Chapter 3

Tableaux and Related Methods

Reiner Hähnle

Second readers: Uli Furbach and Philippe de Groote.

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 1999] with state-of-the-art resolution-based systems, at least for logic without equality. Tableau calculi are also well suited to cooperate with [Ahrendt, Beckert, Hähnle, Menzel, Reif, Schellhorn and Schmitt 1998] interactive theorem provers used for software verification, which are usually based on sequent calculi. Another reason is the increased need for deduction in various non-classical logics for which tableau calculi are a particularly good match.

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 increased by the plethora of names for closely related systems: connection tableaux, connection method, hyper tableaux, 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 [Baaz et al. 2001, Nonnengart and Weidenbach 2001] (Chapters 5 and 6 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. A brief section on comparison and evaluation of calculi follows. The history of tableau-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 7.

It was an editorial decision to handle certain topics closely related to tableaux not in the present chapter. Equality reasoning in sequent and tableau calculi is discussed in [Degtyarev and Voronkov 2001*a*] (Chapter 10 of this Handbook), material on tableaux for non-classical logics in [Baaz, Fermüller and Salzer 2001, Waaler 2001] (Chapters 20 and 22), implementation techniques for (connection) tableau calculi in [Letz and Stenz 2001] (Chapter 28).

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. Each symbol in P_Σ, F_Σ has a fixed non-negative arity. In addition, there is an infinite set Var of *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. 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, then $f(t_1, \ldots, t_n)$ is a term over Σ.
3. If $t_1, \ldots, t_n$ are terms, P is a n-ary predicate symbol from Σ, then $P(t_1, \ldots, t_n)$ 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 *first-order 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 (resp., disjunction) formula, then $\phi_1 \wedge \cdots \wedge \phi_n$ (resp., $\phi_1 \vee \cdots \vee \phi_n$) is a conjunction (disjunction) 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 $(\forall x)$, resp., of $(\exists x)$.

Formulas that are identical up to associativity of $\vee$ and $\wedge$ are identified. Instead of $(\forall x_1) \cdots (\forall x_r)\phi$ write $(\forall x_1, \ldots, x_r)\phi$. A *literal* is an atom or a negated atom. In the former case, one speaks of a *positive literal*, otherwise of a *negative literal*.

The *size* of a (set of) formula(s) is the number of symbols occurring in it. Let $||\phi||$ stand for the size of a formula or set of formulas.

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 variables. The set of ground terms is abbreviated with $\mathcal{T}^0$. A *propositional formula* is, by definition, a ground first-order formula, in which no quantifiers or function symbols occur.

[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 $(\phi \wedge \psi) \vee (\neg\phi \vee \neg\psi)$.

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 a 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 either the formula *false* or a sentence of the form $(\forall x_1, \ldots, x_n)(L_1 \vee \cdots \vee L_m)$, $m \geq 1$, 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). If $m = 1$, we speak of a *unit clause*. The formula *false* is the *empty clause*. Note that clauses are particular formulas. A formula is in *conjunctive normal form* (*CNF*) if it is of the form $\bigwedge_{i=1}^{r} C_i$, where C_i are clauses.

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 *fact* (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 is also a literal of D; $L \in D$ expresses that the literal L is a literal of clause D. A clause is a *tautology* if it contains literals of the form p and $\neg p$ for some atom p.

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

1. $\sigma(c) = c$
2. $\sigma(\mathit{true}) = \mathit{true}$, $\sigma(\mathit{false}) = \mathit{false}$
3. $\sigma(s(t_1, \ldots, t_n)) = s(\sigma(t_1), \ldots, \sigma(t_n))$ for $s \in F_\Sigma \cup P_\Sigma$, arity of s is n
4. $\sigma(\phi_1 \bullet \cdots \bullet \phi_n) = \sigma(\phi_1) \bullet \cdots \bullet \sigma(\phi_n)$ for $\bullet \in \{\wedge, \vee\}$
5. $\sigma(\neg\phi) = \neg\sigma(\phi)$
6. $\sigma((Qx)\phi) = (Qy)\sigma'(\phi)$ for $Q \in \{\forall, \exists\}$, where $\sigma' = \sigma \backslash \{x \mapsto t \mid t \in \mathcal{T}\} \cup \{x \mapsto y\}$ and y is a variable not occurring in $(Qx)\phi$ such that $\sigma(y) = y$
7. $\sigma(\{\phi_1, \ldots, \phi_n\}) = \{\sigma(\phi_1), \ldots, \sigma(\phi_n)\}$

