Chapter 1

Tableaux and Connections

$\boxed{\text{Second Reader's Complete Draft}}$

Reiner Hähnle

Department of Computer Science
University of Karlsruhe
`reiner@ira.uka.de`
`http://i12www.ira.uka.de/~reiner`

Contents

HANDBOOK OF AUTOMATED REASONING
Edited by Alan Robinson and Andrei Voronkov

1. Introduction

Reasoning methods based on tableaux and their relatives gained a lot of attention in the past decade after a long period of near stagnation. One reason is that theoretical and implementational progress finally permitted to build tableau-based theorem provers [Moser, Ibens, Letz, Steinbach, Goller, Schumann and Mayr 1997] that can compete [Sutcliffe and Suttner 1997] with state-of-the-art resolution-based systems. Another reason is the increased need for deduction in various non-classical logics for which tableau calculi are particularly well suited.

Today, a large number of refinements of tableau-like calculi aimed at efficient automated proof search are available. In fact, there are so many of them that it has become quite difficult for the non-specialist keep track of the main developments. The difficulty of this task is incrased by the plethora of names for closely related systems: connection tableaux, connection method, hypertableaux, matrices, matings, model elimination, model generation, near-Horn logic programming, SL-resolution all are relatives of each other.

In this paper I introduce the main lines of development of tableau-like calculi, as far as they are relevant for automated reasoning, in a uniform framework. At the same time I work out their mutual relationships and I classify the refinements according to various properties.

Most refinements of tableau calculi are defined and implemented only for clause normal form. Accordingly, after a brief treatment of tableaux for full first-order logic in Section 3, the bulk of the material is presented on the clause level (the transformation of arbitrary formulas into clause normal form is discussed in detail in Chapters [Chapter `normal`] and [Chapter `small`] of this Handbook). In Section 4 the main types of refinements of tableau-like calculi are defined and discussed; in Section 5 a number of related calculi are defined relative to the coordinates introduced in the section before. The history of tableaux-like proof methods is long and vined. Many key ideas were discovered several times independently. I sketch the major developments in the brief historical Section 6.

2. Preliminaries

Here some basic ingredients of computational logic are collected. This section cannot replace a proper introduction into logic and elementary issues of theorem proving. I recommend Fitting's [1996] book as background.

2.1. Syntax

A *first-order signature* $\Sigma = \langle P_\Sigma, F_\Sigma \rangle$ consists of a non-empty set P_Σ of predicate symbols and a set F_Σ of function symbols. For skolemization we do not use symbols from F_Σ but from a special infinite set F_{sko} of *Skolem function symbols* that is disjoint from F_Σ; the extended signature $\langle P_\Sigma, F_\Sigma \cup F_{sko} \rangle$ is denoted with Σ^*. The symbols in P_Σ, F_Σ and F_{sko} may be used with any non-negative arity. In addition, there is an infinite set Var of *object variables*.

Given a signature Σ, the sets $\mathcal{T}_\Sigma$ of *terms* and $\mathcal{A}_\Sigma$ of *atoms* over Σ are inductively defined by:

1. Object variables and 0-ary function symbols from Σ are terms.
2. If $t_1, \ldots, t_n$ are terms, f is a n-ary function symbol from Σ, and P is a n-ary predicate symbol from Σ, then $f(t_1, \ldots, t_n)$ is a term and $P(t_1, \ldots, t_n)$ is an atom over Σ.

The *logical operators* are the connectives $\vee$ (disjunction), $\wedge$ (conjunction) and $\neg$ (negation), the quantifier symbols $\forall$ and $\exists$, and the constant operators *true* and *false*.

Given a signature Σ, the set $\mathcal{L}_\Sigma$ of *formulas*[1] over Σ is inductively defined by:

1. *true*, *false* and atoms over Σ are formulas.
2. If ϕ is a formula, then $\neg\phi$ is a formula.
3. If $\phi_1, \ldots, \phi_n$, $n > 1$, are formulas none of which is a conjunction (disjunction), then $\phi_1 \wedge \ldots \wedge \phi_n$ (resp. $\phi_1 \vee \ldots \vee \phi_n$) is a formula.
4. If ϕ is a formula and $x \in \mathrm{Var}$, then $(\forall x)\phi$ and $(\exists x)\phi$ are formulas. ϕ is called the *scope* of the quantifier.

Formulas that are identical up to associativity of $\vee$ and $\wedge$ are identified. A *literal* is an atom or a negated atom. In the former case, one speaks of a *positive literal*, otherwise of a *negative literal*.

A formula is in *negation normal form* (NNF) if each occurrence of the negation symbol in it is part of a literal.

A *ground term (atom, literal, formula)* is a term (atom, literal, formula) that contains no object variables. The set of ground terms is abbreviated with $\mathcal{T}^0$. No difference is made between ground first-order formulas and propositional formulas.

The *complement* $\overline{\phi}$ of a formula ϕ is defined by: $\overline{\phi} = \psi$ if ϕ is of the form $\neg\psi$, and $\overline{\phi} = \neg\phi$ otherwise.

An occurrence of an object variable x in a formula is called *bound* if x occurs in the scope of a quantifier over x, it is called *free* otherwise. A formula without free variable occurrences is a *sentence*.

A *clause* is a sentence of the form $(\forall x_1)\cdots(\forall x_n)L_1 \vee \cdots \vee L_m$, where L_i are literals. For sake of readability the quantifier prefix of first-order clauses is usually not written (but assumed to be present). Note that clauses are particular formulas.

A clause with at most one occurrence of a positive literal is a *Horn clause*. A clause with only positive (negative) literals is a *positive (negative) clause*. A non-empty, positive Horn clause is called a definedfact (note that it must contain exactly one literal), a non-empty, negative Horn clause is called a *query*. Non-empty Horn clauses that are neither facts nor queries are called *rule*.

When C, D are ground clauses $C \subseteq D$ means that every literal of C occurs also in D; $L \in D$ expresses that the literal L occurs in clause D. A clause is a *tautology* if it contains p and $\neg p$ for some atom p.

[1] Implication and equivalence are considered to be defined operators, i.e., $\phi \to \psi$ is the same as $\neg\phi \vee \psi$, and $\phi \leftrightarrow \psi$ is the same as $(\phi \wedge \psi) \vee (\neg\phi \vee \neg\psi)$.

A *substitution* is a mapping $\sigma : \mathrm{Var} \to \mathcal{T}_\Sigma$. It is extended to terms and (sets of) formulas as usual:

- $\sigma(c) = c$, $\sigma(\mathit{true}) = \mathit{true}$, $\sigma(\mathit{false}) = \mathit{false}$
- $\sigma(s(t_1, \ldots, t_n)) = s(\sigma(t_1), \ldots, \sigma(t_n))$ for $s \in F_\Sigma \cup P_\Sigma$
- $\sigma(\phi \bullet \psi) = \sigma(\phi) \bullet \sigma(\psi)$ for $\bullet \in \{\neg, \wedge, \vee\}$
- $\sigma((Qx)\phi) = (Qx)\sigma'(\phi)$ for $Q \in \{\forall, \exists\}$, where $\sigma' = \sigma \backslash \{x/t \mid t \in \mathcal{T}\}$
- $\sigma(\{\phi_1, \ldots, \phi_n\}) = \{\sigma(\phi_1), \ldots, \sigma(\phi_n)\}$

If $\sigma(x) = x$ for all but finitely many $x \in \mathrm{Var}$ it is denoted $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$. Application of substitutions is usually written postfix (note that $\phi(\sigma \circ \rho) = (\phi\rho)\sigma = \phi\rho\sigma$). Without loss of generality we assume that the range of substitutions σ never contains object variables that occur bound in a formula ϕ to which σ is applied. When the t_i are ground terms one has a *ground substitution*. A *renaming* is a substitution ν which is a permutation such that the variables in its range are new in the context, where ν appears.

If T is a non-empty set of terms and $|T\sigma| = 1$, then σ is a *unifier* of T. It is a *most general unifier (MGU)* if for all unifiers ρ of T there is a substitution θ such that $\rho = \theta \circ \sigma$.

2.1. Proposition. *Unifiability of a finite set of terms can be decided and, when positive, an idempotent MGU computed in linear time.*

An *instance* of a clause $C = (\forall x_1) \cdots (\forall x_n) L_1 \vee \cdots \vee L_m$ is an expression $(L_1 \vee \cdots \vee L_m)\sigma$, where σ is a renaming. When σ is a ground substitution one has a *ground instance* of C.

A substring ψ of a formula ϕ which is itself a formula is called a *subformula* of ϕ. If ψ is a subformula of ϕ and $\phi \neq \psi$ then ψ is a *proper subformula* of ϕ. If ψ is a proper subformula of ϕ such that there is no proper subformula ρ of ϕ with ϕ being a proper subformula of ρ, then ψ is an *immediate subformula* of ϕ.

As we deal with arbitrary formulas we have to account for the fact that a disjunctive subformula may occur negated and thus is implicitly a conjunctive formula etc. Also, the sign of a literal may be implicitly complemented.

2.2. Definition. An occurrence of a subformula ρ of $\phi \in \mathcal{L}_\Sigma$ is

1. *positive* if $\phi = \rho$,
2. *negative* (*positive*) if ϕ is of the form $\neg\psi$ and the occurrence of ρ is positive (negative) in ψ,
3. *positive* (*negative*) if ψ is an immediate subformula of ϕ, but $\phi \neq \neg\psi$, and the occurrence of ρ is positive (negative) in ψ.

2.2. Semantics

Given a first-order signature Σ a *first-order structure* $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ consists of a non-empty set $\mathbf{D}$ called *domain* and an *interpretation* $\mathbf{I}$ that assigns to each $f \in F_\Sigma$ a mapping $\mathbf{I}(f) : \mathbf{D}^{\alpha(f)} \to \mathbf{D}$ and to each $P \in P_\Sigma$ a relation $\mathbf{I}(P) \subseteq 2^{\mathbf{D}^{\alpha(P)}}$.

A *variable assignment* is a mapping $\mu : \mathrm{Var} \to \mathbf{D}$. The *$d$-variant of μ at x* is

$$\mu_x^d(y) = \begin{cases} d & \text{if } x = y \\ \mu(y) & \text{otherwise} \end{cases}$$

In any first-order structure $\mathbf{M}$ a variable assignment μ is extended to terms:

$x^{\mathbf{M},\mu} = \mu(x)$ for $x \in \mathrm{Var}$

$f(t_1, \ldots, t_n)^{\mathbf{M},\mu} = I(f)(t_1^{\mathbf{M},\mu}, \ldots, t_n^{\mathbf{M},\mu})$ for $f(t_1, \ldots, t_n) \in \mathcal{T}$

Satisfiability of formulas ϕ in $\mathbf{M}$ and μ, written $(\mathbf{M}, \mu) \models \phi$ is defined as follows:

$(\mathbf{M}, \mu) \models \mathit{true}$	for all $\mathbf{M}$ and μ
$(\mathbf{M}, \mu) \models \mathit{false}$	for no $\mathbf{M}$ and μ
$(\mathbf{M}, \mu) \models P(t_1, \ldots, t_n)$	iff $(t_1^{\mathbf{M},\mu}, \ldots, t_n^{\mathbf{M},\mu}) \in \mathbf{I}(p)$ for $P(t_1, \ldots, t_n) \in \mathcal{A}$
$(\mathbf{M}, \mu) \models \neg\phi$	iff not $(\mathbf{M}, \mu) \models \phi$
$(\mathbf{M}, \mu) \models \phi_1 \wedge \cdots \wedge \phi_n$	iff $(\mathbf{M}, \mu) \models \phi_i$ for all $i \in \{1, \ldots, n\}$
$(\mathbf{M}, \mu) \models \phi_1 \vee \cdots \vee \phi_n$	iff $(\mathbf{M}, \mu) \models \phi_i$ for at least one $i \in \{1, \ldots, n\}$
$(\mathbf{M}, \mu) \models (\forall x)\phi$	iff $(\mathbf{M}, \mu_x^d) \models \phi$ for all $d \in \mathbf{D}$
$(\mathbf{M}, \mu) \models (\exists x)\phi$	iff $(\mathbf{M}, \mu_x^d) \models \phi$ for at least one $d \in \mathbf{D}$

A first-order Σ-structure $\mathbf{M}$ is a *model* of a set of formulas $M \subset \mathcal{L}_\Sigma$, denoted $\mathbf{M} \models_\Sigma M$, if $(\mathbf{M}, \mu) \models \phi$ for all $\phi \in M$ and variable assignment μ. ϕ is a *logical consequence* of M, denoted $M \models_\Sigma \phi$ if each model of M is also a model of ϕ. ϕ is *valid*, written $\models_\Sigma \phi$, when each Σ-structure $\mathbf{M}$ is a model of ϕ.

A first-order Σ-structure $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ is a *term structure* if $\mathbf{D} = \mathcal{T}^0$.

2.3. PROPOSITION. *For all $\phi \in \mathcal{L}_\Sigma$ and sets of $\mathcal{L}_\Sigma$-sentences M: $M \models_\Sigma \phi$ iff $M \cup \{\neg\phi\}$ is unsatisfiable.*

2.4. THEOREM. *If a sentence $\phi \in \mathcal{L}_\Sigma$ is satisfiable, then it has a $\Sigma*$-term model.*

Assume F_Σ contains at least one constant, then $\mathcal{T}_\Sigma^0 \neq \emptyset$. A structure $\langle \mathcal{T}_\Sigma^0, \mathbf{I} \rangle$, where $\mathbf{I}(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for all $f \in F_\Sigma$, is called *Herbrand structure*.

2.5. THEOREM (Clausal Version of Herbrand's Theorem). *Let S be a finite, unsatisfiable set of clauses. Then there is a finite set $\widehat{S}$ of ground instances of S such that $\widehat{S}$ is unsatisfiable.*

A proof is, for example, in [Smullyan 1995].

3. The Tableau Method

3.1. Informal Introduction

It is common to view the tableau method as a proof by contradiction and case distinction (this view was already stressed by pioneers Beth [1955] and Hintikka [1955]). More precisely, it allows one to systematically generate subcases until elementary contradictions are reached. Let us go through a small example:

Assume we want to prove the following simple theorem from elementary set theory: for any sets P, Q, R, S, if (1) $S \cap Q = \emptyset$, (2) $P \subseteq Q \cup R$, (3) $P = \emptyset \rightarrow Q \neq \emptyset$, and (4) $Q \cup R \subseteq S$, then $P \cap R = \emptyset$.

We give a proof by contradiction, therefore, assume $P \cap R \neq \emptyset$. From (3) we know that there is an element $c \in P$ (case 1) or there is an element $d \in Q$ (case 2). We proceed with case 1. The assumption applied to c yields that $c \notin P$ (case 1.1) or $c \notin R$ (case 1.2). Case 1.1 is contradictory. In case 1.2 apply (2) to c: if $c \in P$ then $c \in Q$ or $c \in R$. But we know already that $c \in P$ and $c \notin R$ which leaves case 1.2.2, $c \in Q$, to consider. Using (1) similarly as the assumption yields case 1.2.2.1, $c \notin S$. Finally, from (4) we have: if $c \in Q \cup R$ then $c \in S$. Again, the second possibility is a contradiction, and case 1.2.2.1.1, $c \notin Q \cup R$, remains. Thus, $c \notin Q$, contradiction. The remaining case 2 is similar.

This proof is easier to follow if the case distinctions are displayed tree-like as in Figure 1. We observe that case distinctions could be generated schematically depending on the form of their premiss. At several points, premisses had to be suitably instantiated. (1), (2) and (4) are universally quantified, for instance, (1) says that *for all elements x, x cannot be both an element of S and of Q*. In automated theorem proving finding instances is done by *unification*—one tries to find a substitution that produces a contradiction in the current branch of the proof (in the example this is $\{x/c\}$). In general one needs, of course, to apply a premiss more than once during a proof. This *multiplicity* cannot be computed in advance (otherwise, first-order logic would be decidable). One of the problems that tableau methods must solve is to systematically enumerate "enough" (this is made precise later) instances of universally quantified formulas.

From premiss (3) one obtains existentially quantified expressions saying that at least one of P and Q must contain an element. In automated theorem proving such witness elements are produced by Skolemization: the existentially quantified variable is replaced by a "new" (again, this is made precise later) term.

3.2. Non-clausal Tableaux with Unification

3.2.1. Unifying Notation

The first step in formalizing the considerations in the previous section is to supply formal rules that tell in which way a formula is analyzed according to its leading connective. Smullyan [1995] observed that some work can be saved if non-literal formulas are grouped into types which are treated identically: α for formulas of conjunctive type, β for formulas of disjunctive type, γ for quantified formulas of universal, and δ for quantified formulas of existential type. Correspondence between formulas and their types is summarized in Table 1.

The letters α, β, γ, and δ are used to denote formulas of (and only of) the appropriate type. In the case of γ- and δ-formulas the variable x bound by the (topmost) quantifier is made explicit by writing $\gamma(x)$ and $\gamma_1(x)$ (resp. $\delta(x)$ and $\delta_1(x)$); accordingly $\gamma_1(t)$ denotes the result of replacing all occurrences of x in γ_1 by t. Associativity of $\wedge$ and $\vee$ justifies conjunctive and disjunctive formulas with an

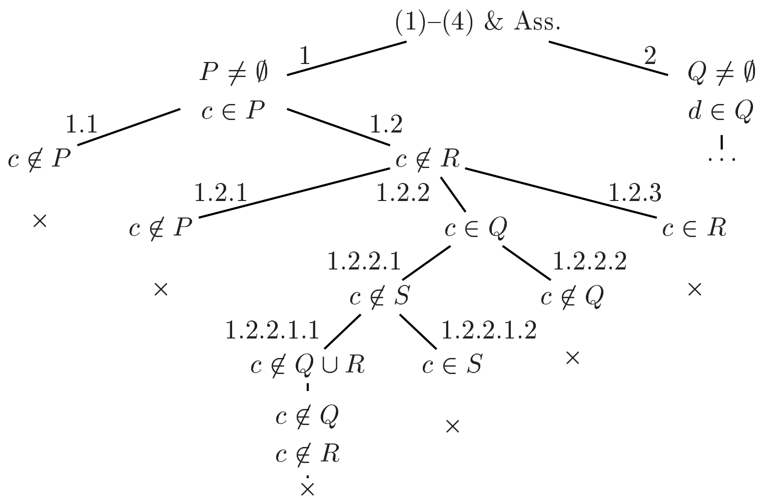


Figure 1: Structure of an informal proof by contradiction and case distinction.

α	$\alpha_1, \ldots, \alpha_n$
$\phi_1 \wedge \ldots \wedge \phi_n$	$\phi_1, \ldots, \phi_n$
$\neg(\phi_1 \vee \ldots \vee \phi_n)$	$\neg\phi_1, \ldots, \neg\phi_n$
$\neg\neg\phi$	ϕ
$\neg$*false*	*true*
$\neg$*true*	*false*

β	$\beta_1, \ldots, \beta_n$
$\phi_1 \vee \ldots \vee \phi_n$	$\phi_1, \ldots, \phi_n$
$\neg(\phi_1 \wedge \ldots \wedge \phi_n)$	$\neg\phi_1, \ldots, \neg\phi_n$

γ	γ_1
$(\forall x)(\phi(x))$	$\phi(x)$
$\neg(\exists x)(\phi(x))$	$\neg\phi(x)$

δ	δ_1
$\neg(\forall x)(\phi(x))$	$\neg\phi(x)$
$(\exists x)(\phi(x))$	$\phi(x)$

Table 1: Correspondence of formulas and their types.

indefinite number of arguments.

Some authors [Lis 1960, Smullyan 1995] prefer to work with *signed formulas*. These are expressions of the form T ϕ, F ϕ, where ϕ is a formula. Signed formula tableaux relate more directly to sequent calculi, because T-signed formulas play the rôle of formulas standing left of a sequent arrow while F-signed formulas are on the right (see also Section 4.8.1 below). In classical logic theorem proving there is no particular gain from signs, but in non-classical logics their use is indispensable [Beckert and Goré 1997, Hähnle and Escalada-Imaz 1997].

α	β	$\gamma(x)$	$\delta(x)$
α_1	$\beta_1 \mid \cdots \mid \beta_n$	$\gamma_1(y)$	$\delta_1(sko_\delta(x_1,\ldots,x_n))$
$\vdots$		$y \in \mathrm{Var}$ is new to the tableau.	$x_1,\ldots,x_n$ are the free variables in δ.
α_n			

Table 2: Rule schemata for tableaux with unification.

3.2.2. Tableau Rules

With unifying notation decomposition rules for arbitrary formulas can be given in a concise way.

In Table 2 expansion rule schemata for the various formula types are given. Premisses and conclusions are separated by a horizontal bar, while vertical bars in the conclusion denote different *extensions*. The formulas in an extension are implicitly conjunctively connected, and different extensions are implicitly disjunctively connected. We use n-ary α- and β-rules, i.e., when the β-rule is applied to a formula $\psi = \phi_1 \vee \ldots \vee \phi_n$, then ψ is broken up into n subformulas (instead of splitting it into two formulas $\phi_1 \vee \ldots \vee \phi_r$ and $\phi_{r+1} \vee \ldots \vee \phi_n$, $0 < r < n$).

Type γ formulas are simply stripped from their quantifier while the quantifed variable is renamed into a object variable not ocurring elsewhere. Instantiation of free variables is delayed.

The δ-rule is the most technical rule. Its purpose is to replace an existential quantifier with a witness element or Skolem term. It incorporates two important optimizations: the first is that the choice of the witness element merely depends on the free variables in δ, not on all free variables on the current branch; secondly, the leading function symbol f of the Skolem term may be the same for δ-formulas which are identical up to variable renaming, formally:

3.1. Definition. Given a signature $\Sigma = \langle P_\Sigma, F_\Sigma \rangle$, the function *sko* assigns to each $\delta \in \mathcal{L}_{\Sigma^*}$ a symbol $sko_\delta \in F_{sko}$ such that (a) $sko_\delta > f$ for all $f \in F_{sko}$ occurring in δ, where $>$ is an arbitrary but fixed ordering on F_{sko}, and (b) for all $\delta, \delta' \in \mathcal{L}_\Sigma$ the symbols sko_δ and sko'_δ are identical if and only if δ and δ' are identical up to variable renaming (including renaming of the bound variables).

