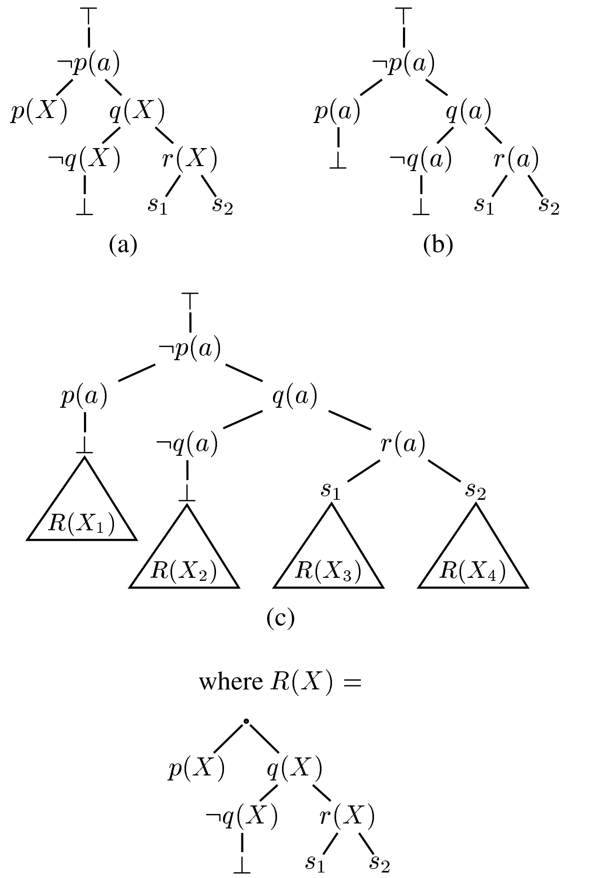

Figure 10. A tableau reconstruction step (Example 9).

Example 10. As an example for the proof construction as described in this section, Figure 11 shows a tableau proof for the clause set consisting of the clauses $\neg p(a)$, $\neg p(b)$, $\neg q(b)$, $(\forall x)(p(x) \vee q(x))$. The proof construction starts with adding the unit literals to the initial tableau; the result is the 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 literals is less than that of the literals in the alternative tableaux. Since the 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 literals were the only criterion, then the tableau T'_5 would have to be derived from T_4, repeating the useless instantiation of a variable with a. However, deriving T'_5 form T_4 is not allowed as T'_5 is subsumed by T_4 (it is easy to check that each branch of T_4 subsumes one of the branches of T'_5). Therefore, the tableau T_5 is derived instead of T'_5; and the 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.

A proof procedure as described in Theorem 3 constructs a sequence $T_1^+, T_2^+, \ldots$ of tableaux such that no tableau is subsumed by any of its predecessors and all tableaux are subsumed by their successors. Such a procedure simulates (in a certain sense) a depth-first iterative deepening search (as described in the introduction). The weight of the literals that can occur in the tableaux increases stepwise. If some (unrestricted) tableau proof exists that does not contain literals of weight bigger than $w_{\max}$, then there is a closed tableau T_n^+ that is the last in the constructed sequence not containing literals of weight bigger than some $w_{\max}^+ \in \mathbb{N}$. It subsumes all tableaux that can be constructed from literals L of weight $w(L) \leq w_{\max}$. 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 literals 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 sub-tableau that should not have been created); but these are not harmful as they can be removed using the *pruning* technique (see [3]).

The deterministic proof procedures for clausal tableaux described in this paper is compatible with all search space restrictions with which the calculus remains proof-confluent such as, for example, selection functions [5, 6].

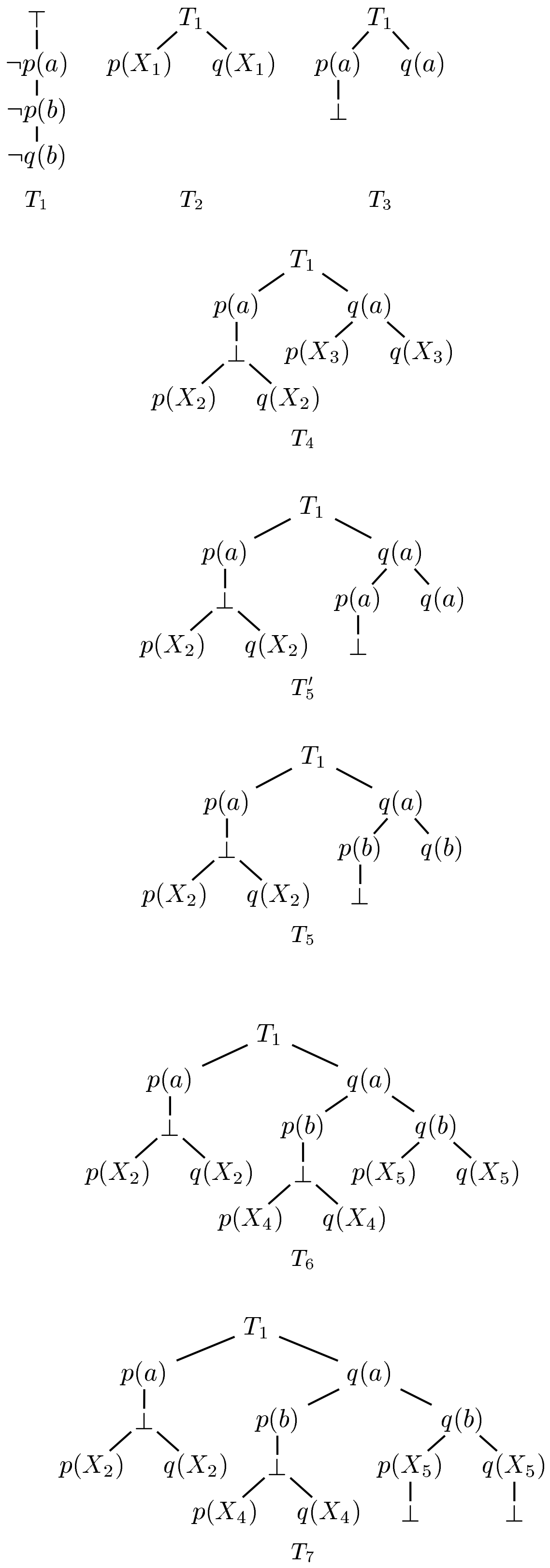

Figure 11. The tableau proof described in Example 10.

References

[1] P. Baumgartner, N. Eisinger, and U. Furbach. A confluent connection calculus. In H. Ganzinger, editor, *Proceedings, Conference on Automated Deduction (CADE), Trento, Italy*, LNCS 1632, pages 329–343. Springer, 1999.

[2] B. Beckert. *Integration und Uniformierung von Methoden des tableaubasierten Theorembeweisens*. PhD thesis, Universität Karlsruhe, Fakultät für Informatik, July 1998.

[3] 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, Dordrecht, 1998.

[4] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, New York, second edition, 1996.

[5] R. Hähnle and S. Klingenbeck. A-ordered tableaux. *J. of Logic and Computation*, 6(6):819–834, 1996.

[6] R. Hähnle and C. Pape. Ordered tableaux: Extensions and applications. In *Proceedings, International Conference on Theorem Proving with Analytic Tableaux and Related Methods, Pont-á-Mousson, France*, LNCS 1227, pages 173–187. Springer, 1997.