If $\sigma(x) = x$ for all but finitely many $x \in \text{Var}$ we denote σ by $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where $\{x_1, \ldots, x_n\}$ are exactly the variables with $\sigma(x_i) = t_i \neq x_i$, and $\sigma(x) = x$ for all other variables. Application of substitutions is usually written postfix, composition of substitutions $\sigma \circ \rho$ is denoted by $\rho\sigma$ (note that $\phi(\sigma \circ \rho) = (\phi\rho)\sigma = \phi\rho\sigma$). When for the substitution σ all $\sigma(x)$ for $x \in V \subseteq \text{Var}$ are ground terms one has a *grounding substitution* for the variables V. A *renaming* for a set of variables $\{x_1, \ldots, x_n\}$ is a substitution $\nu = \{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$ such that the y_i are new and different variables in the context, where ν appears. An *idempotent substitution* is a substitution, for which $\sigma \circ \sigma = \sigma$.

Let $|S|$ denote the cardinality of a set S. 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 $\rho = \theta \circ \sigma$. Unifiability of a finite set of terms can be decided in linear time. A unifiable set of terms has always an idempotent MGU. See [Baader and Snyder 2001] (Chapter 8 in this Handbook) for details.

An *instance* or, more precisely, a σ-instance of a clause $C = (\forall x_1, \ldots, x_n)(L_1 \vee \cdots \vee L_m)$ is a formula $(L_1 \vee \cdots \vee L_m)\sigma$, where σ is any substitution. One has a *new instance* of C, if σ is a renaming for $\{x_1, \ldots, x_n\}$. When σ is a grounding substitution for $\{x_1, \ldots, x_n\}$ one has a *ground instance* of C.

The *subformulas* of a formula ϕ are recursively defined as follows:

1. Every formula is a subformula of itself
2. If $\phi = \phi_1 \bullet \cdots \bullet \phi_n$, then any formula of the form $= \phi_{i_1} \bullet \cdots \bullet \phi_{i_r}$ is a subformula of ϕ, where $\{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$ and $\bullet \in \{\wedge, \vee\}$
3. If $\phi = \neg\psi$, then ψ is a subformula of ϕ
4. If $\phi = (Qx)\psi$, then ψ is a subformula of ϕ, where $Q \in \{\forall, \exists\}$

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.

An occurrence of a subformula ρ in $\phi \in \mathcal{L}_\Sigma$ 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 n-ary function symbol $f \in F_\Sigma$ a mapping $\mathbf{I}(f) : \mathbf{D}^n \to \mathbf{D}$ and to each n-ary predicate symbol $P \in P_\Sigma$ a relation $\mathbf{I}(P) \subseteq 2^{\mathbf{D}^n}$.

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

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

For a first-order structure $\mathbf{M}$ over signature Σ with variable assignment μ we define $t^{\mathbf{M},\mu}$ for all $t \in \mathcal{T}_\Sigma$ inductively:

$$\begin{array}{lcl} x^{\mathbf{M},\mu} & = & \mu(x) \text{ for } x \in \mathrm{Var} \\ f(t_1, \ldots, t_n)^{\mathbf{M},\mu} & = & I(f)(t_1^{\mathbf{M},\mu}, \ldots, t_n^{\mathbf{M},\mu}) \text{ for } f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma \end{array}$$

Truth of formulas $\phi \in \mathcal{L}_\Sigma$ in $\mathbf{M}$ under μ, written $(\mathbf{M}, \mu) \models \phi$, is defined as follows:

$$\begin{array}{ll} (\mathbf{M},\mu) \models \mathit{true} & \text{for all } \mathbf{M} \text{ and } \mu \\ (\mathbf{M},\mu) \models \mathit{false} & \text{for no } \mathbf{M} \text{ and } \mu \\ (\mathbf{M},\mu) \models P(t_1, \ldots, t_n) & \text{iff } (t_1^{\mathbf{M},\mu}, \ldots, t_n^{\mathbf{M},\mu}) \in \mathbf{I}(P) \text{ for } P(t_1, \ldots, t_n) \in \mathcal{A}_\Sigma \\ (\mathbf{M},\mu) \models \neg\phi & \text{iff not } (\mathbf{M},\mu) \models \phi \\ (\mathbf{M},\mu) \models \phi_1 \wedge \cdots \wedge \phi_n & \text{iff } (\mathbf{M},\mu) \models \phi_i \text{ for all } i \in \{1, \ldots, n\} \\ (\mathbf{M},\mu) \models \phi_1 \vee \cdots \vee \phi_n & \text{iff } (\mathbf{M},\mu) \models \phi_i \text{ for at least one } i \in \{1, \ldots, n\} \end{array}$$

$(\mathbf{M},\mu) \models (\forall x)\phi$ 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 formula ϕ is *satisfiable* in $\mathbf{M}$, if there exists a μ such that $(\mathbf{M},\mu) \models \phi$. A set of formulas is satisfiable, if each of its members is satisfiable simultaneously under the same variable assignment.