The purpose of condition (a) in the above definition of *sko* is to avoid cycles like: sko_δ occurs in δ' and sko'_δ occurs in δ.

Both improvements of the δ-rule together have the consequence that its conclusion can be computed locally to the formula δ—no "global" information is required.

3.2.3. Tableau Proofs

As was hinted at already, tableau proofs are trees whose nodes are formulas that are (sub)goals in the proof and the tree structure gives the logical dependence between them. Assume we want to prove that a set of sentences Φ logically implies a sentence

ψ. By Proposition 2.3 this amounts to checking that the set of sentences $\Phi \cup \{\neg\psi\}$ is unsatisfable.

3.2. DEFINITION. Let Σ be a first-order signature. A *tableau* (over Σ) is a finitely branching tree whose nodes are formulas from $\mathcal{L}_{\Sigma^*}$. A *branch* in a tableau T is a maximal path in T.[2] Given a set Φ of sentences from $\mathcal{L}_\Sigma$, a *tableau for* Φ is constructed by a (possibly infinite) sequence of applications of the following rules:

1. The tree consisting of a single node *true* is a tableau for Φ (initialization rule).
2. Let T be a tableau for Φ, B a branch of T, and ψ a formula in $B \cup \Phi$. If the tree T' is constructed by extending B by as many new linear subtrees as an instance of a tableau rule schema in Table 2 with premiss ψ has extensions, and the nodes of the new subtrees are the formulas in the extensions of the rule instance, then T' is a tableau for Φ (expansion rule).
3. Let T be a tableau for Φ, B a branch of T, and ψ and ψ' literals in $B \cup \Phi$. If ψ and $\overline{\psi'}$ are unifiable with MGU σ, and T' is constructed by applying σ to all formulas in T (i.e., $T' = T\sigma$), then T' is a tableau for Φ (closure rule).

The last item of the previous definition incorporates two simplifications: first, MGUs are used instead of arbitrary substitutions; second, ψ and ψ' are literals, not arbitrary formulas. The former is crucial, because there are only finitely many MGUs of (complemented) formulas on a finite tableau. If Φ is finite this implies that there are systematic procedures for enumerating the (finite) tableaux for Φ.

The branches in a tableau correspond to different subcases in a proof. Obviously, Def. 3.2(3) is a means to produce a contradiction in a subcase/branch. A tableau proof is finished when this has been achieved for all branches. The following terminology is standard:

3.3. DEFINITION. In a tableau T for a set Φ of sentences a branch B is *closed* iff $B \cup \Phi$ contains a pair $\phi, \neg\phi \in \mathcal{L}_{\Sigma^*}$ of complementary formulas, or *false*; otherwise, it is *open*. A tableau is closed if *all* its branches are closed.

A *tableau proof* for (the unsatisfiability of) a set $\Phi \subset \mathcal{L}_\Sigma$ of sentences is a closed tableau T for Φ.

3.4. EXAMPLE. We are now in a position to formalize the introductory example from Section 3.1. Sets P, Q, R, S are represented by unary predicates P, Q, R, S: their characteristic functions. Then, over the signature $\Sigma = \langle \{P, Q, R, S\}, \{\}\rangle$, the claim holds if and only if the set Φ consisting of $\mathcal{L}_\Sigma$-sentences (1) $\neg(\exists x)(S(x) \wedge Q(x))$, (2) $(\forall x)(\neg P(x) \vee Q(x) \vee R(x))$, (3) $(\exists x)(P(x)) \vee (\exists x)(Q(x))$, (4) $(\forall x)(\neg(Q(x) \vee R(x)) \vee S(x))$, (5) $\neg(\exists x)(P(x) \wedge R(x))$ is unsatisfiable.

Figure 2 shows a tableau T for Φ. The nodes of the tableau are numbered starting from 6 (the numbers 1–5 refer to the formulas in Φ); an expression $[\mathbf{i}; j]$ is in front

[2] When no confusion can arise, branches are frequently identified with the set of their nodes (formulas).

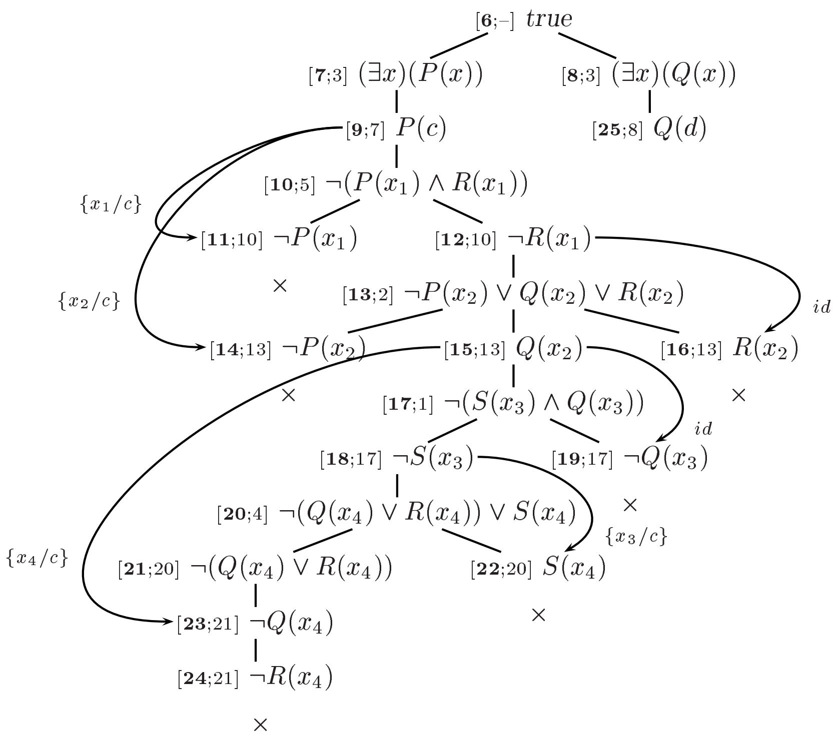


Figure 2: Partial tableau proof for Φ from Example 3.4.

of the i-th node N_i, where j signifies that N_i stems from an expansion rule applied to N_j (respectively, to formula (j) in Φ).

All branches of T except the rightmost one can be closed by virtue of the closure rule; a closure is indicated by an arc between its complementary literals, labelled with the required MGU. T is not yet a tableau proof for Φ, but it can be extended to one by adding a copy of the subtableau with root node 17 below node 25 and instantiating the free variables in that copy with d (instead of c).

3.2.4. Tableau Semantics and Soundness

Since our goal is to use tableaux as a framework for formal proofs, we require to extend semantics from formulas to tableaux. Our guideline here is to ensure that a tableau for Φ is closed iff Φ is unsatisfiable. We fixed already that a tableau is thought of as the disjunction of its branches which in turn are considered as conjunctions of their labels. By a standard argument then, the equivalence above is reduced to the question whether the tableau construction rules leave tableau satisfiability unaltered. This is a routine matter for all but the δ-rule which requires some care. Recall that two optimizations were incorporated into this rule (Section 3.2.2):

(i) the variables of the Skolem term are restricted to the free variables of δ, (ii) the leading function symbol sko_δ of the Skolem term is not unique in a tableau proof. Tableau semantics must be carefully chosen to reflect these restrictions. To meet (i) it suffices to treat free variables in a tableau essentially as if they were universally quantified.

3.5. DEFINITION. A tableau T for $\Phi \subset \mathcal{L}_\Sigma$ is *satisfiable* if there is a structure $\mathbf{M}$ of Φ such that for every variable assignment μ there is a branch B of T with $(\mathbf{M}, \mu) \models B$. In that case we say that $\mathbf{M}$ is a model of T, denoted by $\mathbf{M} \models T$.

For (ii) it is important that Skolem function symbols are interpreted in the "right" way. The most elegant way to achieve this, is to define formula semantics with respect to only such interpretations—let us call them *canonical interpretations*. Of course, one needs to show then that each satisfiable formula can be satisfied by a canonical interpretation.

3.6. DEFINITION. A term structure $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ is *canonical* iff for all variable assignments μ and $\delta(x) \in \mathcal{L}_{\Sigma^*}$: if $(\mathbf{M}, \mu) \models \delta(x)$ then $(\mathbf{M}, \mu) \models \delta_1(sko_\delta(x_1, \ldots, x_n))$, where $x_1, \ldots, x_n$ are the free variables in δ.

3.7. LEMMA ([Beckert and Hähnle 1998]). *Given a signature Σ, if the set $\Phi \subset \mathcal{L}_\Sigma$ of sentences is satisfiable, then there is a canonical structure $\mathbf{M}^*$ over Σ^* such that $\mathbf{M}^* \models \Phi$.*

3.8. COROLLARY. *Let $\mathbf{M}^*$ be a canonical structure over Σ^*, μ a variable assignment, and $\phi \in \mathcal{L}_{\Sigma^*}$; and let ϕ' be constructed from ϕ by (a) replacing a positive occurrence of some $\delta(x)$ in ϕ by $\delta_1(\mathrm{sko}_{\delta(x)}(x_1, \ldots, x_n))$, or by (b) replacing a negative occurrence of $\overline{\delta(x)}$ in ϕ by $\overline{\delta_1(\mathrm{sko}_{\delta(x)}(x_1, \ldots, x_n)}$, where $x_1, \ldots, x_n$ are the free variables in $\delta(x)$. Then $(\mathbf{M}^*, \mu) \models \phi$ implies $(\mathbf{M}^*, \mu) \models \phi'$.*

3.9. LEMMA. *Any tableau for a satisfiable set of $\mathcal{L}_\Sigma$-sentences is satisfiable.*

PROOF. By definition of tableaux there is a sequence $T_1, \ldots, T_m$ $(m > 0)$, where $T = T_m$ and T_1 is the initial tableau whose single node is *true*, and where T_{i+1} is constructed from T_i by applying a single tableau expansion or closure rule. By Lemma 3.7, the input set is satisfied by a canonical structure $\mathbf{M}^*$ over Σ^*. By induction on m one proves that $\mathbf{M}^*$ satisfies all of $T_1, \ldots, T_m$ and hence T. The induction step is trivial and all cases, but the δ-rule case are straightforward (see [Fitting 1996] for a detailed proof). The latter, however, follows from the corollary. □

Now assume we have a closed tableau T for a set of $\mathcal{L}_\Sigma$-sentences Φ. Obviously, no structure and variable assignment can satisfy a closed branch, so T is unsatisfiable. By the preceding lemma, Φ is unsatisfiable as well. This proves:

3.10. THEOREM (Soundness). *If there is a tableau proof for a set $\Phi \subset \mathcal{L}_\Sigma$ of sentences, then Φ is unsatisfiable.*

3.2.5. Universal and Rigid Variables

In general, different instances of the variables in the scope a universal quantifier are needed in order to close a branch (or a subtableau). In free variable tableaux the mechanism to achieve this is to apply the γ-rule multiply to generate formula instances with different free variables. It is crucial to note that free variables in tableaux are *not* implicitly universally quantified locally to the branch on which they occur[3], but are *rigid*: any substitution σ with $\sigma(x) \neq x$ must be applied to *all* occurrences of x in a tableau. Figure 3 shows an unsound tableau proof for the invalid formula $\psi = \neg((\forall x)(P(x) \vee Q(x)) \to ((\forall x)P(x) \vee (\forall x)Q(x)))$ that would be possible if free variables were not handled rigidly.

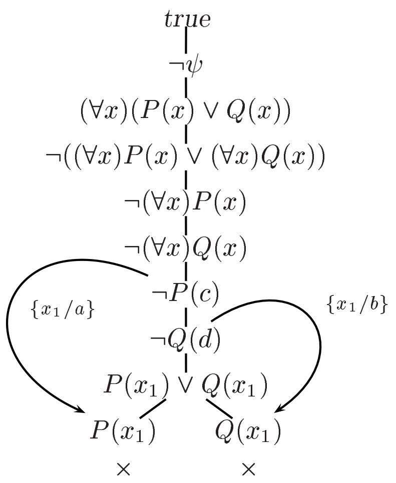


Figure 3: Unsound tableau proof due to non-rigid treatment of free variables.

In some cases, though, it is sound to treat free variables as if they were quantified universally. For example, if we have a tableau for $\Phi = \{\neg P(a) \vee \neg P(b), (\forall x)P(x)\}$ that consists of two branches, one containing $P(x_1)$ and $\neg P(a)$, and the other containing $P(x_1)$ and $\neg P(b)$. This tableau cannot be closed immediately as no single substitution closes both branches. To find a proof, the γ-rule has to be applied again to create another instance of $(\forall x)P(x)$. In this example, as a logical consequence of Φ and the formulas already on the tableau (in a sense made precise in Def. 3.11), $(\forall x)\phi(x)$ can be added to *each branch*. In this situation, substitutions with differing values for x can be used without destroying soundness of the calculus. The tableau for Φ then would close earlier. Recognizing such situations and exploiting them allows to use more general closing substitutions, yields shorter tableau proofs, and in many cases reduces the search space.

[3] In contrast to this, resolvent clauses in a resolution calculus, for example, *are* universally quantified

3.11. DEFINITION. Suppose ϕ is a formula on a branch B of a tableau T for $\Phi \subset \mathcal{L}_\Sigma$. Let T' result from adding $(\forall x)\phi$ to B for some $x \in \text{Var}$. ϕ is *universal* on B with respect to x if $T \models T'$.[4] $\text{UVar}(\phi)$ are the universal variables of ϕ.

Instead of designing a closure rule that takes universal variables into account (Definition 3.2.(3)), we generalize the concept of a unifier:

3.12. DEFINITION. A substitution σ is a *unifier* of formulae ϕ, ϕ' on a branch of a tableau T if it is the restriction of a substitution τ with the property $(\phi\pi)\tau = (\phi'\pi')\tau$ to $\text{Var} \setminus U$, where $U = \text{UVar}(\phi) \cap \text{UVar}(\phi')$ and π, π' are renamings of the variables in U with variables new to T.

With the closure rule based on this modified concept of unification, a tableau proof with less applications of expansion rules than in the standard free variable tableau calculus may be found; the calculus is strengthened.

Recognizing universal formulas is undecidable in general, however, a practically important subclass can be recognized easily (and this can already shorten tableau proofs exponentially): in any sequence of tableau rule applications with a variable x introduced by a γ-rule application and not distributed over different branches by β-rule applications, all formulas generated during this sequence of rule applications are universal w.r.t. x, formally:

3.13. LEMMA. *A formula ϕ on a branch B of a tableau T is universal w.r.t. x on B if in the construction of T the formula ϕ was added to B by applying (1) a γ-rule and x is the free variable introduced; (2) an α-, γ-, or δ-rule to a formula that is universal on B w.r.t. x; or (3) a β-rule to a formula β that is universal on B w.r.t. x, and x does not occur in any $\beta_i \neq \phi$.*

The proof of soundness of free variable tableaux (Theorem 3.10) can accomodate the universal formula technique. Bibel [1982*a*] proposed a technique for reducing the size of proofs in the connection method, called *splitting by need*; like universal formulas it is based on the idea to avoid copying a universally quantified formula in cases where it is sound to use a single copy with different variable instantiations.

3.2.6. Binary versus n-ary Rules

The n-ary branching tableau rules for type β formulas in Table 2 have a binary variant

$$\frac{\beta}{\beta_i \;\Big|\; \beta_1 \vee \cdots \vee \beta_{i-1} \vee \beta_{i+1} \vee \cdots \vee \beta_n} \tag{3.1}$$

which in fact is the more popular one and used, for example, in [Smullyan 1995, Fitting 1996]. Only recently, Massacci [1998*a*] pointed out that tableaux based on

[4] When obvious, a formula ϕ being universal on a branch B w.r.t. a variable x is just referred to as "the universal formula ϕ," and x as "the universal variable x."

the n-ary rule cannot polynomially simulate tableaux based on (3.1) with respect to minimal proof size. In the present paper I work with the n-ary rule to achieve maximum uniformity among clausal and non-clausal tableaux. As (3.1) can easily simulate the n-ary rule, I trust that generality is not lost.

3.3. From Calculus to Proof Procedure

Tableau soundness gives the desirable property of tableaux with unification that a closed tableau for Φ signifies validity of $\widehat{\Phi}$. Remains the question whether for *all* valid sentences a tableau proof exists and, if this is the case, how it can be found. While the first question is answered affirmatively in Section 3.4, for the second one, whether a concrete *tableau proof search procedure* will find a closed tableau provided there exists one, a fully satisfactory answer is not yet available. Let me point out why this so-called *strong completeness problem* is difficult.

It is straightforward to derive a tableau proof procedure from Definition 3.2, but note that such a procedure is indeterministic: usually, a great number of rules is applicable to any given tableau. More precisely, one first must select a branch B, where a rule is applied, then decide whether an expansion rule or a closure rule is used; in the first case one must choose a formula $\psi \in B \cup \Phi$, in the second case a pair of literals on B. Let us refer to these kinds of indeterminism with the phrases *select branch*, *select formula*, *select pair*, and *select mode* in the following. In the ground case no substitutions occur and it obviously suffices to select each non-literal formula exactly once on each branch. Making these choices deterministic in an arbitrary way in the first-order case, however, almost certainly results in an incomplete proof procedure.

In Figure 4, for example, the γ-formula is always preferred for expansion rule application, delaying expansion of the inconsistent second formula indefinitely. In an obvious way, the formula $Q \wedge \neg Q$ is treated *unfair*. This motivates the following definition.

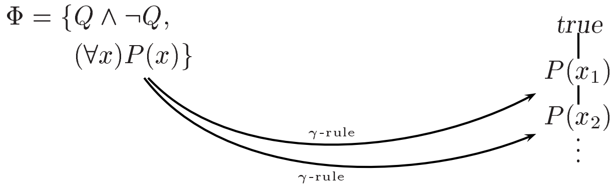


Figure 4: Incompleteness caused by unfair *select formula*.

3.14. Definition. The construction of a sequence $(T_n)_{n>0}$ of tableaux for $\Phi \subset \mathcal{L}_\Sigma$, where T_n is obtained from T_{n-1} by applying a tableau expansion rule, is *fair* if the following holds for all branches B in the infinite tableau that is approximated by that sequence: (1) all α, β, and δ occurring on B or in Φ were used to expand B

(by applying the appropriate expansion rule); (2) all γ occurring on B or in Φ were used infinitely often to expand B (by applying the γ-rule).

One may construct a fair sequence of tableaux for any set of sentences. Combining fair application of the expansion rule with fair application of the closure rule, however, is a difficult problem, because tableaux with unification are *destructive*:

3.15. DEFINITION. A tableau calculus is *non-destructive* if all tableaux that can be constructed from a given tableau T contain T as an initial subtree; otherwise the calculus is *destructive*.

For example, at first sight it might seem to be a good idea to apply the closure rule in a "greedy" manner that is as early as possible. Not so. One problem is that several pairs of closure literals (with incompatible MGUs) may compete, but this is not all. In Figure 5, independently from which branch is closed first, the variable x_1 gets "used up" by a substitution that blocks closure of the other branch. Of course, a second free variable instance of the γ-formula may be created, but then the same happens one level below etc.

$$\psi = ((P(b) \wedge P(c)) \rightarrow P(x)) \rightarrow \neg(Q(x) \rightarrow (Q(b) \vee Q(c)))$$
$$\Phi = \{(\forall x)\psi,\ P(a),\ \neg Q(d)\}$$

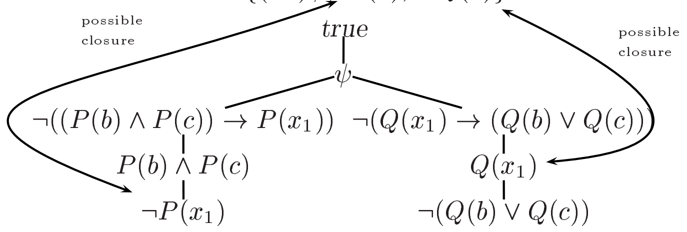


Figure 5: Incompleteness caused by unfair *select mode*.

As a slight extension of the ground case, in a non-destructive tableau calculus one can simply arrange input clauses in a queue on each branch and thus ensure that enough instances to obtain a ground proof are present. (Examples of non-destructive tableau calculi are Smullyan's [1995] ground tableaux and Fitting's [1996] delayed instantiation rule, see below). The problem is that free variable tableaux *are* destructive. The culprit is the closure rule, Definition 4.1.(iii).

Independently of destructiveness, a complete tableau procedure may fail to be proof confluent:

3.16. DEFINITION. A tableau proof procedure is *proof confluent* if from every tableau for an unsatisfiable set of sentences a closed tableau can be constructed.

In other words, the search space of a proof confluent tableau procedure contains no "dead ends" in which no proof can be found.

A destructive tableau proof procedure still might be strongly complete, but as witnessed by the examplea above, it might as well be not. At the present time, no practical, strongly complete, destructive tableau proof procedure is known (there is hope, however, see Section 4.7). Therefore, it is worth discussing possible ways around the problem. Another reason is that some of the techniques dealing with destructiveness also deal with lack of proof confluence: some of the more important complete refinements of the tableau calculus are not proof confluent. This is discussed in Section 4.3 below.

An obvious way to enforce strong completeness of tableaux with unification consists in delaying any application of the closure until all tableau branches can be closed simultaneously by a suitable substitution. This is the path chosen in Fitting's [1996] text book— and it has its inefficencies: first, one cannot discard closed branches until the proof is essentially finished which might lead to storage problems (this can be partially remedied, see Section 3.5), and second, after each expansion rule the whole tableau must be tested for closure which is very redundant.

Another option for obtaining a strongly complete tableau proof procedure comes from the observation that its indeterminism is locally finite—from each tableau only a finite number of successors can be constructed. Thus one can turn each tableau into a finitely branching OR node of an AND-OR search tree which may then be searched in a breadth first manner. This approach is impractical because of space requirements.

Stickel [1992] suggested to replace breadth first search by *d*epth *f*irst search with backtracking and *i*terative *d*eepening of the search depth (DFID search) which has only a small overhead in run time as compared to breadth first search, but is a lot more space efficient [Korf 1985].

DFID search is based on a mapping m from $\mathbb{N}$ to sets of tableaux constructed with fair *select formula* rule such that $\bigcup_{i\in\mathbb{N}} m(i)$ contains all these tableaux. Common choices for m, which is called *completion mode*, include the following:

- $m(i)$ = all tableaux with depth i
- ... with i nodes
- ... with i applications of γ-rule (per branch)
- ... with nesting depth i of terms

The tableaux in $m(1)$, $m(2)$, ... are enumerated by depth first search with backtracking over all possible choices in *select mode* and *select pair*.

Tableau proof search procedures based on DFID turned out to be elegantly and efficiently implementable in logic programming languages [Stickel 1992, Baumgartner and Furbach 1994, Beckert and Posegga 1995].

To increase efficiency of search it is important to get rid of as many OR nodes as possible. In the DFID setup this means to minimize the amount of backtracking. It is obvious that one may choose any deterministic strategy for *select branch* as all branches need to be closed eventually. In addition, it is not difficult to implement a fair strategy for *select formula* [Fitting 1996]. This leaves the—destructive—branch closure. Ideally, one would like to have a fair selection rule not only for formula selection, but for closure selection as well, thus yielding a proof confluent proof search procedure for tableaux with unification.

3.4. Tableau Completeness

The preceding discussion shows that it is difficult to ensure completeness of tableaux with unification for a deterministic proof search procedure. On the other hand, it is not too difficult to show mere existence of a closed tableau for each unsatisfiable set of sentences. The presentation of the latter result closely follows [Beckert and Hähnle 1998].

It is convenient to work with a data structure that slightly abstracts from tableau branches: so-called Hintikka sets (named after their inventor [Hintikka 1955]) may contain an infinite number of formulas whose order is irrelevant. A model can be immediately constructed for any Hintikka set.

3.17. DEFINITION. A set $H \subset \mathcal{L}_{\Sigma^*}$ of sentences is a *Hintikka set* if it satisfies the following conditions: (1) *false* $\notin H$ and there are no complementary literals in H. (2) If $\alpha \in H$, then all α_i are in H. (3) If $\beta \in H$, then some β_i is in H. (4) If $\gamma(x) \in H$, then $\gamma_1(t) \in H$ for all ground Σ^*-terms t. (5) If $\delta(x) \in H$, then $\delta_1(t) \in H$ for some ground Σ^*-term t.

3.18. LEMMA (Hintikka). *Every Hintikka set is satisfiable.*

PROOF. A term model $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ of H is defined by: $t^{\mathbf{I}} = t$ for all $t \in \mathbf{D}$, and $P^{\mathbf{I}}(t_1, \ldots, t_k) = \mathit{true}$ iff $P(t_1, \ldots, t_k) \in H$ for ground atoms $P(t_1, \ldots, t_k)$ over Σ^*. By induction on the structure of formulas in H it is easy to prove that $\mathbf{M} \models H$. □

3.19. THEOREM (Completeness). *If the set $\Phi \subset \mathcal{L}_\Sigma$ of sentences is unsatisfiable, then there is a tableau proof for Φ.*

PROOF. Let $(T_n)_{n>0}$ be a fair sequence of tableaux for Φ starting with the tableau consisting of the single node *true*; this sequence approximates the infinite tree T_∞. We define a particular ground substitution σ_∞ as follows: let $(B_k)_{k>0}$ be an enumeration of the branches of T_∞ and $(\phi_i)_{i>0}$ an enumeration of the γ-formulas in T_∞. For every γ-formula ϕ_i, if ϕ_i occurs on B_k let x_{ijk} name the new variable introduced by the j-th application of the γ-rule to ϕ_i on B_k. (Note that different x_{ijk} can name the same variable.) Finally, let $(t_j)_{j>0}$ be an enumeration of all Σ^*-ground terms.

To ensure that B_k defines a model, the instances of ϕ_i on $B_k\sigma_\infty$ must "cover" all ground terms t_j. It suffices to choose $\sigma_\infty(x_{ijk}) = t_j$ for all $i, j, k > 0$. (If x_{ijk} and $x_{i'j'k'}$ name the same variable, then $i = i'$ and $j = j'$, so σ_∞ is well-defined.)

By construction of σ_∞ and fairness of T_∞, if B is a branch in T_∞ and $B\sigma_\infty$ is open, then $B\sigma_\infty \cup \Phi$ is a Hintikka set and so Φ is satisfiable. This contradicts the assumption, hence $T_\infty\sigma_\infty$ is closed. $T_\infty\sigma$ is a finitely branching tree and the distance of all formulas involved in closures to the root node is finite. Then, by

König's Lemma[5], there is an $n > 0$ such that the finite tableau $T_n\sigma_\infty$ is closed.

In general, σ_∞ is not a *most general* unifier of complementary literals used in closures and cannot be used in an MGU closure rule application to T_n. Therefore, it remains to show that σ_∞ can be suitably decomposed: $\sigma_\infty = \sigma \circ \sigma_r \circ \sigma_{r-1} \circ \cdots \circ \sigma_1$, where σ_i is a most general closing substitution for the instance $B_i\sigma_1\sigma_2 \ldots \sigma_{i-1}$ of the i-th branch in T_n $(0 < i < r + 1)$; σ is the part of σ_∞ not actually needed to close T_n. The σ_i are constructed inductively:

Let $\sigma'_1 = \sigma_\infty$. For $1 < i < r + 1$, let σ_i be a most general substitution such that (1) σ'_{i-1} is a specialization of σ_i (there is a substitution σ'_i such that $\sigma'_{i-1} = \sigma'_i \circ \sigma_i$) and (2) σ_i is a closing substitution for $B_i\sigma_1\sigma_2 \ldots \sigma_{i-1}$. Now σ_i is a most general *closing* substitution of $B_i\sigma_1\sigma_2 \ldots \sigma_{i-1}$. Otherwise, there is a closing substitution σ''_i being more general than σ_i. The is-more-general relation is transitive, hence σ''_i is more general than σ'_{i-1} in contradiction to σ_i being already a suitable most general substitution. Finally, let $\sigma = \sigma'_r$. □

It suffices to apply the appropriate expansion rule exactly once to each α, β, or δ-formula on each branch to obtain a Hintikka set from a fairly constructed sequence of tableaux. This has the practically relevant consequence that only to γ-formulas must a rule be applied more than once per branch.

3.5. Proof Representation

3.5.1. Trees, Matrices, Connections & Matings

Trees are a quite redundant way to represent proofs. Notably, each expansion step gives rise to new copies of some subformulas. This is unnecessary, as the result of an expansion step is uniquely determined once the position, where it is to be applied, is fixed. In the case of negation normal form (NNF) formulas, the situation is even simpler: up to variable instantiation, any formula occurring on a tableau for an NNF formula ψ is just a subformula of ψ. Consequentially, given a suitable representation for NNF formulas, a tableau branch can be viewed as a sequence of positions of certain subformulas.

Such representations were first suggested independently by Bibel and Schreiber [1975], Andrews [1976], and Davydov [1973]. Andrews and Bibel also defined procedures to check the validity of NNF formula representations in full first-order logic. Full accounts of Bibel's *matrix* or *connection method* are [Bibel 1982*b*, Bibel 1987], of Andrews' *general matings* it is [Andrews 1981].

For the present discussion it is sufficient to define a *matrix* simply as an NNF formula for which a two-dimensional notation is used: disjoint subformulas are stacked vertically onto each other, while conjoint subformulas are written in a horizontal row and enclosed between square brackets; literals are unchanged. To simplify things we start with propositional logic.

[5] "A tree that is finitely branching but infinite must have an infinite branch." A proof is, for example, in [Fitting 1996].

The NNF formula $\psi = P \wedge (\neg P \vee (\neg Q \wedge (\neg P \vee Q)))$ is represented as follows (different occurrences of the same literal are distinguished by a superscript):

$$\begin{bmatrix} P & \begin{bmatrix} \begin{matrix} \neg P^1 \\ Q \end{matrix} & \neg Q \end{bmatrix} \\ & \neg P^2 \end{bmatrix}$$

A *path* through a matrix M is a set $\pi = \{M_1, \ldots, M_r\}$ of occurrences of submatrices (which can be literals) such that from each column in M not within one of the M_i, exactly one element is present in S.

Some paths in the example are $\pi_1 = \{P, \begin{bmatrix} \begin{matrix} \neg P^1 \\ Q \end{matrix} & \neg Q \end{bmatrix}\}$, $\pi_2 = \{P, Q, \neg Q\}$, but *not*, say, $\pi_3 = \{P, \neg P^1, \neg P^2\}$.

It is fairly easy to prove by induction that the set of formulas on any branch of any tableau for a propositonal NNF formula ψ contains a path through the matrix of ψ and, vice versa, each path through ψ is contained[6] in a branch of some tableau for ψ. Thus, complementary formulas on branches are just complementary submatrices in paths. They are called *connection* by Bibel and *mates* by Andrews.

In the NNF case, only connections between literals are possible. The paths of the example with only literals in them are π_2, $\pi_4 = \{P, \neg P^1, \neg Q\}$, $\pi_5 = \{P, \neg P^2\}$. One notices that each path contains a connection. As ψ is unsatisfiable, by soundness and completeness of tableaux and the correspondence between paths and tableau branches just stated, this is to be expected, of course. A set of connections $\mathbf{C}$ such that each path through a matrix M contains a connection from $\mathbf{C}$ was called *spanning* by Bibel [1981] who was the first to give a matrix characterization of unsatisfiable formulas (without recurrence to tableaux):

3.20. THEOREM ([Bibel 1981]). *A propositional NNF formula ψ is unsatisfiable iff there is a spanning set of connections for its matrix.*

Bibel's [1982*b*] connection method and Andrews's [1981] general matings consist of a formal notation for matrices, paths, and connections together with a systematic procedure to find a spanning set of connections. It turns out that the paths these procedures look at correspond to the branches successively generated by a certain tableau procedures.

An exact tableau counterpart to the non-clausal connection method with some additional restrictions was discussed in [Hähnle and Klingenbeck 1996]. The restriction of the connection method to clausal input corresponds exactly to weak connection tableaux with left-first branch selection discussed in Section 4.3.2 below.

Matrix methods were extended to first-order logic [Andrews 1981, Bibel 1982*b*]. In this case matrices contain additional notation to signify the kind and scope of

[6] "Contained" can be replaced by "is equal to" both times if only those formulas on a tableau branch are considered not being a subformula of another formula on the same branch.

quantifiers. True to the spirit of matrix methods, Skolem functions are avoided in [Bibel 1982*b*]. Instead, existentially quantified variables are considered as parameters that cannot be instantiated. To ensure soundness, ordering constraints on terms of the form "t may not occur as a subterm of t'" are being generated from the nesting structure of quantifiers. These constraints must then be satisfied by substitutions. The technique is independent of proof representation issues and, in fact, was employed for tableau calculi as well [Reeves 1987].

Just as γ-formulas need to be applied several times in tableau proofs, the scope of universally quantified submatrices must be present in a sufficient number of instances called *multiplicity*. Bibel [1982*b*] stresses that most of the structure of a universally quantified submatrix can be shared in an implementation. Again, the problems of proof search in destructive first-order calculi discussed in Section 3.3 are orthogonal to proof representation. Therefore, in practice, matrix methods tend to be implemented by DFID search [Bibel, Brüning, Egly, Korn and Rath 1995], just as free variable tableaux.

Matrix methods go quite some way towards an efficient implementation of deduction systems. This positive feature, on the otherhand, makes their formal presentation very technical. I suspect that this is the reason why many refinements were conceived within the more abstract—and redundant—tableau formalism.

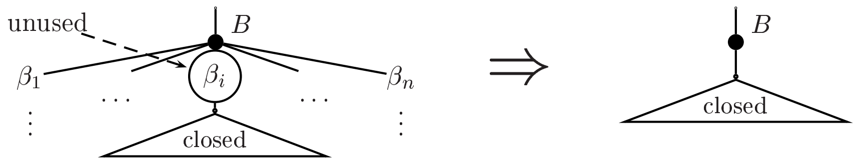
There is an important additional merit of matrix formulations of tableau methods, besides taking implementation issues seriously: for instance, there are sound transformations on the matrix level, called *reduction* in [Bibel 1982*b*], that cannot necessarily be efficiently simulated on the level of paths or branches. More generally, the global view suggested by matrices may very well lead to refinements difficult to detect with the path- or branch-based view of tableaux. Evidence of this consideration is provided by Letz [1998] who defined a tableau refinement based on the observation that one spanning *set* of connections gives possibly rise to many different tableaux that differ only in the sequence in which these connections occur on the branches.

Finally, it should be mentioned that further formula representations than matrices that try to avoid redundancy do exist: I want to mention clausal [Gallo and Urbani 1989] and non-clausal [Preiß 1998] *hypergraphs* and binary decision diagrams (BDD) [Bryant 1986]. While hypergraphs are an alternative notation for formulas and can be computed in linear time, BDDs combine normal form computation and deduction, in fact, a BDD is a normal form of a propositional formula from which its models can be directly read off. Both, hypergraphs [Preiß 1998] and BDDs [Posegga 1993] are closely related to tableaux. They share, however, the drawback that generalization to first-order logic so far has proven to be problematic [Rago 1994, Posegga and Schmitt 1995].

3.5.2. Pruning Irrelevant Parts of a Proof

Pruning, which is closely related to the *condensing* technique of Oppacher and Suen [1988], allows the reduction of both the size of the search space and the size of generated tableau proofs. It appears in the literature also under the name *level* *cut* [Baumgartner, Furbach and Niemelä 1996]. Koshimura and Hasegawa [1999] showed that condensing is a special case of the *non-Horn magic set* transformation [Hasegawa, Inoue, Ohta and Koshimura 1997] which in turn was shown [Ohta, Inoue and Hasegawa 1998] to be essentially the same as *relevancy testing* [Loveland, Reed and Wilson 1995].

Suppose a branch B of a tableau was extended by a β-rule application and one of the extensions β_i was *not* used to close the subtableau T_i below β_i, then T_i is still closed when appended to any of the other extensions β_j, $j \neq i$, or even when appended immediately to B (the extension β_i is *used* if β_i itself or any of the formulas resulting from it through tableau rule application is used in an application of the closure rule). To make use of this situation, either the closure rule is changed such that all branches in the tableau containing B as a subbranch are considered to be closed, or—similarly—all branches containing one of the β_j are *pruned*, i.e., the effects of the β-rule application are removed:



3.5.3. Simplification

The benefits of intermediate simplification steps to be applied after each tableau rule application is stressed by Massacci [1998*b*]. The idea is that for each ground formula ψ present on a branch B each positive occurrence of ψ as a subformula can soundly be replaced by *true* while each negative occurrence can be replaced by *false* with subsequent simplification steps of the form $\mathit{true} \vee \theta \Rightarrow \mathit{true}$ etc. The well-known linear unit-resolution/purity subprocedure of the Davis-Putnam-Loveland procedure [Davis, Logemann and Loveland 1962] is a special case of Massacci's [1998*b*] simplification rule who demonstrated its usefulness for (modal) propositional logic (see also Section 5.4). It remains to be seen, however, if a useful variant for free variable first-order tableau will emerge.

4. Clause Tableaux

In the present section a number of refinements of the tableau procedure are introduced. For a number of reasons, these refinements are merely discussed on the clause level:

- Simplified notation leads to easier detection of new refinements
- Efficient implementability
- Completeness proofs stay managable
- Comparability (most deduction procedures implemented on clause level)

Restricting attention to the clause level implies some limitations as well:

- Some applications (such as software verification) expect proofs on non-clausal level: backtranslation from clauses can be tricky
- For some non-classical logics a clause normal form is unknown
- Proofs become harder to read for humans
- Some applications (such as computing prime implicants) require models be preserved in which case CNF may be exponential

So there is considerable incentive to generalize the following results (partially this has been done, for example, in [Hähnle and Klingenbeck 1996, Hähnle, Murray and Rosenthal 1997]), but I think the material is more accessible in the present, syntactically limited form.

4.1. Normal Form Computation

How first-order sentences are efficiently transformed into sets of clauses is shown, for example, in [Plaisted and Greenbaum 1986, Nonnengart, Rock and Weidenbach 1998], and, of course in Chapter [Chapter `normal`] and [Chapter `small`].

4.2. Clause Tableau Proofs, Soundness, Completeness

4.2.1. Clause Tableaux

Let us start by stating suitably simplified versions of Definition 3.2 and 3.3.

4.1. Definition. Let Σ be a first-order signature. A *clause tableau* (over Σ) is a finitely branching tree whose nodes are literals from $\mathcal{L}_\Sigma$. Given a clause set S from $\mathcal{L}_\Sigma$, a clause tableau for S is constructed by a (possibly infinite) sequence of applications of the following rules:

(i) The tree consisting of a single node labeled with *true* is a tableau for S (initialization rule).

(ii) Let T be a tableau for S, B a branch of T, and $L_1 \vee \cdots \vee L_r$ an instance of $C \in S$. If the tree T' is constructed by extending B with r new subtrees and the nodes of the new subtrees are labeled with L_i, then T' is a tableau for S (extension rule).

(iii) Let T be a tableau for S, B a branch of T, and L and L' literals on B. If L and $\overline{L'}$ are unifiable with MGU σ, and T' is constructed by applying σ to all literals in T (i.e., $T' = T\sigma$), then T' is a tableau for S and branch $B\sigma$ is marked as *closed* (closure rule).

4.2. Definition. A clause tableau T for a set S is *closed* if *all* its branches are marked as closed.

A *clause tableau proof* for (the unsatisfiability of) a clause set $S \subset \mathcal{L}_\Sigma$ is a closed clause tableau T for S.

4.3. EXAMPLE. A clause form of the formula set in Example 3.4 consists of clauses (1) $\neg S(x) \vee \neg Q(x)$, (2) $\neg P(x) \vee Q(x) \vee R(x)$, (3) $P(c) \vee Q(d)$, (4) $\neg Q(x) \vee S(x)$, (5) $\neg R(x) \vee S(x)$, (6) $\neg P(x) \vee \neg R(x)$. Observe that the nodes are a subset of the nodes in Figure 2. Clause (5) is not used.

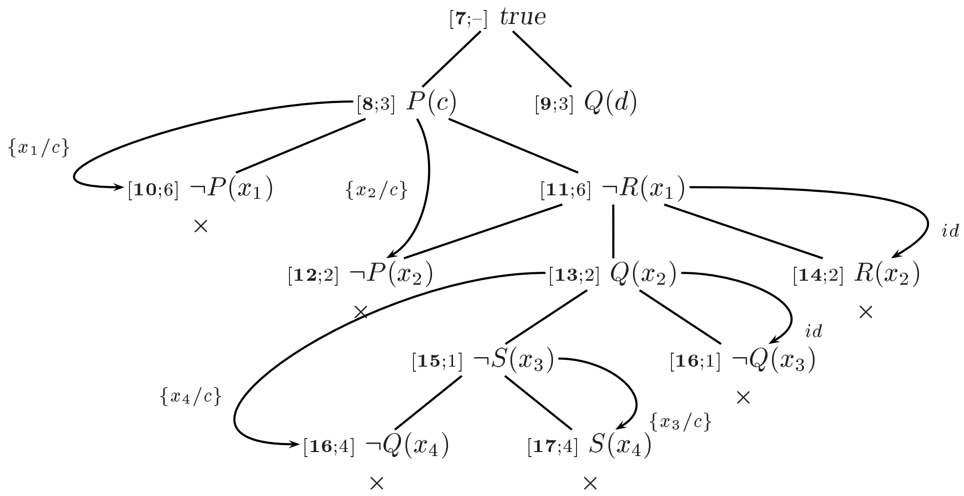


Figure 6: Partial clause tableau proof for S from Example 4.3.

Clause tableaux mainly constitute a syntactic simplification of full first-order tableaux. The main properties of the calculus are the same, in particular the discussion in Section 3.3 applies to them as well.

In contrast to the full first-order case the extension and closure rule only use literals and clauses from the input set. This simplifies some definitions.

4.2.2. Soundness and Completeness

Soundness of clause tableaux follows immediately from Theorem 3.10 by observing that clauses are particular first-order formulas and extension rule 4.1.(ii) can be composed of several applications of rule 3.2.(2).

Completeness could be obtained easily by suitable simplification of the proof of Theorem 3.19, but in the clausal case a more modular approach is useful. Following Robinson [1965*b*], lifting a ground proof to a first-order proof is separated from proving ground completeness of a calculus. The advantage is that the lifting part is similar for all following tableau refinements and must at most be sketched each time. So it is sufficient to concentrate on ground completeness. Abstraction from first-order issues greatly simplifies completeness proofs of the more complicated calculi that follow.

4.4. THEOREM (Lifting). *Let S be an unsatisfiable clause set, $\widehat{S}$ an unsatisfiable set of ground instances of S and $\widehat{T}$ a clause tableau proof for $\widehat{S}$. Then there is a clause tableau proof T for S and a substitution τ such that $\widehat{T} = T\tau$.*

PROOF. The main technical difficulty of this proof is that in Definition 4.1.(iii) only MGUs are to be used whereas $\widehat{S}$ may contain arbitrary ground instances of clauses. The following property of MGUs is needed:

$$\begin{array}{l}\text{If } T' \text{ is a clause tableau, } \tau \text{ a substitution such that } T'\tau \text{ is closed,}\\ \text{and } \rho \text{ an MGU that closes any branch of } T'\text{, then } T'\rho\tau = T'\tau.\end{array} \tag{4.1}$$

Proof of (4.1): by definition of an MGU and as τ closes T', there is τ' with $\rho\tau' = \tau$. MGUs can be assumed to be idempotent, so $T'\rho\tau = T'\rho\rho\tau' = T'\rho\tau' = T'\tau$.