A first-order structure $\mathbf{M}$ over signature Σ is a *model* of a set of formulas $\Psi \subseteq \mathcal{L}_\Sigma$, denoted $\mathbf{M} \models \Psi$, if $(\mathbf{M},\mu) \models \phi$ for all $\phi \in \Psi$ and variable assignments μ. In the light of this definition, CNF formulas are identified with finite sets of clauses. A sentence ϕ is a *logical consequence* of a set of sentences Ψ, denoted $\Psi \models \phi$ if each model of Ψ is also a model of ϕ. ϕ is *valid*, written $\models \phi$, when each structure $\mathbf{M}$ is a model of ϕ.

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

2.1. Proposition. *For all sentences $\phi \in \mathcal{L}_\Sigma$ and sets of $\mathcal{L}_\Sigma$-sentences Ψ: $\Psi \models_\Sigma \phi$ iff $\Psi \cup \{\neg\phi\}$ is unsatisfiable.*

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^*.

Here is a variant of the Löwenheim-Skolem theorem that will be needed:

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

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

2.3. Theorem (Herbrand's Theorem). *Assume that ϕ is a sentence of the form $(\forall x_1,\dots,x_r)\psi$, where ψ is quantifier-free; ϕ is unsatisfiable iff there is an $m \geq 1$ and grounding substitutions θ_i for $\{x_1,\dots,x_r\}$ such that $\bigwedge_{i=1}^m(\psi\theta_i)$ is unsatisfiable. The minimal number m, for which this is possible is the* Herbrand complexity *of ϕ.*

In particular, let S be a finite set of clauses. Then S is unsatisfiable iff there is a finite unsatisfiable set $\widehat{S}$ of ground instances of S.

The result is due to Herbrand [1930*b*]; 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 arbitrary sets P, Q, R,

$$\text{if } \left\{ \begin{array}{ll} (1) & P \neq \emptyset \\ (2) & P \subseteq Q \\ (3) & Q \subseteq R \end{array} \right\} \text{ then } P \cap R \neq \emptyset.$$

The proof is by contradiction: assume $P \cap R = \emptyset$. From (1) we know that there is an element $c \in P$. Now apply (2) to c: if $c \in P$, then $c \in Q$. But we know already that $c \in P$, hence, $c \in Q$. Note that (2) can be seen as an implicit case distinction: either $c \notin P$ or $c \in Q$, where the first case immediately contradicts (1). In the same way, deduce from $c \in Q$ with (3) that $c \in R$. At this point, apply the assumption to c: either $c \notin P$ or $c \notin R$. Both cases yield a contradiction immediately.

The proof is easier to follow, if displayed tree-like as in Figure 1. Observe that case distinctions can be generated schematically depending on the form of their premise. At several points, premises had to be suitably instantiated.

Premises (2), (3), and the assumption are universally quantified, for instance, the assumption says that *for all elements x, x cannot be both an element of P and of R*. 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 $\{x \mapsto c\}$). In general one needs, of course, to apply a premise more than once during a proof. The number of applications, closely related to Herbrand complexity in Theorem 2.3, 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 premise (1) one obtains existentially quantified expressions saying that P must contain at least one element. In automated theorem proving such witness elements are produced by skolemization: the existentially quantified variable is replaced by a "new" term that has not yet an interpretation (again, this is made precise later).

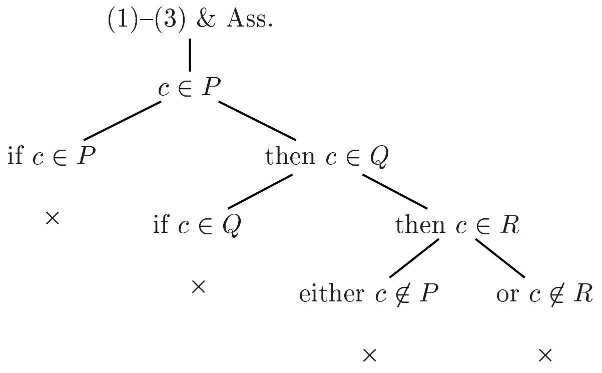


Figure 1: Structure of an informal proof by contradiction and case distinction.

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 between formulas and their types.