Back to the main proof, let T^0 be constructed exactly as $\widehat{T}$ but for each extension step with $\widehat{C} \in \widehat{S}$ used in $\widehat{T}$, take instead an instance of the clause $C \in S$ of which $\widehat{C}$ is a ground instance. Obviously, $T^0\tau = \widehat{T}$ for a suitable ground substitution τ.

If B is an arbitrary open branch of T^0, then it is closed by τ, so there is an MGU ρ that closes B and rule 4.1.(iii) is applicable to obtain a clause tableau $T^1 = T^0\rho$. By (4.1), $T^1\tau = T^0\tau = \widehat{T}$. Repeating this argument in a straightforward induction over the number n of open branches in T^0 yields a closed clause tableau $T = T^n$ such that $T\tau = \widehat{T}$. □

In the proof the sequence of branch closures was arbitrary which shows independence of the *select branch* strategy.[7]

In the following it is sufficient to prove ground completeness for various instances X of clause tableau restrictions:

4.5. THEOREM (Ground Completeness Schema). *If the finite ground clause set S is unsatisfiable, then there is a X-tableau proof for S.*

We could proceed to prove ground completeness of unrestricted clause tableaux right now, but in following sections ground completeness of various restrictions of clause tableaux is proven of which completeness of the unrestricted calculus is an immediate consequence.

4.6. THEOREM (Completeness). *If the clause set S is unsatisfiable, then there is an X-tableau proof for S.*

PROOF. Herbrand's Theorem 2.5 provides a finite, unsatisfiable set $\widehat{S}$ of ground instances of S. By Theorem 4.5 there is a closed ground X-tableau $\widehat{T}$ for $\widehat{S}$ and, by Theorem 4.4, there is a closed X-tableau for S, whenever X has the lifting property: if $T\tau$ is an X-tableau, then T is an X-tableau as well. □

As announced already, the next goal is to find complete restrictions of clause tableaux. It is sufficient to provide a suitable instance of Theorem 4.5, whenever

[7] When *select clause* is arbitrary, but fair, the theorem still holds in the weakened form that there is a clause tableau T for S such that $T\tau$ can be homeomorphically embedded into $\widehat{T}$.

a ground X-tableau proof $\widehat{T}$ lifts to a first-order X-tableau proof T. It is usually sufficient to check that the proof of Theorem 4.4 can be used unaltered.

From the point of proof search, restricting the tableau calculus means to exclude certain choices in *select clause* and *select pair* and to fix *select branch* in some way.

4.3. Connections

Connection conditions have been pioneered by Bibel [1982*b*] and Andrews [1981].

4.3.1. Strong Connections

A major drawback of the tableau calculus is that the extension rule 4.1.(ii) is applied completely unguided which can clutter up tableaux with many nodes that do not contribute to a proof.

4.7. EXAMPLE. Consider the two clause tableaux for $S = \{P(x) \vee Q(x), R(x) \vee S(x), \neg P(a), \neg Q(a), \neg R(b), \neg S(b)\}$ displayed in Figure 7. The tableau on the right constitutes a minimal proof, while the second extension step in the tableau on the left is completely unrelated to the initial step.

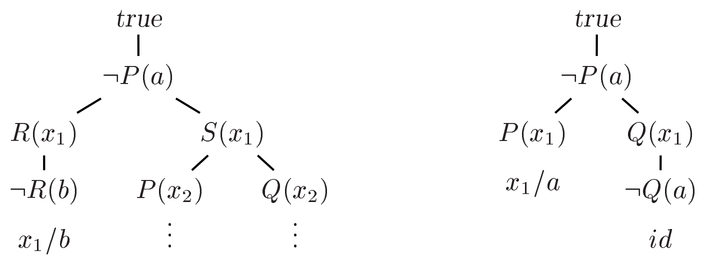


Figure 7: Redundant nodes in a tableau.

4.8. DEFINITION *([Letz, Schumann, Bayerl and Bibel 1992]).* A *connection tableau* is a clause tableau in which every non-leaf node L (except *true*) has $\overline{L}$ as one of its immediate successors.

The tableau on the right in Figure 7 is a connection tableau, the tableau on the left is not. It is excluded by the connection restriction.

Connection tableaux are complete, but the proof is deferred until the next section. The definition of connection tableaux implies that when $T \neq$ *true* at least one of the new branches generated by the tableau extension rule can be closed. This suggests a *procedural* definition of connection tableaux obtained from Definition 4.1 by changing the first two rules:

(i′) For any instance $L_1 \vee \cdots \vee L_r$ of $C \in S$ the tree constructed by extending *true* with r new subtrees with nodes L_i is a connection tableau for S.

(ii′) Let T be a tableau for S, B a branch of T ending with L, $L_1 \vee \cdots \vee L_r$ an instance of $C \in S$. If $\overline{L}$, L_i (where $i \in \{1, \ldots, r\}$) are unifiable with MGU σ and the tree T' is constructed by extending B with r new subtrees, where the nodes of the new subtrees are the L_j, then $T'\sigma$ is a connection tableau for S, in which the branch ending with $L_i\sigma$ is marked as closed. This is called a *strongly connected extension step*.

Closure of open branches (Definition 4.1.(iii)) is unchanged and called *reduction step*. Note that, besides in reduction steps, branches can be in addition be closed in extension steps. It is important to note that while clause tableaux are proof confluent independently of the choice of *select clause*, connection tableaux are not:

4.9. Proposition. *Ground connection tableaux are not proof confluent.*

Proof. Consider $S = \{P, \neg P, Q\}$ and let Q be the initial clause. It impossible to make any further extension step, although S is clearly unsatisfiable. (Examples independent of the choice of the initial clause can be found in [Letz 1993].) □

In Section 3.3 it was pointed out that a "greedy" strategy for preferring closure over expansion steps leads to incompleteness. It is tempting to employ a greedy strategy at least for closures occurring within reduction steps, but the following counter example due to Letz[8] shows that even this results in incompleteness:[9]

4.10. Example. If $S = \{P(a) \vee P(x) \vee Q(x), \neg Q(b) \vee R, \neg P(b) \vee R, \neg R\}$, then $\{\neg P(a)\} \cup S$ is unsatisfiable.

A proof starting with $\neg P(a)$ must use $P(a) \vee P(x) \vee Q(x)$ in the first extension step. A left-first *select branch* rule forces reduction with $P(x)$ and the proof is stuck. With different starting clause, a greedy reduction proof is possible, but a trick taken from [Letz, Mayr and Goller 1994] gives a general counter example. Let S' be as S, but P replaced with P', Q with Q' and R with R'. Then $\{\neg P(a) \vee \neg P'(a)\} \cup S \cup S'$ is still unsatisfiable. Regardless of the starting clause, however, $\neg P(a) \vee \neg P'(a)$ must be used at some point. This cannot be the last extension step in a proof, because the signatures of S and S' are disjoint. Therefore, the proof gets stuck in the same way as above.

4.3.2. Weak Connections

The extension rule (ii′) of connection tableaux, however, has a natural relaxation that restores proof confluence:

(ii″) Let T be a tableau for S, B a branch of T *containing L not necessarily as leaf*, $L_1 \vee \cdots \vee L_r$ an instance of $C \in S$. If $\overline{L}$, L_i (where $i \in \{1, \ldots, r\}$) are unifiable with MGU σ and the tree T' is constructed by extending B with r new subtrees, where the nodes of the new subtrees are the L_j, then $T'\sigma$ is a

[8] Personal communication.

[9] In fact, the authors of SL-resolution [Kowalski and Kuehner 1971], discussed as a close relative to connection tableaux in Section 5.2, were tempted enough: they suggested to reduce greedily without noting the incompleteness problem.

clause tableau for S, in which the branch ending with $L_i\sigma$ is marked as closed. (This is called a *weakly connected extension step.*)

Let us call the resulting calculus *weak connection tableaux.*

4.11. DEFINITION. A clause set is *minimally unsatisfiable (mu)* when it is unsatisfiable and each of its proper subsets is satisfiable. A clause is *relevant* in S when it is contained in a mu subset of S.

It can be shown that weak connection tableaux are proof confluent provided that *select clause* is implemented in a fair manner and the initial clause is relevant. Unfortunately, testing for membership in a mu set is as expensive as testing unsatisfiability itself. This limits the usefulness of weak connection tableaux in practice, but in Section 4.5 a slight relaxation will build the basis of a whole class of interesting calculi which are proof confluent regardless of the initial clause.

4.4. Regularity

An important device in tableau-based theorem proving that avoids to construct certain redudant proofs is *regularity*:

4.12. DEFINITION. A clause tableau is *regular* when (i) all of its branches contain at most one occurrence of the same literal and (ii) no tautologous clause is used in any extension step.

4.13. EXAMPLE. Regularity can help to avoid substitutions that lead to redundant proofs. Consider the tableau for $S = \{P(0), \neg P(x) \vee P(s(x)), \neg P(s(s(0)))\}$ in Figure 8. The first possible substitution for the middle branch renders the right branch irregular and is thus avoided.

Implementing regular tableaux is not straightforward, because an admissible closure substitution can potentially unify as well formerly different literals on branches closed already. For efficiency reasons one discards closed branches immediately, so there must be a mechanism to exclude such critical substitutions. It was suggested in [Letz et al. 1992] to create an inequality constraint of the form $t_1 \neq t'_1 \vee \cdots \vee t_m \neq t'_m$, whenever two unifiable literals $(\neg)P(t_1, \ldots, t_m)$ and $(\neg)P(t'_1, \ldots, t'_m)$ are encountered on one branch. Similar constraints are generated to characterize tautologous instances of clauses used in extension steps. Then substitutions are applied to constraints as well and must ensure their satisfiability. In the example above, the second extension step generates the constraint $s(0) \neq s(x_2)$ which is not satisfied by $x_2/0$.

The following standard lemma is needed in the proof of of ground completeness of regular connection tableaux. Its easy proof is given, for instance, in [Loveland 1978, Lemma 2.3.2, p. 63]. The completeness theorem was first proven (differently) in [Letz 1993]. The present proof is due to [Hähnle et al. 1997].

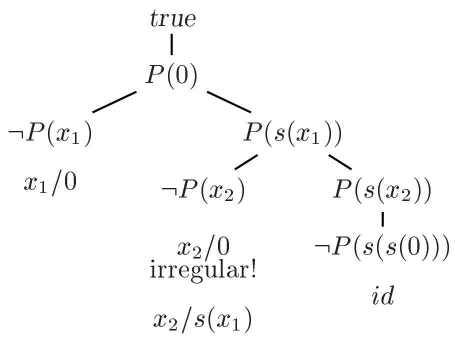


Figure 8: Advantage from regularity.

4.14. LEMMA. *Let S be a mu ground clause set with $C \in S$, $D \subseteq C$, and $S_D = (S - \{C\}) \cup \{D\}$. Then for any mu subset S'' of S_D: (i) $D \in S''$; (ii) $D \not\subseteq D''$ for all $D \neq D'' \in S''$.*

4.15. THEOREM. *If the finite ground clause set S is unsatisfiable, then there is a regular connection tableau proof for S.*

PROOF. We show by induction on the number k of literal occurrences in S: for any relevant non-unit clause $C_j \in S$, there is a closed regular connection tableau for S whose initial step uses C_j. When there is no such clause there must be a mu subset of unit clauses in S; it is trivial to find a regular connection tableau proof for such a set.
$k \in \{0, 1, 2\}$: either S is satisfiable or it contains only unit clauses or the empty clause and the claim is trivially satisfied.
$k > 2$: (see Figure 9) let $C_j = L_1 \vee \cdots \vee L_i \vee \cdots \vee L_n$ be a relevant non-unit clause in S and let T be the regular connection tableau resulting from an initial step with C_j.

For all $i \in \{1, \ldots, n\}$ let $S_{L_i} = (S - \{C_j\}) \cup \{L_i\}$. By Lemma 4.14.(i), $C'_j = L_i$ is contained in an mu subset S'_i of S_i. Hence, $\overline{L_i}$ occurs in a clause C^i of S'_i. Moreover, S'_i contains less literals than S.

If C^i is a unit clause, then the i-th branch of T can be closed immediately, resulting in a regular connection tableau. Otherwise, applying the induction hypothesis on C^i and S'_i yields a closed regular connection tableau T_i for $S_i \subseteq S'_i$, where the first extension step uses C^i.

By Lemma 4.14.(ii), L_i occurs at most in C'_j and is therefore only used in unit extension steps with C'_j in T_i (highlighted by boldface type in the figure). As shown in the figure, each T_i is glued together with T at L_i maintaining connectedness. In the resulting tableau irregularity can at most occur with the L_i. But as L_i occurs on top of each T_i (circled occurrence) the unit extension steps with L_i simply can be replaced by reduction steps with the circled occurrence of L_i. As $S_i - \{C'_j\} \subseteq S$, the result is a regular connection tableau for S. □

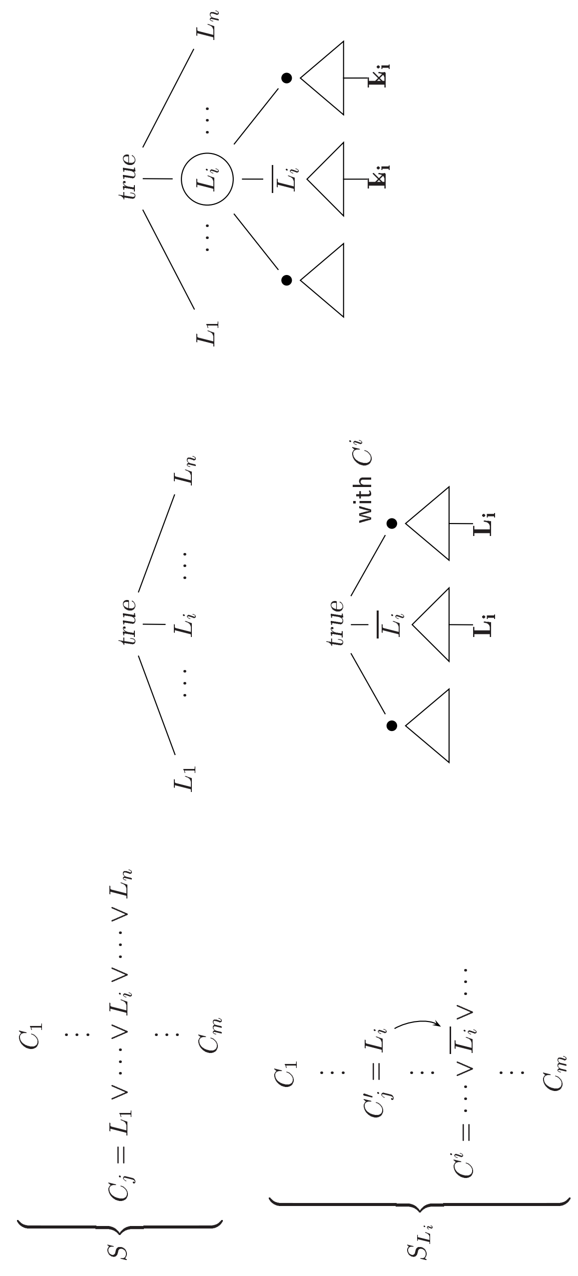


Figure 9: Illustration of the proof of Theorem 4.15

.

4.5. Orderings and Selection Functions

4.5.1. Redundancy and Saturation in Tableaux

Let us take up the theme expressed in the regularity restriction, namely to avoid redundancy in tableau proofs.

Any open branch B in a ground clause tableau T for S, or equivalently, any consistent set of ground literals B defines a partial *interpretation* $\mathbf{I}_B$ on S via $\mathbf{I}_B \models L$ iff $L \in B$.

A clause tableau is rendered irregular by an extension step of branch B with clause C iff $\mathbf{I}_{\exists B} \models C$, where $\exists B$ is obtained from B by replacing its variables with new and differing constant symbols. When B is ground, regularity amounts therefore to avoid extension whenever $\mathbf{I}_B \models C$ holds. A stronger notion of redundancy is desirable, though. Until further notice we work with ground clauses.

4.16. DEFINITION. An open clause tableau branch B has a *saturation* wrt a clause set S iff it has an extension $\widetilde{B} \supseteq B$ such that $\mathbf{I}_{\widetilde{B}} \models S$.

It is, of course, not realistic to consider *all* possible extensions of a branch (the empty branch, for example, always has a saturation when S is satisfiable), so we check only one of them to guide tableau extension. The idea is to consider extensions of B that contain certain literals from all clauses of S that keep the current tableau regular. These literals are determined by a selection function.

4.5.2. Tableaux with Selection Function

Let f be a *selection function* on clauses: a function mapping each clause into a (possibly empty) subset of its literals. The *extension* $\widetilde{B}_f$ *of* B *with respect to* f is defined as follows:

$$\widetilde{B}_f = B \cup \{f(C) \mid C \in S, \mathbf{I}_B \not\models C\} \tag{4.2}$$

If $\widetilde{B}_f$ is consistent and all clauses with no selected literals were used on B, then $\mathbf{I}_{\widetilde{B}_f} \models S$ by construction and proof search can be stopped here. In general, however, $\widetilde{B}_f$ does not induce an interpretation, because it may contain complementary literals. In this case, one of the clauses not yet satisfied by $\mathbf{I}_B$ is selected for extension whose selected literal(s) contribute to a contradiction in $\widetilde{B}_f$. Formally, let $\overline{f(C)}$ denote the set of complements of literals selected by f in C. Then extension steps (Definition 4.1.(ii)) are restricted to clauses in

$$\{C \mid C \in S, \mathbf{I}_B \not\models C \text{ and } (\overline{f(C)} \cap \widetilde{B}_f \neq \emptyset \text{ or } f(C) = \emptyset)\} \tag{4.3}$$

By definition, if $L \in \overline{f(C)} \cap \widetilde{B}_f$, either $L \in B$ or $L \in f(D)$ for some clause $D \in S$. In the first case the extension is a *weak connection step* in the sense of (ii$''$) on page 27 (accordingly, the branch containing $\overline{L} \in f(C)$ is marked as closed). The second case and the case when no literal is selected are called a *restart step* (and C a *restart clause*) to emphasize that this part of a tableau proof bears no direct connection with the current branch. The top clause in a tableau proof is always a restart step. No new restart clauses are added once a selection function f is fixed: they can be computed in advance.

4.17. EXAMPLE. Consider the clause set $S_1 = \{\neg Q \vee \underline{\neg S},\ \neg R \vee \underline{S},\ P \vee Q \vee \underline{R},\ \underline{\neg P}\}$ in which the lexicographically largest literal is selected (these are underlined). Initially, $B = \{\mathit{true}\}$ and $\widetilde{B}_f = \{\neg S, S, R, \neg P\}$. The first two clauses are the only restart clauses of which the first is selected (see Figure 10). Branch $B = \{\neg Q\}$ implies $\widetilde{B}_f = B \cup \{S, R, \neg P\}$ which models S_1. On the other branch $B = \{\neg S\}$ one has $\widetilde{B}_f = B \cup \{S, R, \neg P\}$, so an extension step (the only one) with the second clause is possible. The only open branch is now $B = \{\neg S, \neg R\}$ with $\widetilde{B}_f = B \cup \{R, \neg P\}$ and only extension with the third clause is allowed. The first open branch, $\{\neg S, \neg R, P\}$ is closed by a further extension while the last open branch $B = \{\neg S, \neg R, Q\}$ yields $\widetilde{B}_f = B \cup \{\neg P\}$ and thus the second model of S. Observe that all but the very first extension step were determined which demonstrates the potential for downsizing the search space with selection functions.

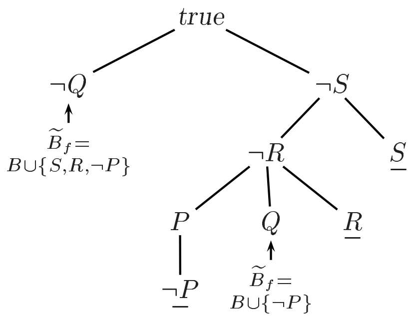


Figure 10: Tableau with selection function.

4.18. THEOREM. *If the finite ground clause set S is unsatisfiable, then for any selection function f there is a tableau proof with selection function f for S.*

PROOF. Assume there is a tableau with selection function f and an open branch B in which all possible extension steps were made. As S is finite, B is finite as well. We claim that $\widetilde{B}_f$ is consistent from which $\mathbf{I}_{\widetilde{B}_f} \models S$ follows by construction (in particular, all clauses with no selected literal are satisfied). If $\widetilde{B}_f$ is inconsistent there is a literal $L \in \{f(C) \mid C \in S, \mathbf{I}_B \not\models C\}$ such that $\overline{L}$ occurs (I) in B or (II) in $f(D)$ for some clause D not yet satisfied by B. In case (I) a weakly connected extension step is possible on B in case (II) a restart step is admissible. Either way, the assumption that all possible extension steps were made on B is contradicted. $\square$

The proof is independent of the sequence of extension steps chosen, so tableaux with selection function are *proof confluent*. Moreover, a slight generalization of the proof shows that completeness is retained even when f is changed during tableau construction.

4.5.3. Related Calculi

A number of recently suggested restrictions of clause tableaux can be considered as special cases of tableaux with selection function, for example, ordered tableaux [Klingenbeck and Hähnle 1994, Hähnle and Klingenbeck 1996].

4.19. Definition. A ground *literal (L-)ordering* is a binary relation $<$ on ground literals which is irreflexive and transitive.

L-orderings give a complete tableau restriction which can be expressed via selection functions as follows: simply use $f_<(C) = \{L \mid L \text{ maximal in } C \text{ wrt } <\}$

The regularity restriction $\mathbf{I}_B \not\models C$ in (4.2), (4.3) is not enforced by Hähnle and Pape [1997] and $|f(C)| > 0$ is required, otherwise their calculus is identical to the present version of tableaux with selection function. On the other hand, Pape and Hähnle [1997] showed that tableaux with selection functions can be modified to accommodate strongly connected extension steps.

In [Hähnle and Klingenbeck 1996] it is shown that ordered tableaux without restart steps (that is *each* extensions step is weakly connected via f) are incomplete. The same holds for tableaux with selection function as they can be instantiated to ordered tableaux. In some cases, however, restart steps are naturally restricted:

4.20. Definition. A selection function is *consistent* (with respect to S) if $\bigcup_{C\in S} f(C)$, the set of all its selected literals in a clause set S, is consistent.

Whenever f is consistent with respect to S, only restart steps with clauses having no selected literals are possible. The special case of S being consistent, $|f(C)| \leq 1$, and there being exactly one restart clause in S, is known in the literature as *SL-resolution without contrapositive clause variants* (*SLWV-resolution*) [Pereira, Caires and Alferes 1992].

4.5.4. First-Order Issues

Lifting is straightforward for tableaux with selection function f provided that f lifts. Call f *stable wrt substitutions* if $f(C\sigma) \subseteq f(C)\sigma$ for all clauses C and substitutions σ. For L-orderings this translates into the requirement $L < L'$ implies $L\sigma < L'\sigma$ for all substitutions σ and literals L, L'.

Implementation of first-order tableaux with selection function generates similar problems as regularity. Further constraints than regularity are derived from (4.3) and take on the form $L \in f(C)$ [Pape 1996, Hähnle and Pape 1997].

Checking these latter constraints for satisfaction can be expensive (NP-complete), so one may decide to suppress their generation. The resulting calculus might be called *tableaux with input selection function* [Hähnle and Pape 1997], because the selection restriction is only enforced on clauses that serve as input for extension steps, but not on instances of clauses appearing in a tableau. This has another advantage: it was noted after the proof of Theorem 4.18 that the selection function may be arbitrarily changed during tableau construction. This implies immediately that selection functions need not be stable wrt substitutions in input selection tableaux, rather, stability wrt variable renamings is sufficient [Hähnle and Pape 1997].

4.6. Hyper Tableaux

Recently, tableau calculi based on hyper extension rules have gained considerable attention [Bry and Yahya 1996, Baumgartner et al. 1996, Shults 1997, Baumgartner 1998]. This is not surprising as hyperresolution [Robinson 1965*a*] is long known to be a key ingredient to success in theorem proving. Hyper calculi share the feature that several deduction steps are combined into one. This yields a speed-up in proof search, but the main advantage is that some intermediate results are not computed in the first place and this can limit the search space considerably. In fact, hyper tableaux were considered early on by Brown [1978], but this work did not make the impact it deserved. The family of calculi known as model generation [Manthey and Bry 1988, Fujita and Hasegawa 1991] is essentially a variant of hyper tableaux and will be discussed here as well.

It is well-known that hyperresolution can be seen as an instance of semantic resolution [Slagle 1967]. The same kind of generalization is done in the following for hyper tableaux.

Again, we start with the ground case. One stipulates a similar condition as (4.3) on extending clauses saying that *all* selected literals of an extending clause must be weakly connected to the current branch, formally, each clause used in an extension step (Definition 4.1.(ii)) on B must be from the set:

$$\{C \mid C \in S, \mathbf{I}_B \not\models C \text{ and } \overline{f(C)} \subseteq B\} \tag{4.4}$$

A ground clause tableau constructed with this restriction is called a *hyper tableau*. In each extension step the branches containing complements of selected literals of the extending clause are marked as closed.

Given a selection function f and a clause C with $f(C) = \{L_1, \ldots, L_m\}$ and $C - f(C) = \{L_{m+1}, \ldots, L_n\}$, one can rewrite C as a rule in the following fashion:

$$\overbrace{\overline{L_1} \wedge \cdots \wedge \overline{L_m}}^{\text{selected literals}} \rightarrow \overbrace{L_{m+1} \vee \cdots \vee L_n}^{\text{unselected literals}} \tag{4.5}$$

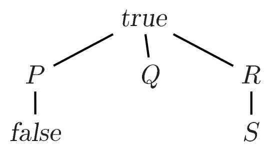
The premiss (the set of literals $\{\overline{L_1}, \ldots, \overline{L_m}\}$) becomes $\{\mathit{true}\}$ when there are no selected literals, the conclusion (the set of unselected literals $\{L_{m+1}, \ldots, L_n\}$) becomes $\{\mathit{false}\}$ when there are only selected literals. Note that the premiss of a rule contains *complements* of selected literals.

As *true* occurs on every tableau branch, the following reformulation of (4.4) is possible:

$$\{C \mid C \in S, \mathbf{I}_B \not\models C \text{ and all literals of the premiss of } C \text{ occur in } B\} \tag{4.6}$$

Clauses with no selected literals are unrestricted in application and, therefore, play the rôle of *restart clauses* in hyper tableaux. In the following the convention is adopted not to display branches closed by extension steps and to treat *false* as a literal indicating closure by clauses having only selected literals.

4.21. EXAMPLE. Let f_N select exactly the negative literals of each clause. Ground clause set $S_2 = \{\underline{\neg Q} \vee \underline{\neg S}, \underline{\neg R} \vee S, P \vee Q \vee R, \underline{\neg P}\}$ in rule notation becomes: $\{\underline{Q} \wedge \underline{S} \rightarrow \mathit{false}, \underline{R} \rightarrow S, \mathit{true} \rightarrow P \vee Q \vee R, \underline{P} \rightarrow \mathit{false}\}$. The only possible hyper tableau for f_N and S_2 is as follows:



Any hyper tableau for S must begin with the only restart clause. Closure of the leftmost branch and extension of the rightmost one are mandatory either. No more rule can be applied after this.

In general, hyper tableaux are not complete (for example, when *all* literals are selected no initial extension step can be made), but completeness is regained for consistent selection functions.[10]

4.22. THEOREM. *If the finite ground clause set S is unsatisfiable, then for any consistent selection function f there is a hyper tableau proof with selection function f for S.*

PROOF. The proof is very similar to the proof of Theorem 4.18. Consider a maximal, open branch B in a hyper tableau for S and its extension $\widetilde{B}_f$, see (4.2). The set $\widetilde{B}_f - B$ is consistent, because f is consistent, so when $\widetilde{B}_f$ is inconsistent there are literals $L \in \{f(C) \mid C \in S, \mathbf{I}_B \not\models C\}$ such that $\overline{L}$ occurs in B. Obtain B' by removing all such literals from $\widetilde{B}_f$. Now B' is consistent and we claim $\mathbf{I}_{B'} \models S$. By construction $B' \supseteq B$, so it suffices to prove $\mathbf{I}_{B'} \models C$ for clauses C not used on B (this implies $|f(C)| > 0$).

$\widetilde{B}_f$ contains the literals from $f(C)$; if all of these were removed when computing B' then $\overline{f(C)} \subseteq B$ and a hyper extension step would be possible with C on B contradicting the assumption that all possible extension steps were made. So at least one literal in $f(C)$ is still present in B', thus $\mathbf{I}_{B'} \models C$. $\square$

[10]In fact, for consistent selection functions that select at most one literal per clause conditions (4.3) and (4.4) become identical.

Obviously, it holds that $B' = B \cup \{L \in f(C) \mid C \in S, \overline{L} \notin B\}$.

4.6.1. Positive and Semantic Hyper Tableaux

Many hyper tableau calculi [Manthey and Bry 1988, Fujita and Hasegawa 1991, Brown 1978, Baumgartner et al. 1996, Bry and Yahya 1996, Shults 1997, Baumgartner 1998] focus on the particular selection function f_N which selects exactly the negative literals in a clause. The ensuing calculi are called *positive hyper tableaux*, because negative literals occur only as leaves of branches closed by an extension step in a hyper tableau with f_N. Therefore, closure by reduction steps cannot occur. If Σ contains the atoms of S, then computing B' simplifies to $B \cup \{\neg P \mid P \in (\mathcal{A}_\Sigma - B)\}$. From the two open branches in the tableau of the previous example one reads off the models $\{Q, \underline{\neg S}, \underline{\neg R}, \underline{\neg P}\}$ and $\{R, S, \underline{\neg Q}, \underline{\neg P}\}$.

More generally, a *complete selection function*[11] for a ground clause set S selects at least one of $P, \neg P$ for each ground atom in the signature of S.[12] In the presence of complete selection functions reduction steps are not required, because they would occur with two complementary, unselected literals whose existence is exactly what has been ruled out;[13] this proves:

4.23. COROLLARY. *If the finite ground clause set S is unsatisfiable, then for any consistent and complete selection function f there is a hyper tableau proof with selection function f for S that does not use the reduction rule.*

4.24. EXAMPLE. Consider $S = \{true \rightarrow P,\ true \rightarrow \neg P\}$ in which nothing is selected. Both clauses are restart clauses, so P and $\neg P$ are present on any branch, however, closure is only possible with a reduction step.

4.6.2. First-Order Issues

For sake of clarity, the first-order version of the *hyper tableau extension* rule is stated explicitly:

(ii‴) Let T be a hyper tableau for S and consistent selection function f, B a branch of T, $C = \overline{L_1} \wedge \cdots \wedge \overline{L_m} \rightarrow L_{m+1} \vee \cdots \vee L_n$ an instance of a clause in S. If there is an MGU σ such that $\mathbf{I}_{\exists B\sigma} \not\models \forall C\sigma$, all literals of the premiss of $C\sigma$ occur in $B\sigma$ (see also (4.6)) and the tree T' is constructed by extending B with r new subtrees, where the nodes of the new subtrees are the literals of C, then $T'\sigma$ is a hyper tableau for S and f, in which all new branches ending with $L_i\sigma$ ($1 \leq i \leq m$) are marked as closed.

---

[11] This notion of completeness is unrelated to completeness of calculi.

[12] Consistent and complete selection functions on S can be considered as interpretations of S ($\mathbf{I}_f \models L$ iff $L \in f(C)$). Hyper tableaux based on such f can be seen as a tableau counterpart to *semantic resolution* [Slagle 1967] and would be best called *semantic* tableaux if the latter phrase were not used sometimes for the whole tableau framework. Perhaps one should call them *semantic semantic tableaux*?

[13] Alternatively, one argues that consistent and complete selection functions f induce a suitable literal renaming of S on which f_N can be used to the same effect as f.

As before, selection functions should be liftable that is stable wrt substitutions. In addition to the easy regularity check, for a hyper extension step on branch B with clause C, one must compute a $B_0 \subseteq B$ as well as a simultaneous MGU of $\{\{L, L'\} \mid L \in B_0, L' \in \overline{f(C)}\}$. This is easily seen to be an instance of the first-order clause subsumption problem which is NP-hard [Garey and Johnson 1979]. This complexity is implicitly present in proof search without hyper steps as well, although on a different level. It certainly does not indicate inferiority of hyper tableaux.

A more interesting question is: how is dealt with completeness problems arising from difficulty of fair substitution selection in destructive calculi? For positive hyper tableaux, several strategies can be found in the literature:

The first option is to fix a heuristics for formula selection and trade incompleteness for success in certain problem domains [Brown 1978, Shults 1997]. Another form of incompleteness (in syntactical expressivity) ensues from restriction to range-restricted sets of clauses:

4.25. Definition. A first-order clause C is *range-restricted* with respect to a selection function f when all variables occurring the conclusion of C occur also in its premiss.

Observe that range-restricted clauses with no selected literals (premiss is $\{true\}$) are ground. A trivial induction shows that when all clauses in S are range-restricted, then a hyper tableau for S is ground. As an immediate consequence, in this case hyper tableaux are not destructive, so fair selection of clauses used in extension steps renders hyper tableaux a strongly complete calculus of which efficient implementations were realized with respect to f_N [Manthey and Bry 1988, Fujita and Hasegawa 1991, Hasegawa, Koshimura and Fujita 1992, Bry and Yahya 1996, Hasegawa, Fujita and Koshimura 1997], where reduction steps are not necessary (Corollary 4.23). These calculi are further discussed in Section 4.6.3 below.

Back to general clause sets, call the variables violating range-restrictedness *and* occuring in more than one literal of the conclusion of a clause *critical*. Obviously, non-critical variables in a tableau are *universal* in the sense of Section 3.2.5 and need, therefore, not be instantiated. Baumgartner et al. [1996] enumerate ground instances of clauses restricted to critical variables to obtain a strongly complete calculus which is somewhat better than enumerating all ground instances as in Smullyan's [1995] tableaux.

Baumgartner [1998] improves on this: like in the range-restricted case, a tableau is treated as if it were ground: substitutions are only applied to instances of input clauses (matching). If an instance $L\sigma$ (with critical variables) of a literal occurrence L on a tableau branch is required, then σ is applied to an instance of the clause containing L and the result is added to the input clause set. The latter possibility regains completeness.[14] Needless to say, lifting is not trivial in such a calculus.

[14]This idea can be applied to any tableau calculus, not only to hyper tableaux. They are particularly suitable, though, because less clause instances are generated.

This version of first-order hyper tableaux is not destructive, as only input clauses are instantiated. The destructive part of the substitution of the closure rule is recorded "outside" of the tableau as additional instances of input clauses. They can be arranged in a fair manner easily. This can also be seen as a kind of constraint on substitutions to guide the proof search.

In principle, a strongly complete, *destructive* calculus could be obtained if one compiled these "outside" clauses (and their fair selection) into the selection rule. Consequences for such a calculus would be: (I) clause selection is probably not defined branch-local, because "outside" clauses touch on several branches; (II) one needs to work up to renamability of clauses; (III) a suitable ordering must be used to enumerate required instances of literals in a fair manner. All three ingredients are actually contained in a recent suggestion for a strongly complete, destructive calculus by Beckert [1998] and by Baumgartner, Eisinger and Furbach [1999, to appear], the first of which is discussed in Section 4.7.

4.6.3. Model Generation

Positive hyper tableaux for range-restricted clause sets are better known as *model generation* and were suggested by Manthey and Bry [1988]. Traditionally, they are described in a somewhat different manner (adapted from [Fujita and Hasegawa 1991]):

4.26. DEFINITION. $\mathcal{I}$ is an inductively defined set of ground interpretations called **model candidates** each of which is represented by a consistent set of ground literals.

Init: Set $\mathcal{I}$ to $\{\emptyset\}$

Model Extension: $\mathbf{I}$ is a model candidate, $D \to E$ an instance of a clause in S. If there is a substitution σ such that $D\sigma \subseteq \mathbf{I}$ and $\mathbf{I} \not\models E\sigma$, then set $\mathcal{M}$ to

$$(\mathcal{M} - \{\mathbf{I}\}) \cup \bigcup_{L \in E\sigma} (\{L\} \cup \mathbf{I})$$

Model Rejection: $\mathbf{I}$ is a model candidate, $D \to \mathit{false}$ an instance of a clause in S. If there is a substitution σ such that $D\sigma \subseteq \mathbf{I}$, then delete $\mathbf{I}$ from $\mathcal{M}$.

Obviously, model candidates correspond to open hyper tableaux branches while model extension and model rejection are special cases of the positive hyper tableau extension rule. Model candidates that can neither be extended nor rejected correspond to open hyper tableau branches with no applicable rule and, therefore, induce a model of the input clause set. A closed tableau corresponds to the empty set of model candidates.

In the light of Corollary 4.23, model generation works unaltered for range-restricted rule sets with consistent and complete selection functions. The latter must be defined on the first-order level, of course:

4.27. DEFINITION. A selection function f is *consistent* on a first-order clause set S if it is consistent on all its ground instances. It is *complete* on S if for each ground instance p of an atom occuring in S: f selects a literal q or $\neg q$ such that p is an instance of q.

A further generalization of model generation was obtained by Shirai and Hasegawa [1995] and called *constraint model generation*. It can be described in the present framework as hyper tableaux with arbitrary selection function and range-restricted input. Theorem 4.22 (plus a trivial lifting step) grants completeness for fair input clause selection and consistent selection functions, when a reduction rule is present. But the latter can be expressed *within* the calculus by adding a clause

$$P(x_1, \ldots, x_n) \wedge \neg P(x_1, \ldots, x_n) \rightarrow \mathit{false} \tag{4.7}$$

for each n-ary predicate symbol P. The rules (4.7) were called *integrity constraints* by Shirai and Hasegawa.

Whether constraint model generation is complete for a given problem S was unclear in general, because the selection function induced by the rules S might not be consistent. Theorem 4.22 ensures completeness for each problem with a consistent set of premisses (with the exception of the problem-independent rules (4.7)). The quasi-group problems discussed in [Shirai and Hasegawa 1995] are a practically relevant case: rules (M1–M12) indeed have consistent premisses. (Rules (M13–M18) have not, but they are derived rules unnecessary for completeness.)

If necessary, any clause set can be made consistent wrt any given selection function f: assume there is a rule $R = L \wedge C \rightarrow D$ such that L is unifiable with $\overline{L'}$ occuring in the premiss of another rule. Replace R with $R' = d(\vec{x}) \wedge C \rightarrow D \vee \overline{L}$, where $\vec{x}$ are the variables occurring in L but not in C, and d is a *domain predicate* that enumerates the ground terms of the problem signature [Manthey and Bry 1988]. This preserves satisfiability and range-restrictedness and eliminates the inconsistency at L.

4.7. A Destructive and Strongly Complete First-Order Calculus

Recall from the discussion in Section 3.3 that destructive tableau calculi cannot easily be implemented in a proof confluent way so that one usually retracts to DFID search, although backtracking is not necessary in principle.

It is the closure rule (Definition 4.2.(iii)) that renders tableaux with free variables destructive, but the MGUs computed in it are needed for guidance of proof search.

Baumgartner's [1998] idea was to record MGUs outside of the tableau in the form of a constantly growing input clause set. Beckert [1998] takes a different approach which does not modify the input clause set.[15]

Whenever a substitution σ must be applied to a tableau T, the smallest subtableau T' affected by σ (that is, containing variables σ acts upon) is *reconstructed*.

[15]Beckert's framework is designed for quite general tableaux calculi including non-classical and non-clausal logics. I present it in simplified form for clause logic, because the technicalities of the general case are beyond the scope of this article.

This can be done simply by copying T' below each open branch of $T'\sigma$ as depicted in Figure 11, but obviously less redundant strategies are conceivable.

The effect is that substitution is syntactically still destructive, but from a semantical point of view absolutely no harm is done.

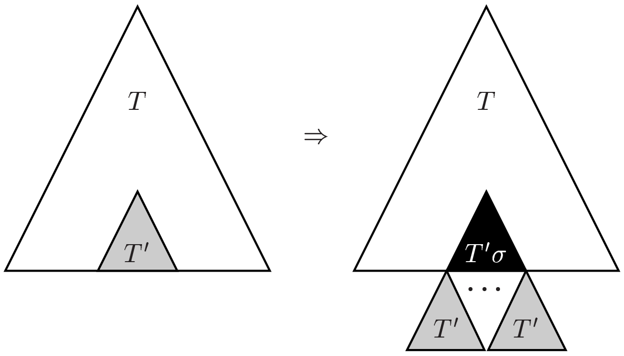


Figure 11: Reconstructing a subtableau "destroyed" by substitution.

Select clause and *select pair* must still be fair, of course, but this is much easier to achieve once destructivity is essentially eliminated. Unfair selection can result in two phenomena which have to be both avoided: (i) generating arbitrarily large terms of one kind (such as $s^n(x)$ for all n, when also, say, 0 is present); (ii) avoid loops of tableaux that subsume each other.

To avoid unfair generation of terms one restricts the order in which they can be introduced. A *well-order* $<$ on literals is any partial, well-founded order, such that there are not infinitely many incomparable elements (up to variable renaming).

4.28. EXAMPLE. Let $|L|$ denote the number of symbols in L, where variables are counted twice. Then define $L < L'$ iff $|L| < |L'|$ and *false* $< L$ for all L. This implies $P(a) < P(x) < \neg P(y)$.

Now the effect of each rule, when applied to a tableau, can be measured in terms of $<$:

4.29. DEFINITION. With each tableau construction rule R applied to tableau T one associates a set of literals $R(T)$, depending on the rule type:
Extension: the set of literals in the extending clause;
Closure: $\{\mathit{false}\} \cup \{L\sigma \mid L \text{ occurs in } T,\ L\sigma \neq L\}$, when closure is by MGU σ.

It remains to assure that each tableau rule application derives "new" information so that progress towards a proof is not infinitely delayed. In tableau calculi progress can be measured in terms of branch closure. If a tableau can be closed, and it turns out that its predecessor could have been closed as well, then this tableau is not needed. The following definition formally captures this:

4.30. DEFINITION. A tableau T *subsumes* a tableau T' if each branch of T "subsumes"[16] at least one branch of T'.

Putting things together, one obtains:

4.31. THEOREM ([Beckert 1998]). *If S is an unsatisfiable first-order clause set and $<$ a well-order on literals, then* any *sequence of first-order clause tableaux for S results in a tableau proof for S after finitely many steps provided that (A) a reconstructing version of the closure rule is used and (B) a rule R is only be applied to a tableau T when*

(i) the $<$-maximal elements of $R(T)$ are $<$-minimal in $\displaystyle\bigcup_{R' \text{ applicable on } T} \max R'(T)$

(ii) T does not subsume the result of applying R to T

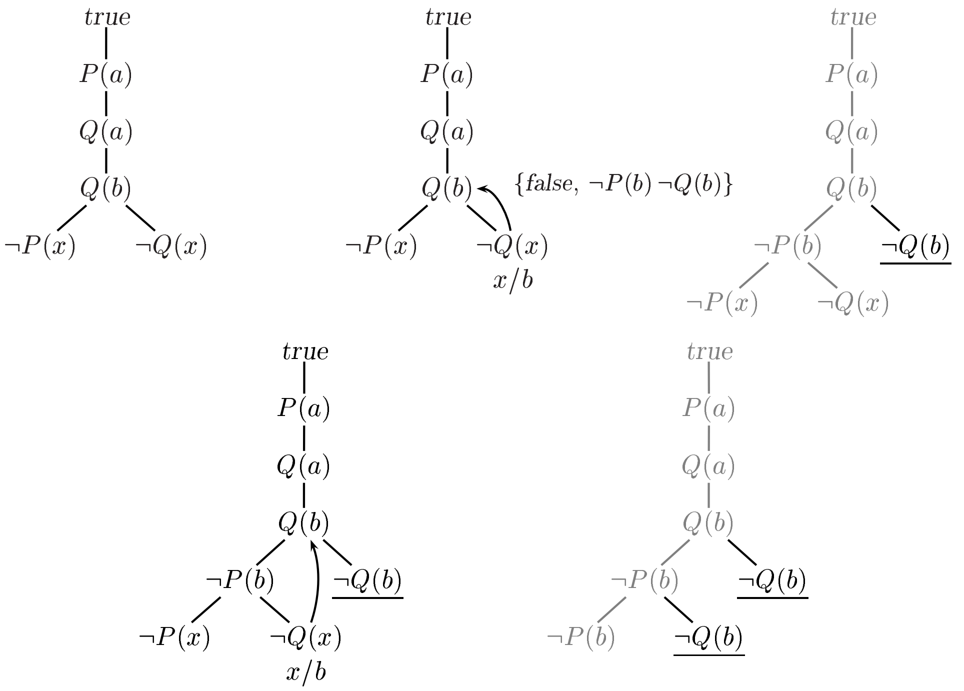


Figure 12: Illustration of Example 4.32.

4.32. EXAMPLE. Consider $S = \{P(a), Q(a), Q(b), \neg P(x) \vee \neg Q(x)\}$ and the well-order defined in the previous example. Starting with the initial tableau, the ground

[16] To be defined formally. For the time being work with set inclusion which suffices for the example.

clauses must be applied first, because closure is not possible and ground literals are minimal. No ground clause can be applied twice, because the resulting tableau would be subsumed. The top lefthand tableau in Figure 12 is obtained.

Assume *select branch* is implemented right-first. Now a second application of the non-ground clause competes with closure as indicated (the substitution that does not immediately lead to a proof is chosen deliberately). The literal set associated with the closure contains only ground literals and so is preferred over extension. Substitution $\{x/b\}$ affects the tableau below $Q(b)$ which is, therefore, replicated. The reconstructed tableau is on the right on the top row.

In the rightmost open branch, again, closure is preferred over extension. But if the same substitution as before is used, then the resulting tableau (righthand in bottom row) is subsumed by the top right tableau: the unchanged branch is trivially subsumed; *both* gray branches contain the same gray branch in the subsumed tableau.

This leaves closure with $\{x/a\}$ as the only legal rule application which results in a tableau proof immediately

4.8. Tableaux with Cuts and Lemmas

So far we discussed restrictions of tableau calculi aimed at diminishing the search space. Some problems, however, have extremely long tableau proofs. Already on the ground level there exist classes of formulas S_n such that the smallest tableau proof is exponential in the size of S_n whereas short resolution proofs exist [Cook and Reckhow 1979]. The reason is that resolution incorporates an atomic cut rule or (and this is just another name) lemma generation.

4.33. EXAMPLE. Let $\{P_1, \dots, P_n\}$ be different ground atoms. Consider the clause set

$$S_n = \{L_1 \vee \cdots \vee L_n \mid L_i \in \{P_i, \neg P_i\},\, i \in \{1, \dots, n\}\} \tag{4.8}$$

Obviously, S_n is unsatisfiable. D'Agostino [D'Agostino 1992] proved that the smallest closed clause tableau for S_n has at least $n!$ inner nodes, whereas S_n contains merely $n2^n$ literals. Even simple truth table checking has linear cost in the size of S_n.

Now consider the clause set $T_n = \{P_i \vee \neg P_i \mid i \in \{1, \dots, n\}\}$. $S_n \cup T_n$ has a short proof (displayed for $n = 3$ in Figure 13). The point is that the tautologies in T_n can be used to enumerate all interpretations over $\{P_1, \dots, P_n\}$. Then each clause in S_n is contradicted by exactly one interpretation. The resulting tableau has $n2^n + 2^{n+1} - 1 \in \mathcal{O}(|S_n \cup T_n|)$ nodes.

The effect of the clauses T_n can also be achieved by adding a new tableau extension rule

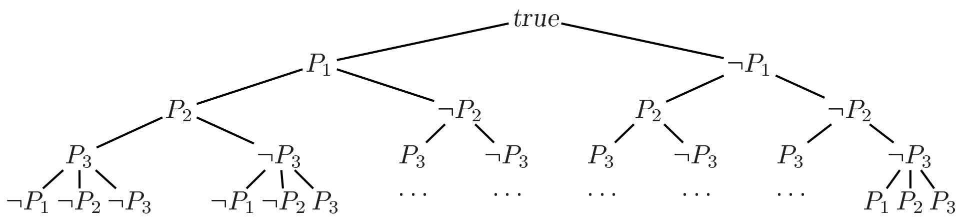


Figure 13: Clause tableau proof for $S_3 \cup T_3$.

$$\frac{}{P \;\mid\; \neg P} \tag{4.9}$$

called *atomic cut* rule. It is closely related the well-known cut rule part of sequent calculi [Gentzen 1935].

4.8.1. Tableaux and Sequent Calculi

A *propositional clausal sequent* is an expression of the form $\Gamma \Rightarrow \Delta$, where Γ is a tuple of clauses and Δ is a tuple of atoms. $\Gamma \Rightarrow \Delta$ is *valid* iff the formula $\bigwedge_{G \in \Gamma} G \to \bigvee_{D \in \Delta} D$ is valid. Hence, a clause set S is unsatisfiable if the sequent $S \Rightarrow$ is valid. The propositional clausal sequent calculus consists of only three rules:

$$\begin{gathered}\frac{\Gamma, L_1, \Gamma' \Rightarrow \Delta \;\mid\; \cdots \;\mid\; \Gamma, L_n, \Gamma' \Rightarrow \Delta}{\Gamma, L_1 \vee \cdots \vee L_n, \Gamma' \Rightarrow \Delta} \vee\text{-left} \\ \frac{\Gamma, \Gamma' \Rightarrow P, \Delta}{\Gamma, \neg P, \Gamma' \Rightarrow \Delta} \neg\text{-left} \qquad \frac{\times}{\Gamma, P, \Gamma' \Rightarrow \Delta, P, \Delta'} \text{axiom}\end{gathered} \tag{4.10}$$

It is straightforward to show that a sequent is valid iff it has a proof tree in which all leaves are marked with a $\times$. Moreover, there is a strong duality between clause tableaux and clausal sequent calculi, summarized in Table 3.[17]

While literal occurrences can be shared among several tableau branches, they are duplicated in sequents. In the light of this and the duality between sequent and tableau proofs, rule (4.9) is exactly the same as the usual cut rule of sequent calculi, where the *cut formula* ϕ is restricted to atomic formulas:

$$\frac{\Gamma, \Gamma' \Rightarrow \Delta, \phi, \Delta' \;\mid\; \Gamma, \phi, \Gamma' \Rightarrow \Delta, \Delta'}{\Gamma, \Gamma' \Rightarrow \Delta, \Delta'} \tag{4.11}$$

4.8.2. Tableaux with Lemmas

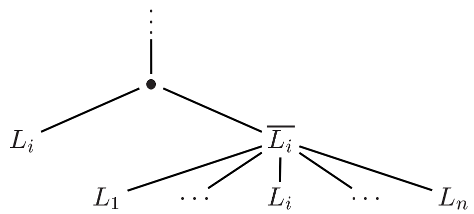
The main problem of the atomic cule (4.9) is that its application is completely unrestricted. A first restriction is obtained by allowing its application only if the

[17]This duality extends to the non-clausal and first-order case, see [Smullyan 1995].

Clausal Sequents	Clause Tableaux
Axiom rule	Closed branch
∨-left rule	Tableau extension
Atoms/unit clauses left of ⇒	positive branch literals
Atoms right of ⇒	negative branch literals
sequent proofs interpreted as logical conjunction of their end sequents	tableaux interpreted as logical disjunction of their branches
sequents interpreted as logical disjunction of their elements	branches interpreted as logical conjunction of their literals
Validity proof	Unsatisfiability proof

Table 3: Duality between sequent and tableau calculi.

following extension rule constitutes a strong connection step:



This way no more branches are generated if one had applied the extension step immediately. On the other hand, in $n-1$ of the new branches the literal $\overline{L_i}$ is present. Applying $n-1$ atomic cuts before an extension step and writing the resulting proof tree as a "macro rule" yields the *extension rule with local lemmas*:

$$\frac{}{\begin{array}{c|c|c|c|r} L_{\pi(1)} & \overline{L_{\pi(1)}} & \overline{L_{\pi(1)}} & \cdots & \overline{L_{\pi(1)}} \\ & L_{\pi(2)} & \overline{L_{\pi(2)}} & \cdots & \overline{L_{\pi(2)}} \\ & & L_{\pi(3)} & \cdots & \overline{L_{\pi(3)}} \\ & & & & \vdots \\ & & & \ddots & \overline{L_{\pi(n-1)}} \\ & & & & L_{\pi(n)} \end{array}} \qquad \begin{array}{l} L_1 \vee \cdots \vee L_n \in S \\ \pi \text{ permutation of } \{1, \ldots, n\} \end{array} \tag{4.12}$$

The complemented literals are called *local lemmas*: for example, $\overline{L_{\pi(i)}}$ is obtained as a lemma after the current branch B is closed with the help of $L_{\pi(i)}$. (Recall that closed branches are unsatisfiable, so this amounts to $B \models \overline{L_{\pi(i)}}$.) The lemma is *local* to certain branches as opposed to being global (on the whole tableau).

4.8.3. Tableaux with Folding Up
Depending on the sequence of branch closures, there are $n!$ different permutations of the local lemma rule applied to a n-literal clause. Not all of these are equally useful. To remedy this situation, a version of local lemma generation called *folding up rule* has been suggested. It can be seen as "lazy lemma generation".

4.34. EXAMPLE. Consider the clause set $S = \{P \vee T,\ P \vee \neg T,\ \neg P \vee Q \vee S,\ \neg Q \vee R,\ \neg R \vee \neg P,\ \neg S \vee R\}$ and the partial tableau proof for S displayed in Figure 14. After closure of branch $B = \{P, Q, R, \neg P\}$ one knows that $S \cup \{P, Q\} \models \neg R$. This lemma is useless, though, because branch $\{P, Q, \neg Q\}$ to the left is closed anyway. But inspection of the proof obtained so far shows that in fact $S \cup \{P\} \models \neg R$ holds, because Q was not used to close B.

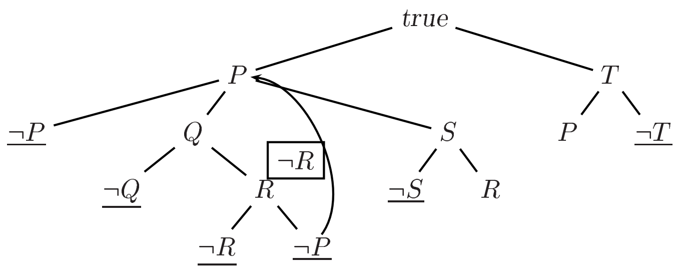


Figure 14: Illustration of Example 4.34.

In *clause tableaux with folding up rule* each local lemma may be moved up in the tableau until the lowest literal that was used in its proof [Letz 1993, Letz et al. 1994] and it can be used to close any branch below its new position.

With the folding up rule, in the previous example lemma $\neg R$ may be moved up to P and lemma $\neg P$ from the right subtree is moved up to *true* (the latter is also possible with a suitable variant of the local lemma rule). In Figure 15 new lemma positions are boxed, moves are indicated by dashed arrows, while additional closures through new lemmas are indicated by solid arrows.

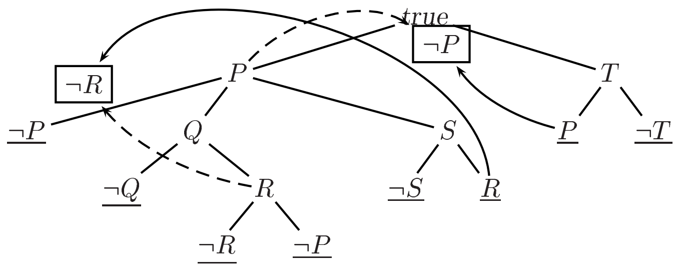


Figure 15: Tableau with folding up rule.

One can show that folding up does not change the *indeterministic power* of tableau with local lemmas that is the length of shortest proofs does not change. In a deterministic implementation folding up can nevertheless be very beneficial, because an ill choice of *select clause* can be remedied.

4.8.4. Tableaux with Factoring

Tableaux with factoring (also called *tableaux with merging*) are related to the subsumption rule in resolution theorem proving [Robinson 1965*b*]. Assume branch B is not yet closed, but there is a "more specific" branch B'. Formally, a branch B' is *more specific* than a branch B iff $B' \subseteq B\sigma$. Instead of trying to close B, one may simply refer to B' then, apply σ to B and consider it as closed provided that these references are acyclic. Soundness of tableaux with factoring is straightforward to prove; on the other hand it is sufficient to observe that tableaux with factoring can be simulated by tableaux with local lemmas: assume B was closed by referring to the more specific branch B' and that L is the top-most literal on B' not on B. Now replace the extension rule that produced L with a local lemma version putting $\overline{L}$ on B. As B' is more specific than B, there must be $L' \in B$ and substitution σ such that $L'\sigma = L$, so B can be closed with σ.

An improvement of tableaux with factoring called *tableaux with regressive merging* was introduced in [Wallace and Wrightson 1995] and is essentially the same as tableaux with the folding up rule; details can be found in [Wallace 1994].

4.8.5. Problems of Strengthening Tableaux

Strengthening of tableau procedures at first seems a sure win, because length of proofs can be drastically reduced. On the propositional level this holds without reservation and it can safely be claimed that any competitive propositional proof procedure embodies some variant of cut. On the first-order level, however, additional literals introduced as cuts or lemmas create additional possibilities for branch closure. The negative effects from this increase of the search space can easily outweigh the possibility of finding shorter proofs.

4.35. EXAMPLE. $S = \{\neg P(x) \vee Q(x) \vee R(x),\ P(a),\ \neg Q(a),\ P(b),\ \neg Q(b),\ \neg R(b)\}$. In the partial connection tableau in Figure 16 the left branch was closed first by extension with $P(a)$ (where only $P(b)$ leads to success). This forces extension with $Q(a)$ in the middle branch and generation of some lemmas (framed literals). The lemmas allow to extend the right branch with another instance of the first clause which would have been impossible without them. Detection of the wrong first extension is thus possibly delayed a long time.

In the example, a regularity check would help ($R(a)$ becomes irregular on the rightmost branch) and also restriction of lemma usage to reduction steps. Although this helps somewhat, more complex examples create the same problems as before.

On the other hand, local lemmas are not strong enough on the first-order level. In the clause tableau in Figure 6, for example, the lemma $\neg Q(c)$ can be folded up to the *true* node, but is useless to close the open branch on the right, because a

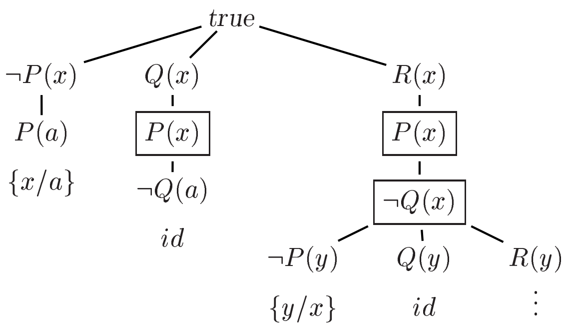


Figure 16: Search space increase caused by local lemmas.

different instance is required. But in fact it is justified to derive even $(\forall x)\neg Q(x)$ as a lemma. This is always possible when the proof of (that is: the tableau below) the lemma does not instantiate any variables that occur outside of it. It remains to be seen whether such an optimization can be efficiently implemented and does not blow up the search space beyond any usefulness.

5. Tableaux as a Framework

In this section it is argued that many well-known calculi can be uniformly presented within a tableau framework and that it is beneficial to do this. This was demonstrated already in Section 4.6 with model generation. The relationship between tableaux and the connection method [Bibel 1982*b*] was discussed earlier in Section 3.5 for the general, non-clausal case.

5.1. Model Elimination

Model elimination was suggested by Loveland [1968*b*, 1969] as a space efficient proof search procedure with small local search space. The following definitionis taken from his [1969] article and was slightly simplified.

5.1. Definition. A *chain* is a finite sequence of literals $\langle L_1, \ldots, L_n \rangle$ each of which has either *type* A or *type* B. (Type A literals will be framed for easy distinction.)

Define $\langle L_1, \ldots, L_n \rangle\sigma = \langle L_1\sigma, \ldots, L_n\sigma \rangle$, then construction rules for *admissible chains* for a given set of clauses S are as follows:

Init: If $L_1 \vee \cdots \vee L_n \in S$ then the chain $\langle L_1, \ldots, L_n \rangle$ consisting of type B-literals admissible.

Extend: If $\langle L_1, \ldots, L_n \rangle$ is an admissible chain, L_n of type B, $C = M_1 \vee \cdots \vee M_m$ an instance of a clause in S, and there is an MGU σ such that $\overline{L_n}\sigma \in C\sigma$, then $\langle L_1, \ldots, L_{n-1}, \boxed{L_n}, M_1, \ldots, M_{j-1}, M_{j+1}, \ldots, M_m \rangle\sigma$ is admissible, where the M_i are of type B.

Reduce: If $\langle L_1, \ldots, L_n \rangle$ is an admissible chain, $\boxed{L_i}$ of type A, $L_j, \ldots, L_n$ of type B $(i < j)$, and there is an MGU σ of $\{\overline{L_i}, L_j\}$ then $\langle L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_n \rangle\sigma$ is admissible.

Contract: If $\langle L_1, \ldots, \boxed{L_n} \rangle$ is an admissibe chain and $\boxed{L_j}, \ldots, \boxed{L_n}$ are of type A, then $\langle L_1, \ldots, L_{j-1} \rangle$ is admissible.

Type A literals can be memoized as *ancestor literals.*

5.2. DEFINITION. A **model elimination** proof of a clause set S is a sequence of admissible chains for S ending with the empty chain $\langle\rangle$.

5.3. THEOREM ([Loveland 1969]). *A clause set S is unsatisfiable iff it has a model elimination proof.*

5.4. EXAMPLE. Consider $S = \{P,\ R \vee \neg P \vee Q,\ S \vee \neg Q,\ \neg Q \vee \neg S,\ \neg Q \vee \neg R\ \neg R \vee Q\}$. A model elimination proof of S is as follows:

0.	P	Init				
1.	$\boxed{P}$	R	Q	Extend		
2.	$\boxed{P}$	R	$\boxed{Q}$	S	Extend	
3.	$\boxed{P}$	R	$\boxed{Q}$	$\boxed{S}$	$\neg Q$	Extend
4.	$\boxed{P}$	R	$\boxed{Q}$	$\boxed{S}$	Reduce	
5.	$\boxed{P}$	R	Contract			
6.	$\boxed{P}$	$\boxed{R}$	Q	Extend		
7.	$\boxed{P}$	$\boxed{R}$	$\boxed{Q}$	$\neg R$	Extend	
8.	$\boxed{P}$	$\boxed{R}$	$\boxed{Q}$	Reduce		
9.	Contract					

For sake of readability delimiters of chains are not displayed, lines are numbered starting with 0. The initial step can be with any clause from S: take the first. Now P is the single type B literal. Neither reduction nor contraction is allowed. The only possibility to extend is with the second clause, because a clause with $\neg P$ is required. Type B literal P gets type A and $\neg P$ is deleted.

Again, only extension is possible and a clause with $\neg Q$ in it is required of which the first is chosen. Then extend with $\neg S \vee \neg Q$, the result is line 3.

Finally, a reduce step is possible with $\boxed{Q}$; type B literal $\neg Q$ is deleted. The rightmost bloc of type A literals is removed with the contract rule.

Two extension steps with $\neg R \vee Q$ and $\neg Q \vee \neg R$ are followed by reduction with $\boxed{R}$, whereby $\neg R$ is deleted. The empty chain is obtained by contraction of all remaining (type A) literals.

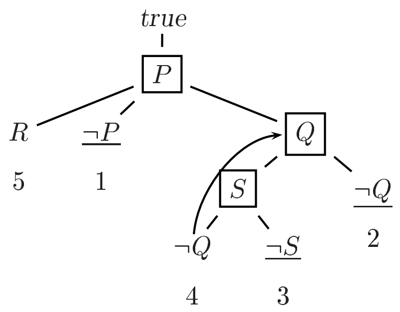


Figure 17: Partial regular strong connection tableau corresponding to model elimination proof in Example 5.4.

Model elimination is closely linked to strong connection tableaux. This is demonstrated by the tableau in Figure 17 which simulates lines 0 to 5 in the model elimination proof above. The numbers below each branch indicates which line in the model elimination proof corresponds to its closure.

Type A literals become inner tableau nodes. The type B literals between type A literals in chains are leaves of branches yet to be closed. Obviously, the initialization, the extension, and the reduction rule correspond to tableau rules with the same name. The contract rule cleans up recently closed branches. Hence, a chain is a simply a linear format to represent open branches of a strong connection tableau. This is only possible, because *select branch* chooses always the rightmost branch for extension.

In the paper [Loveland 1969, p. 351] chains are restricted in a way that corresponds to regularity in tableaux. Two different completion modes of chains for a given clause set are specified which vary the number of extensions made to produce a chain. A lemma generation mechanism that corresponds to local lemmas in tableaux is described as well.

Like matrices [Andrews 1981, Bibel 1981], model elimination chains are a much more implementation-oriented format than tableau trees. This makes it harder to see further optimizations (such as more liberal branch selection rules or folding up) and to prove completeness. The latter follows directly from completeness of regular strong connection tableaux, of course.

Although model elimination can be interpreted as a systematic procedure to exclude certain interpretation as possible models of clause set [Loveland 1969, p. 362] one should stress that the type A literals in a non-empty chain to which no rule can be applied in general do *not* constitute a partial model, because model elimination, just like strong connection tableaux, is not proof confluent.

To summarize, model elimination (without the lemma generation mechanism) is a notational variant of regular strong connection tableaux.

5.2. Linear Resolution

Linear resolution [Loveland 1968*a*, Luckham 1968] (see also Chapter [Chapter `resolution`]) and its refinements do not fit directly into the framework of standard resolution [Robinson 1965*b*], because their definition like tableau calculi with connection condition relies on the form of the derivation. Indeed, as Loveland [1972] observed, SL-resolution [Kowalski and Kuehner 1971, Reiter 1971] (formally defined in Chapter [Chapter `resolution`], Section 9.3) is almost identical to model elimination plus the factoring rule (as defined in Section 4.8.4) and, hence, to regular connection tableaux with factoring.

5.3. Tableaux and (Disjunctive) Logic Programming

In order to benefit from this section, a little background in logic programming is of advantage, which can be gained from [Lloyd 1987].

Historically, one root of logic programming [Kowalski 1974] is SL-resolution [Kowalski and Kuehner 1971] and, hence, model elimination. It is not surprising, therefore, that logic programming and many of its extensions have natural interpretations as variants of strong connection tableaux.

More formally, consider Theorem 4.15 for the special case when S is a set of Horn clauses and without regularity enforced. Any unsatisfiable set of Horn clauses not containing the empty clause must contain a relevant negative clause (query), because Horn clause sets without query clauses are satisfiable by the interpretation that makes all positive literals true. Let Q be a relevant query of S. Without loss of generality, assume Q is not a unary clause (otherwise, replace Q with $Q \vee \neg P$, where P does not occur in S, and add the fact P).

The proof of Theorem 4.15 guarantees existence of a strong connection tableau proof T for S starting with Q. By a trivial induction on the depth of T using that S is Horn and strong connectedness of T one obtains that the positive literals of T are exactly the literals at leaf nodes. Consequentially, no reduction steps occur in T and the connection literal of clauses used in extension steps is always the positive literal of a rule or a fact in S.

It was noted on p. 25 that the existence of T is independent of the *select branch* strategy, so we may assume that to be left-first. In this case, T is a notational variant of a *Prolog-SLD refutation* of $\{Q\} \cup (S - \{Q\})$ in the usual sense [Lloyd 1987]. This remains true on the first-order level.

5.5. Example. To better approximate the usual notation of logic programming, we employ rule notation of clauses as introduced in Section 4.6.[18]

[18] Without loss of generality, clause sets are assumed to be in *goal normal form* in this section, that is, they contain at most one query clause. When there is more than one query clause, this can easily be achieved by adding a new head predicate P to each query as well as the new query $P \rightarrow$ *false*.

$$S = \{C_1(x) = Q(x) \wedge R \rightarrow P(x),\ \mathit{true} \rightarrow P(d),\ C_2(x) = S(x) \wedge T(x) \rightarrow Q(x),$$
$$\mathit{true} \rightarrow Q(a),\ \mathit{true} \rightarrow S(b),\ \mathit{true} \rightarrow S(c),\ \mathit{true} \rightarrow T(c),\ \mathit{true} \rightarrow R,$$
$$P(x) \rightarrow \mathit{false}\}$$

One of several possible strong connection tableaux for S starting with the only query $P(x) \rightarrow \mathit{false}$ in S is displayed on the left in Figure 18.

The open branches are the ones ending with $\neg T(z)\{z/c\} = \neg T(c)$ and $\neg R$, respectively. The tableau contains a lot of redundant information such as branches already closed and non-leaf nodes (which are never used, because only strongly connected extension steps are made). Thus a less redundant notation for strongly connected Horn clause tableau merely records (a) the sequence of leaves of open branches and (b) the clause instance and MGU leading to the successor tableau. The result of this simplification is displayed above on the right: each node represents a strong connection tableau constructed from its parent node. The tableau on the left corresponds to the bottom-most node in black. The grey nodes complete the proof.

At the same time, the structure on the right constitutes a Prolog-SLD refutation of S from $P(x) \rightarrow \mathit{false}$ in common notation [Lloyd 1987]. Another valid view of it is, of course, as a model elimination proof, where only the rightmost bloc of literals in an ME chain is kept.

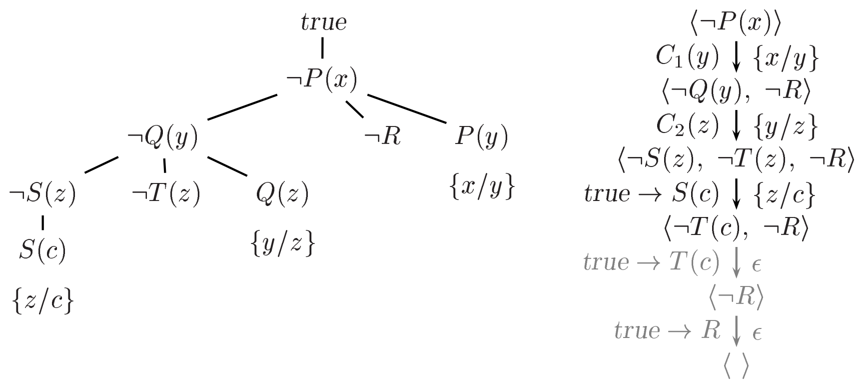


Figure 18: Illustration of Example 5.5.

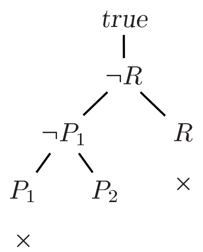
We stress that although the Prolog procedure is often presented as a resolution refinement it can be completely and naturally fitted into the strong connection tableau (and hence: model elimination) framework. In the remainder of this section I intend to show that it is even advantageous to do so.

Disjunctive logic programming, instances of which are *Non-Horn clause* [Plaisted 1988] and *Near-Horn logic programming* [Loveland 1987], is the attempt to lift the Horn restriction from Prolog while retaining the efficient implementability of Prolog-SLD resolution. One line of development was to implement model elimination (that is: regular strong connection tableaux) for arbitrary clauses directly via an extended Prolog Abstract Machine [Letz et al. 1992]. This is discussed in detail in Chapter [Chapter `me`]. In a second group of papers one can find several suggestions how to extend Prolog-SLD resolution such that general clauses can be handled [Plaisted 1982, Gabbay and Reyle 1984, Plaisted 1988, Loveland 1991, Reed and Loveland 1992*a*, Baumgartner and Furbach 1994, Baumgartner and Furbach 1997, Baumgartner and Furbach 1998]. These calculi are in some aspects more restrictive in others more general than strong connection tableaux and model elimination.[19] We start our discussion by two examples that highlight some differences.

5.6. EXAMPLE. Consider $S_1 = \{\mathit{true} \to P \lor Q,\ P \land Q \to \mathit{false},\ P \to Q,\ Q \to P\}$. In the Prolog-SLD restriction of strong connection tableau no reduction steps are allowed. Obviously, no strong connection tableau proof without reduction steps can exist for S_1: as there are no unit clauses, no extension step diminishes the number of open branches.

The example shows that reduction steps are necessary to handle general clause sets. The second examples concerns positive leaf literals of tableau proofs. Recall that these cannot occur in the Horn case when one starts the proof with a query.

5.7. EXAMPLE. Consider $S_2 = \{P_1 \to R,\ P_2 \to R,\ R \to \mathit{false},\ \mathit{true} \to P_1 \lor P_2\}$. Below is a partial strong connection tableau proof for S_2:



In strong connection tableaux the open branch ending in P_2 can be extended with $P_2 \to \mathit{false}$ which completes the proof (the case when $P_2 \to R$ is used first, leads to a symmetric situation). To do so, however, is a violation of a main principle of Prolog-interpretation, namely, that the current subgoal (the leaf) may only be unified with a *head* literal of a rule. Indeed, in the linear format for Prolog-SLD resolution discussed in Example 5.5, the literal P_2 does not even have a slot. It is, of course, possible to extend the linear proof format to accomodate head literals [Reed and Loveland 1992*a*], but I prefer to use the clause tableau format, where all required information is present and subgoal dependencies are easy to spot. Some authors also like to use a sequent-style format [Plaisted 1990, Reed and Loveland 1992*b*].

[19]The proximity of near-Horn Prolog to model elimination was pointed out by Reed and Loveland [1992*b*] and again by Baumgartner and Furbach [1994].

The principle of Prolog execution, not to extend via body literals, discussed in the previous example is often referred to as “no use of *contrapositives*”. The set of contrapositives of a clause $L_1 \vee \cdots \vee L_n$ is the rule set

$$\{L_1 \vee \cdots L_{i-1} \vee L_{i+1} \vee \cdots \vee L_n \rightarrow L_i \mid 1 \leq i \leq n\} \enspace .$$

Apart from implementational issues it is obvious that the possibility of extension steps with arbitrary contrapositives such as in connection tableaux and model elimination increases the local search space. Thus the interest in calculi that are complete without contrapositives.

How does one render strong connection tableaux without contrapositives complete? The situation is quite similar to tableaux with selection function discussed in Section 4.5.2, because the head literals can be seen as the selected literals of a clause. It does not come as a surprise then, that the remedy is similar: certain *restart steps* that is extension steps without strong connection are permitted.

The coordinates of clause tableau calculi for disjunctive logic programming are thus given as follows:

- strongly connected extensions steps, respectively, the Prolog-SLD resolution rule
- usable contrapositives of clauses are explicitly given, by default, as a set of clauses in rule notation in which exactly the positive literals are head literals
- reduction steps (called *ancestor cancellation* rule by Reed and Loveland [1992*b*] who derive the name from ancestor resolution), which are only allowed from literals that belonged to bodies of clauses
- restart extension steps, which are only allowed below literals stemming from heads of clauses

The last two items constitute the parameterizable part. The presence of the first, second and fourth feature imply that a calculus is not proof confluent as the example $S = \{\mathit{true} \rightarrow C,\, A \rightarrow C,\, B \rightarrow A,\, A \rightarrow \mathit{false}\}$ shows: Any tableau starting with the last clauses, followed by a strong extension step with the second nut last clause, cannot be extended, although S is clearly unsatisfiable. Hence, all calculi described in the remainder of this section are not proof confluent, unless otherwise noted.

All deduction procedures incorporating at least the first three ingredients are said to belong to the *ancestry family* of deductive procedures by Reed and Loveland [1992*b*]. In the remainder of this section I define some of its important representatives within the tableau framework.

5.3.1. Near-Horn Prolog, Simplified Problem Reduction Format

Near-Horn Prolog, later called *Unit near-Horn Prolog* (UnH-Prolog), was suggested in [Loveland 1987, Loveland 1991], a variant called *Inheritance near-Horn Prolog* (InH-Prolog) is due to [Loveland and Reed 1991]. The most detailed reference on near-Horn Prolog and related procedures is [Reed and Loveland 1992*b*].

In UnH-Prolog restart steps are allowed with query clauses and with any clause containing a head literal that is weakly connected to the current branch (when head literals are considered as selected literals this is exactly a weakly connected extension step of tableaux with selection function in Section 4.5.2). In UnH-Prolog restarts can only occur below head literals and reduction steps are only allowed to the head literal above the most recent restart.

In InH-Prolog, on the other hand, restart steps are only permitted with query clauses, but reduction steps can be to the head literal above *any* restart step on the current branch.

Both, UnH- and InH-Prolog, use a left-first branch selection rule and both feature the so-called *cancellation pruning rule* which excludes certain redundant proofs: each head literal above a restart step must be used in a reduction step with no restart step in between.

5.8. Example. The tableau in the previous example is both an UnH- and an InH-Prolog derivation. In UnH-Prolog the open branch can be extended by a restart step with $P_2 \to R$, and the proof is immediately finished by a reduction step to P_2. In InH-Prolog only a restart step with $R \to \mathit{false}$ is possible, leading to a slightly longer proof.

It was observed by Baumgartner and Furbach [1994] that completeness of InH-Prolog entails completeness of weak connection tableaux (Section 4.3.2) and thus the clausal version of the connection method (Section 3.5) even without contrapositives provided that the input is in goal normal form. This comes from the simple fact that the query clause of a restart step becomes weakly connected to the top literal when each clause in an InH-Prolog proof is replaced by its goal normal form.

In Plaisted's [1982] *Simplified Problem Reduction Format* (SPRF) reduction steps are as in InH-Prolog while restart steps may occur anytime with non-Horn clauses. Branch selection is flexible. In SPRF-D (for SPRF with delay) [Plaisted 1988] restart steps are restricted exactly as in InH-Prolog, but need not occur below a head literal. Up to this feature and the cancellation pruning rule which is not present in SPRF-D both calculi are identical [Reed and Loveland 1992*a*].

Another deduction system related to InH-Prolog is N-Prolog [Gabbay and Reyle 1984, Gabbay 1985]. As N-Prolog is not clause-based and has an intuitionistic semantics, a comparison hinges on the translation chosen for clause representation. Reed and Loveland [1992*a*] showed that it is possible to embed both InH-Prolog and SPRF into N-Prolog.

SLWV-resolution [Pereira et al. 1992], often discussed in connection with near-Horn Prolog [Reed and Loveland 1992*b*, Baumgartner and Furbach 1994], was mentioned in Section 4.5.3, because it is actually an instance of tableaux with selection function.

5.3.2. Restart Model Elimination

While near-Horn Prolog and its relatives were developed "bottom-up" as extensions of Horn clause logic programing, the motivation of *restart model elimination* for Baumgartner and Furbach [1994] was to obtain a version of strong connection tableaux/model elimination being complete without contrapositives. Even if restart steps need to be added for this purpose, it can be seen as a "top-down" approach.

The base calculus of restart model elimination has exactly the same restart rule as InH-Prolog: only query clauses (in goal normal form: *the* query clause) can be used as a restart below head literals. Reduction is completely unrestricted, hence InH-Prolog is a refinement of restart model elimination.

The restriction of reduction steps starting from body literals in InH-Prolog is called *strict restart model elimination* in [Baumgartner and Furbach 1994]. This leaves the cancellation pruning rule and left-first strategy as the only difference. The former is considered in [Baumgartner and Furbach 1998], the latter may be safely assumed in PTTP-technology implementations such as the one reported in [Baumgartner and Furbach 1994].

Restart model elimination has a number of refinements [Baumgartner and Furbach 1994, Baumgartner and Furbach 1998] which justify the investigation of this procedure in its own right:

Head Selection Function: similar as in Section 4.5.2 a selection function is used to choose one head literal to be used in an extension step.

Blockwise Regularity: several restarts with the same literal must be admitted to maintain completeness, but between subsequent restarts regularity can be enforced.

Independence of Goal Clause: in goal normal form, there is exactly one query clause, but in the second extension step any of the query clauses that existed before goal normal form can be used. Independence of the goal clause means that completeness is independent of the choice of this latter clause.

Not all refinements are compatible with each other, certain combinations result in an incomplete deduction system. Details are given in [Baumgartner and Furbach 1998].

An important issue when using Disjunctive Logic Programming as a programming language is the computation of all correct answer substitutions (closing substitutions for the tableau proof restricted to variables occurring in the query). Baumgartner, Furbach and Stolzenburg [1997] prove answer completeness of restart model elimination.

Many aspects discussed in the present section are covered in greater detail in [Reed and Loveland 1992*b*, Baumgartner and Furbach 1998] than it is possible to do here.

5.3.3. Restart Tableaux

There are calculi that combine features from tableaux with selection function and near-Horn Prolog/restart model elimination.

Restart tableaux [Pape and Hähnle 1997] can be viewed as a one possible way to render the latter complete: recall from Section 4.5.3 that combining selection functions and the weak connection condition for every extension step results in an incomplete calculus. Adding restart steps in the sense of Section 4.5.2 gives completeness. Restart tableaux restrict these latter restart steps to occur only when no connected extension step is possible, but, in contrast to near-Horn Prolog and restart model eliminiation, they may occur below body literals as well. This feature renders restart tableaux *proof confluent*.

Restart tableaux admit to select an arbitray subset S_s of the input clause set S such that only strongly connected extension steps can be performed with clauses from S_s. Depending on the degree of "strong connectedness", a more restrictive notion of regularity than in restart model elimination may be enforced. Thus restart tableaux combine features of restart model elimination and tableaux with selection function.

There is a non-proof confluent version of restart tableaux, called *strict restart tableaux* [Pape and Hähnle 1997], which does not allow restart steps below body literals, but has a more liberal regularity concept. It can be instantiated to strict restart model elimination.

5.4. Davis-Putnam Procedure and KE method

Tableaux with cut rule (4.9) are a redundant calculus in the sense that the cut rule is not required for completeness. It was noted in Section 4.8.5 that this can blow up the search space considerably. Mondadori [1988] designed a tableau system with cut called *KE* for full first-order logic in which every rule, including the cut rule, is needed for completeness. Together with D'Agostino, it was further developed and extended to non-classical logics in a series of papers [Mondadori 1989*a*, Mondadori 1989*b*, D'Agostino 1990, D'Agostino and Mondadori 1994].

KE in the clausal case can succinctly described as follows: consider weak connection tableaux (Section 4.3.2) with atomic cut, where the extension rule (ii$''$) on page 27 is replaced with:

(ii$''''$) Let T be a tableau for S, B a branch of T containing L, $L_1 \vee \cdots \vee L_r$ *is on* B *or* an instance of $C \in S$. If $\overline{L}$, L_i (where $i \in \{1, \ldots, r\}$) are unifiable with MGU σ and the tree T' is constructed by extending B with *two* new subtrees, where the nodes of the new subtrees are the L_i and $L_1 \vee \cdots \vee L_{i-1} \vee L_{i+1} \vee \cdots \vee L_r$, then $T'\sigma$ is a clause tableau for S, in which the branch ending with $L_i\sigma$ is marked as closed. (This is called a *KE extension step*.)

Formally, the resulting calculus is not a variant of clause tableaux, but of non-clausal tableaux, because nodes may be clause instances. The extension rule (ii$''''$) does not alter the number of open branches, so the system is incomplete without atomic cut which is called *principle of bivalence* by Mondadori [1988].

Rule (ii$''''$) is better known under the name *unit resolution*. Its ground version originates from [Davis and Putnam 1960], where it was called *elimination of one-literal clauses*. The *affirmative-negative* rule in [Davis and Putnam 1960], better known as *pure clause rule* is realized by the weak connection condition. Finally, atomic cut is a notational variant of the *splitting rule* in [Davis et al. 1962].

To summarize, in the ground clause case, Mondadori's [1988] KE system and the *Davis-Putnam-Loveland procedure* [Davis et al. 1962] are the same and an instance of weakly connected tableaux with atomic cut. The idea is, of course, to apply the splitting/cut rule only when no other rule is applicable. This heuristic is very useful for propositional logic, but it makes the Davis-Putnam-Loveland procedure problematic to lift, because splitting may be delayed indefinitely.

It is justified to view the full KE system as a non-clausal version of the Davis-Putnam-Loveland procedure. In a tableau framework with *analytic cut* (the cut formula is any subformula of the input), it is characterized by unchanged non-branching (type α, γ, δ) rules, while type β rules are used in the following version:

$$\frac{\begin{array}{c}\beta\\ \beta_i\end{array}}{\beta_1 \vee \cdots \vee \beta_{i-1} \vee \beta_{i+1} \vee \cdots \vee \beta_n} \tag{5.1}$$

Mondadori [1989*b*] also suggest a system called *KI* which uses the following, inverted, rules at the propositional level:

$$\frac{\beta_i}{\beta} \qquad\qquad \frac{\begin{array}{c}\alpha_1\\ \vdots\\ \alpha_n\end{array}}{\alpha} \tag{5.2}$$

The rules are analytic in the sense that the conclusion must occur as a subformula in the input. They may either be used in addition to the KE rules (in which case they can be seen as an approximation of Massacci's [1998*b*] simplification rule, see Section 3.5.3), alternatively, they form yet another complete system together with *atomic cut*.

6. Historical Remarks & Resources

I can only give a very rough sketch here. More detailed accounts of the history of tableau-like theorem proving can be found in [Bibel and Schmitt 1998, pp. 4–7] and [Fitting 1999].

Despite their recent flourishing, the history of tableau methods is way older than that of resolution. They can be traced back to the cut-free version of Gentzen's [1935] sequent calculus. Hintikka [1955] and Beth [1955] abstracted from structural rules in sequent calculi (essentially treating sequents as sets of formulas), improved the proof representation, and introduced signed formulas. They also stressed the semantic view of tableaux as a procedure that tries systematically to find a counter example for a given formula (a model in which its negation is true) as opposed to Gentzen's purely proof theoretical motivation.[20] Further improvements were made by Schütte [1960]; Smullyan's [1995] elegant formulation, employing unifying notation which greatly simplified matters, became very popular while similar contributions by Lis [1960] unfortunately went unnoticed.

Many important improvements directed to automated proof search in sequent/tableau/model elimination calculi were made quite early: free variables

[20]The proof theoretic tradition is very much alive today as witnessed, for example, by Girard's [1987] work, but it is not really relevant for automated deduction.

[Kanger 1957, Prawitz 1960], unification [Loveland 1969, Bibel 1982*b*, Andrews 1981], proof representation and connection refinements [Davydov 1973, Bibel and Schreiber 1975, Andrews 1976].

The relative success of resolution-based theorem proving, however, eclipsed this progress and serious implementations of tableau-like calculi are spurious before the late 1980s [Brown 1978, Oppacher and Suen 1986].

In the last decade tableau-like calculi became focal points of research again, spurred by the success of efficient implementations and the demand for a computational treatment of non-classical logics. The tableau community gathers in an international conference,[21] where many of the relevant results are now published. Part I of [Bibel and Schmitt 1998] contains some survey articles on various aspects of state-fo-the-art research on tableau merthods. The Handbook of Tableau Methods [D'Agostino, Gabbay, Hähnle and Posegga 1999] contains extensive and fairly up-to-date information, not limited to automated reasoning.

Acknowledgments

Thanks are due to Bernhard Beckert and Ryuzo Hasegawa for discussions and helpful observations. Reinhold Letz contributed Example 4.10.

Bibliography

ANDREWS P. B. [1976], 'Refutations by matings', *IEEE Transactions on Computers* **C-25**(8), 801–807.

ANDREWS P. B. [1981], 'Theorem proving through general matings', *JACM* **28**, 193–214.

BAUMGARTNER P. [1998], Hyper Tableaux — The Next Generation, *in* H. de Swart, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands', number 1397 *in* 'LNCS', Springer-Verlag, pp. 60–76.

BAUMGARTNER P., EISINGER N. AND FURBACH U. [1999, to appear], A confluent connection calculus, *in* S. Hölldobler, ed., 'Intellectics and Computational Logic — Papers in Honor of Wolfgang Bibel', Kluwer.

BAUMGARTNER P. AND FURBACH U. [1994], 'Model Elimination without Contrapositives and its Application to PTTP', *Journal of Automated Reasoning* **13**, 339–359.

BAUMGARTNER P. AND FURBACH U. [1997], Refinements for Restart Model Elimination, *in* 'Proc. International Workshop on First Order Theorem Proving', Technical Report, RISC-Linz. *http://www.uni-koblenz.de/ peter/Publications/ftp97-final.ps.gz

BAUMGARTNER P. AND FURBACH U. [1998], Variants of clausal tableaux, *in* W. Bibel and P. Schmitt, eds, 'Automated Deduction: A Basis for Applications', Vol. I, Kluwer, chapter 3, pp. 73–101.

BAUMGARTNER P., FURBACH U. AND NIEMELÄ I. [1996], Hyper tableaux, *in* J. J. Alferes, L. M. Pereira and E. Orłowska, eds, 'Proc. European Workshop: Logics in Artificial Intelligence, JELIA', Vol. 1126 of *LNCS*, Springer-Verlag, pp. 1–17.

BAUMGARTNER P., FURBACH U. AND STOLZENBURG F. [1997], 'Computing answers with model elimination', *Artificial Intelligence Journal* **90**(1–2), 135–176. *http://www.uni-koblenz.de/ peter/Publications/ARME-AIJ.ps.gz

[21] `http://www.cs.albany.edu/ nvm/tab99/`

Beckert B. [1998], Integrating and Unifying Methods of Tableau-based Theorem Proving, PhD thesis, University of Karlsruhe, Department of Computer Science.

Beckert B. and Goré R. [1997], Free variable tableaux for propositional modal logics, *in* 'Proc. International Conference on Theorem Proving with Analytic Tableaux and Related Methods, Pont-a-Mousson, France', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 91–106.

Beckert B. and Hähnle R. [1998], Analytic tableaux, *in* W. Bibel and P. Schmitt, eds, 'Automated Deduction: A Basis for Applications', Vol. I, Kluwer, chapter 1, pp. 11–41.

Beckert B. and Posegga J. [1995], 'leanTAP: Lean tableau-based deduction', *Journal of Automated Reasoning* **15**(3), 339–358.

Berka, K. and Kreiser, L., eds [1986], *Logik-Texte. Kommentierte Auswahl zur Geschichte der modernen Logik*, Akademie-Verlag, Berlin.

Beth E. W. [1955], 'Semantic entailment and formal derivability', *Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde, N.R.* **18**(13), 309–342. Partially reprinted in [Berka and Kreiser 1986].

Bibel W. [1981], 'On matrices with connections', *JACM* **28**, 633–645.

Bibel W. [1982*a*], *Automated Theorem Proving*, Vieweg, Braunschweig.

Bibel W. [1982*b*], 'A comparative study of several proof procedures', *Artificial Intelligence* **18**, 269–293.

Bibel W. [1987], *Automated Theorem Proving*, second revised edn, Vieweg, Braunschweig.

Bibel W., Brüning S., Egly U., Korn D. and Rath T. [1995], Issues in theorem proving based on the connection method, *in* P. Baumgartner, R. Hähnle and J. Posegga, eds, 'Proceedings of the 4th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods', Vol. 918 of *LNCS*, Springer-Verlag, pp. 1–16.

Bibel, W. and Schmitt, P., eds [1998], *Automated Deduction: A Basis for Applications*, Kluwer.

Bibel W. and Schreiber J. [1975], Proof search in a Gentzen-like system of first-order logic, *in* F. Gelenbe and F. Poitier, eds, 'International Computing Symposium', North-Holland, pp. 205–212.

Brown F. M. [1978], 'Towards the automation of set theory and its logic', *Artificial Intelligence* **10**(3), 281–316.

Bry F. and Yahya A. [1996], Minimal model generation with positive unit hyper-resolution tableaux, *in* P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi, eds, 'Theorem Proving with Tableaux and Related Methods, 5th International Workshop, TABLEAUX'96, Terrasini, Palermo, Italy', Vol. 1071 of *LNCS*, Springer-Verlag, pp. 143–159.

Bryant R. E. [1986], 'Graph-based algorithms for Boolean function manipulation', *IEEE Transactions on Computers* **C-35**, 677–691.

Cook S. and Reckhow R. [1979], 'The relative efficiency of propositional proof systems', *Journal of Symbolic Logic* **44**, 36ff.

D'Agostino M. [1990], Investigations into the Complexity of some Propositional Calculi, PhD thesis, Oxford University Computing Laboratory, Programming Research Group. Also Technical Monograph PRG–88, Oxford University Computing Laboratory.

D'Agostino M. [1992], 'Are tableaux an improvement on truth tables? Cut-free proofs and bivalence', *Journal of Logic, Language and Information* **1**, 235–252.

D'Agostino, M., Gabbay, D., Hähnle, R. and Posegga, J., eds [1999], *Handbook of Tableau Methods*, Kluwer, Dordrecht.

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

Davis M., Logemann G. and Loveland D. [1962], 'A machine program for theorem-proving', *Communications of the ACM* **5**, 394–397.

Davis M. and Putnam H. [1960], 'A computing procedure for quantification theory', *JACM* **7**(3), 201–215.

DAVYDOV G. V. [1973], 'Synthesis of the resolution method with the inverse method', *Journal of Soviet Mathematics* **1**, 12–18. Translated from Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Mathematicheskogo Instituta im. V. A. Steklova Akademii Nauk SSSR, vol. 20, pp. 24–35, 1971.

FITTING M. [1999], Introduction, *in* M. D'Agostino, D. Gabbay, R. Hähnle and J. Posegga, eds, 'Handbook of Tableau Methods', Kluwer, Dordrecht.

FITTING M. C. [1996], *First-Order Logic and Automated Theorem Proving*, second edn, Springer-Verlag, New York.

FUJITA H. AND HASEGAWA R. [1991], A model generation theorem prover in KL1 using a ramified-stack algorithm, *in* K. Furukawa, ed., 'Proceedings 8th International Conference on Logic Programming, Paris/France', MIT Press, pp. 535–548.

GABBAY D. M. [1985], 'N-Prolog: An extension of Prolog with hypothetical implication II—logical foundations, and negation as failure', *Journal of Logic Programming* **2**(4), 251–283.

GABBAY D. M. AND REYLE U. [1984], 'N-Prolog: An extension of Prolog with hypothetical implications I', *Journal of Logic Programming* **1**(4), 319–355.

GALLO G. AND URBANI G. [1989], 'Algorithms for testing the satisfiability of propositional formulae', *Journal of Logic Programmming* **7**(1), 45–62.

GAREY M. R. AND JOHNSON D. S. [1979], *Computers and Intractability*, Freeman, San Francisco.

GENTZEN G. [1935], 'Untersuchungen über das Logische Schliessen', *Mathematische Zeitschrift* **39**, 176–210, 405–431. English translation [Szabo 1969].

GIRARD J.-Y. [1987], 'Linear logic', *Theoretical Computer Science* **50**, 1–102.

HÄHNLE R. AND ESCALADA-IMAZ G. [1997], 'Deduction in many-valued logics: a survey', *Mathware & Soft Computing* **IV**(2), 69–97.

HÄHNLE R. AND KLINGENBECK S. [1996], 'A-ordered tableaux', *Journal of Logic and Computation* **6**(6), 819–834.

HÄHNLE R., MURRAY N. AND ROSENTHAL E. [1997], Completeness for linear regular negation normal form inference systems, *in* Z. W. Raś and A. Skowron, eds, 'Foundations of Intelligent Systems, 10th International Symposium, ISMIS'97, Charlotte, North Carolina, USA', number 1325 *in* 'LNCS', Springer-Verlag, pp. 590–599.

HÄHNLE R. AND PAPE C. [1997], Ordered tableaux: Extensions and applications, *in* D. Galmiche, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 173–187.

HASEGAWA R., FUJITA H. AND KOSHIMURA M. [1997], MGTP: a model generation theorem prover—its advanced features and applications, *in* D. Galmiche, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 1–15.

HASEGAWA R., INOUE K., OHTA Y. AND KOSHIMURA M. [1997], Non-Horn magic sets to incorporate top-down inference into bottom-up theorem proving, *in* W. McCune, ed., 'Proc. 14th International Conference on Automated deduction', Vol. 1249 of *LNCS*, Springer-Verlag, pp. 176–190.

HASEGAWA R., KOSHIMURA M. AND FUJITA H. [1992], MGTP: A parallel theorem prover based on lazy model generation, *in* D. Kapur, ed., 'Proc. 11^{th} International Conference on Automated Deduction', LNAI 607, Springer-Verlag, pp. 776–780.

HINTIKKA K. J. J. [1955], 'Form and content in quantification theory', *Acta Philosohica Fennica* **8**, 7–55.

KANGER S. [1957], *Provability in Logic*, Vol. 1 of *Acta Universitatis Stockholmiensis*, Almqvist & Wiksell, Stockholm.

KLINGENBECK S. AND HÄHNLE R. [1994], Semantic tableaux with ordering restrictions, *in* A. Bundy, ed., 'Proc. 12th Conference on Automated Deduction CADE, Nancy/France', Vol. 814 of *LNCS*, Springer-Verlag, pp. 708–722.

KORF R. E. [1985], 'Depth-first iterative deepening: an optimal admissible tree search', *Artificial Intelligence* **27**, 97–109.

KOSHIMURA M. AND HASEGAWA R. [1999], A proof of completeness for non-horn magic sets and its application to proof condensation. To appear.

KOWALSKI R. [1974], 'Predicate logic as a programming language', *Information Processing* **74**, 569–574.

KOWALSKI R. AND KUEHNER D. [1971], 'Linear resolution with selection function', *Artificial Intelligence* **2**(3), 227–260.

LETZ R. [1993], First-Order Calculi and Proof Procedures for Automated Deduction, PhD thesis, TH Darmstadt.

LETZ R. [1998], Using matings for pruning connection tableaux, *in* 'Proc. 15th International Conference on Automated Deduction (CADE), Lindau', Vol. 1421 of *LNCS*, Springer-Verlag, pp. 381–396.

LETZ R., MAYR K. AND GOLLER C. [1994], 'Controlled integration of the cut rule into connection tableau calculi', *Journal of Automated Reasoning,* **13**(3), 297–338.

LETZ R., SCHUMANN J., BAYERL S. AND BIBEL W. [1992], 'SETHEO: A high-perfomance theorem prover', *Journal of Automated Reasoning* **8**(2), 183–212.

LIS Z. [1960], 'Wynikanie semantyczne a wynikanie formalne (logical consequence, semantic and formal', *Studia Logica* **10**, 39–60. Polish, with Russian and English abstracts.

LLOYD J. W. [1987], *Foundations of Logic Programming*, Second edn, Springer, Berlin.

LOVELAND D. W. [1968*a*], A linear format for resolution, *in* 'Proc. IRIA Symp. Automatic Demonstration', Springer-Verlag, Versailles, France, pp. 147–162. Reprinted in [Siekmann and Wrightson 1983*a*].

LOVELAND D. W. [1968*b*], 'Mechanical theorem proving by model elimination', *Journal of the ACM* **15**(2), 236–251. Reprinted in: [Siekmann and Wrightson 1983*a*].

LOVELAND D. W. [1969], 'A simplified format for the model elimination procedure', *Journal of the ACM* **16**(3), 349–363. Reprinted in: [Siekmann and Wrightson 1983*a*].

LOVELAND D. W. [1972], 'A unifying view of some linear Herbrand procedures', *Journal of the ACM* **19**(2), 366–384.

LOVELAND D. W. [1978], *Automated Theorem Proving. A Logical Basis*, Vol. 6 of *Fundamental Studies in Computer Science*, North-Holland.

LOVELAND D. W. [1987], Near-Horn PROLOG, *in* J.-L. Lassez, ed., 'Proc. Fourth International Conference on Logic Programming, ICLP', MIT Press, Melbourne, Australia, pp. 456–469.

LOVELAND D. W. [1991], 'Near-Horn Prolog and beyond', *Journal of Automated Reasoning* **7**, 1–26.

LOVELAND D. W. AND REED D. W. [1991], A near-Horn Prolog for compilation, *in* J.-L. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of Alan Robinson', MIT Press, Cambridge, MA.

LOVELAND D. W., REED D. W. AND WILSON D. S. [1995], 'SATCHMORE: SATCHMO with RElevancy', *Journal of Automated Reasoning* **14**(2), 325–351.

LUCKHAM D. [1968], Refinements in resolution theory, *in* 'Proc. IRIA Symp. Automatic Demonstration', Springer-Verlag, Versailles, France, pp. 163–190. Reprinted in [Siekmann and Wrightson 1983*a*].

MANTHEY R. AND BRY F. [1988], SATCHMO: A theorem prover implemented in Prolog, *in* 'Proceedings 9th Conference on Automated Deduction', LNCS, New York, Springer-Verlag, pp. 415–434.

MASSACCI F. [1998*a*], Cook and Reckhow are wrong: subexponential proofs for their families of formulae, *in* H. Prade, ed., 'Proc. 13th European Conference on Artificial Intelligence, Brighton', John Wiley & Sons, pp. 408–409.

MASSACCI F. [1998*b*], Simplification: A general constraint propagation technique for propositional and modal tableaux, *in* H. de Swart, ed., 'Proc. International Conference on Automated

Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands', Vol. 1397 of *LNCS*, Springer-Verlag, pp. 217–232.

MONDADORI M. [1988], Classical analytical deduction, Annali dell' Università di Ferrara, Nuova Serie, sezione III, Filosofia, discussion paper, n. 1, Università degli Studi di Ferrara.

MONDADORI M. [1989*a*], Classical analytical deduction, part II, Annali dell' Università di Ferrara, Nuova Serie, sezione III, Filosofia, discussion paper, n. 5, Università degli Studi di Ferrara.

MONDADORI M. [1989*b*], An improvement of Jeffrey's deductive trees, Annali dell' Università di Ferrara, Nuova Serie, sezione III, Filosofia, discussion paper, n. 7, Università degli Studi di Ferrara.

MOSER M., IBENS O., LETZ R., STEINBACH J., GOLLER C., SCHUMANN J. AND MAYR K. [1997], 'SETHEO and E-SETHEO—the CADE-13 systems', *Journal of Automated Reasoning* **18**(2), 237–246.

NONNENGART A., ROCK G. AND WEIDENBACH C. [1998], On generating small clause normal forms, *in* H. Kirchner and C. Kirchner, eds, 'Proc. 15th International Conference on Automated Deduction, Lindau, Germany', number 1421 *in* 'LNCS', Springer-Verlag, pp. 397–411.

OHTA Y., INOUE K. AND HASEGAWA R. [1998], On the relationship between non-Horn magic sets and relevancy testing, *in* 'Proc. 15th International Conference on Automated Deduction (CADE), Lindau', Vol. 1421 of *LNCS*, Springer-Verlag, pp. 333–347.

OPPACHER F. AND SUEN E. [1986], Controlling deduction with proof condensation and heuristics, *in* J. H. Siekmann, ed., 'Proc. 8th International Conference on Automated Deduction', pp. 384–393.

OPPACHER F. AND SUEN E. [1988], 'HARP: A tableau-based theorem prover', *Journal of Automated Reasoning* **4**, 69–100.

PAPE C. [1996], Vergleich und Analyse von Ordnungseinschränkungen für freie Variablen Tableau. (In German), Interner Bericht 30/96, Universität Karlsruhe, Fakultät für Informatik.

PAPE C. AND HÄHNLE R. [1997], Restart tableaux with selection function, *in* G. Gottlob, A. Leitsch and D. Mundici, eds, 'Fifth Kurt-Gödel-Colloquium, KGC'97, Vienna', Vol. 1289 of *LNCS*, Springer-Verlag, pp. 219–232.

PEREIRA L. M., CAIRES L. AND ALFERES J. [1992], SLWV — a theorem prover for logic programming, *in* E. Lamma and P. Mello, eds, 'Proc. Third International Workshop on Extensions of Logic Programming, Bologna', Vol. 660 of *LNCS*, Springer-Verlag, pp. 1–23.

PLAISTED D. A. [1982], 'A simplified problem reduction format', *Artificial Intelligence* **18**, 227–261.

PLAISTED D. A. [1988], 'Non-Horn clause logic programming without contrapositives', *Journal of Automated Reasoning* **4**, 287–325.

PLAISTED D. A. [1990], 'A sequent-style model elimination strategy and a positive refinement', *Journal of Automated Reasoning* **6**, 389–402.

PLAISTED D. A. AND GREENBAUM S. [1986], 'A structure-preserving clause form translation', *Journal of Symbolic Computation* **2**, 293–304.

POSEGGA J. [1993], Deduktion mit Shannongraphen für Prädikatenlogik erster Stufe, PhD thesis, University of Karlsruhe. Diski 51, infix Verlag.

POSEGGA J. AND SCHMITT P. H. [1995], 'Deduction with first-order Shannon graphs', *Journal of Logic and Computation* **5**(6), 697–729.

PRAWITZ D. [1960], 'An improved proof procedure', *Theoria* **26**, 102–139. Reprinted in [Siekmann and Wrightson 1983*b*].

PREISS R. [1998], Beweisvisualisierung und -analyse mit Hypergraphen, PhD thesis, Universität Karlsruhe, Fakultät für Informatik. Published by Shaker-Verlag, Aachen.

RAGO G. [1994], Optimization, Hypergraphs and Logical Inference, PhD thesis, Dipartimento di Informatica, Università di Pisa. available as Tech Report TD–4/94.

REED D. W. AND LOVELAND D. W. [1992*a*], 'A comparison of three prolog extensions', *Journal of Logic Programming* **12**, 25–50.

Reed D. W. and Loveland D. W. [1992*b*], Near-Horn Prolog and the ancestry family of procedures, Technical Report Technical report DUKE–TR–1992–20, Department of Computer Science, Duke University.
*ftp://ftp.cs.duke.edu/dist/techreport/1992/1992-20.ps.gz

Reeves S. [1987], Semantic tableaux as a framework for automated theorem-proving, *in* C. S. Mellish and J. Hallam, eds, 'Advances in Artificial Intelligence (Proceedings of AISB-87)', Wiley, pp. 125–139.

Reiter R. [1971], 'Two results on ordering for resolution with merging and linear format', *Journal of the ACM* **18**, 630–646.

Robinson J. A. [1965*a*], 'Automatic deduction with hyper-resolution', *Int. Journal of Computer Math.* **1**, 227–234. Reprinted in [Siekmann and Wrightson 1983*b*].

Robinson J. A. [1965*b*], 'A machine-oriented logic based on the resolution principle', *JACM* **12**(1), 23–41. Reprinted in [Siekmann and Wrightson 1983*b*].

Schütte K. [1960], *Beweistheorie*, Vol. 103 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen mit besonderer Berücksichtigung der Anwendungsgebiete*, Springer-Verlag.

Shirai Y. and Hasegawa R. [1995], Two approaches for finite-domain constraint satisfaction problem: CP and MGTP, *in* L. Stirling, ed., 'Proc. 12th International Conference on Logic Programming', MIT Press, pp. 249–263.

Shults B. [1997], A framework for using knowledge in tableau proofs, *in* D. Galmiche, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 328–342.

Siekmann, J. and Wrightson, G., eds [1983*a*], *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, Vol. 2, Springer-Verlag.

Siekmann, J. and Wrightson, G., eds [1983*b*], *Automation of Reasoning: Classical Papers in Computational Logic 1957–1966*, Vol. 1, Springer-Verlag.

Slagle J. R. [1967], 'Automatic theorem proving with renamable and semantic resolution', *Journal of the ACM* **14**(4), 687–697. Reprinted in [Siekmann and Wrightson 1983*a*].

Smullyan R. M. [1995], *First-Order Logic*, second corrected edn, Dover Publications, New York. First published 1968 by Springer-Verlag.

Stickel M. E. [1992], 'A Prolog technology theorem prover: A new exposition and implementation in Prolog', *Theoretical Computer Science* **104**(1), 109–129.

Sutcliffe G. and Suttner C. [1997], 'The results—of the CADE-13 ATP system competition', *Journal of Automated Reasoning* **18**(2), 271–286.

Szabo, M. E., ed. [1969], *The Collected Papers of Gerhard Gentzen*, North-Holland, Amsterdam.

Wallace K. [1994], Proof Truncation Techniques in Model Elimination Tableaux, PhD thesis, University of Newcastle, Australia.

Wallace K. and Wrightson G. [1995], 'Regressive merging in model elimination tableau-based theorem provers', *Journal of the Interest Group in Pure and Applied Logics* **3**(6), 921–938. Special Issue: Selected Papers from Tableaux'94.

Index

Topic index