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 [1963] and Lis [1960] independently observed[2] 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. By convention, doubly negated formulas and negated logical constants are treated as type α-formulas (with $n = 1$).

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. Without loss of generality assume that no variable of t occurs in the scope of a quantifier in γ_1 or δ_1. If necessary, this can be achieved by renaming the bound variables in γ_1 and δ_1. Associativity of $\wedge$ and $\vee$ justifies conjunctive and disjunctive formulas with an indefinite number of arguments.

Some authors [Lis 1960, Smullyan 1995] prefer to work with *signed formulas*. These are expressions of the form $\mathrm{T}\,\phi$, $\mathrm{F}\,\phi$, 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

[2]See Section 7; see [Fitting 1999] for a full historical account.

[Beckert and Goré 1997, Hähnle 1999].

3.2.2. Tableau Rules

With the help of 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. Premises 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; for example, 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 quantified variable is renamed into a variable not occurring 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 with respect to the more straightforward rule of [Fitting 1990]: 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; in addition, 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 $\mathit{sko}_\delta \in F_{\mathit{sko}}$ such that (a) $\mathit{sko}_\delta > f$ for all $f \in F_{\mathit{sko}}$ 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.

Skolemization rules for normal form computation are due to Andrews [1971] and Bibel [1982*c*]; specific tableau rules seem to appear first in [Brown 1978], they gained wide popularity through [Fitting 1990]. Our δ-rule is from [Beckert, Hähnle and Schmitt 1993], further improvements are possible [Baaz and Fermüller 1995, Giese and Ahrendt 1998, Cantone and Nicolosi Asmundo 1998].

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 2.1 this amounts to checking that the set of sentences $\Phi \cup \{\neg\psi\}$ is unsatisfiable.

$\dfrac{\alpha}{\begin{array}{c}\alpha_1\\ \vdots\\ \alpha_n\end{array}}$	$\dfrac{\beta}{\beta_1 \mid \cdots \mid \beta_n}$	$\dfrac{\gamma(x)}{\gamma_1(y)}$	$\dfrac{\delta(x)}{\delta_1(sko_\delta(x_1,\ldots,x_n))}$
		$y \in \mathrm{Var}$ is new to the tableau.	$x_1,\ldots,x_n$ are the free variables in δ.

Table 2: Rule schemata for tableaux with unification.

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.[3] A *tableau calculus* is a set R of rules each having a set Φ of sentences from $\mathcal{L}_\Sigma$ and, optionally, a tableau for Φ as premises, and another tableau for Φ as conclusion. For each concrete Φ, the transitive closure of these rules defines a set of *tableaux constructed with R for Φ*.

Our main example of a tableau calculus in the present section are *tableaux with unification*:

3.3. Definition. Given a set Φ of sentences from $\mathcal{L}_\Sigma$, a *tableau with unification for* Φ is defined as a tableau constructed with 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 $B \cup \Phi$. Consider an arbitrary instance of a tableau rule schema in Table 2 with premise ψ and n extensions. Obtain the tree T' by extending B with n new linear subtrees whose nodes 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 in this definition incorporates two conditions: 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 (up to renaming of variables) of formulas and their complements in a finite tableau. If Φ is finite this implies that there are systematic procedures for enumerating the (finite) tableaux for Φ.

Branches in a tableau correspond to different subcases in a proof. Formulas occurring in tableaux with unification may contain free variables, hence, Definition 3.3(3) is required to produce an explicit contradiction in a subcase/branch. A tableau proof is finished when this has been achieved for all branches, formally:

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

3.4. 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.

3.5. Remark. In the previous definition, not only formulas from B, but also from Φ are permitted to participate in branch closure. As a consequence, for example, any tableau for $\Phi = \{\mathit{false}\}$ is closed. Some authors prefer to define branch closure with respect to only B. In this case an additional tableau construction rule that fetches formulas from Φ and places them on some branch is required. Our version was chosen, because it allows a more uniform presentation of various calculi.

3.6. Example. We are now in a position to formalize the introductory example from Section 3.1. Sets P, Q, R are represented by unary predicates P, Q, R: their characteristic functions. Then, over the signature $\Sigma = \langle\{P, Q, R\}, \{\}\rangle$, the claim holds if and only if the set Φ consisting of the following $\mathcal{L}_\Sigma$-sentences is unsatisfiable:

$$\begin{array}{ll}
(1) & (\exists x)P(x) \\
(2) & (\forall x)(\neg P(x) \vee Q(x)) \\
(3) & (\forall x)(\neg Q(x) \vee R(x)) \\
(4) & (\forall x)(\neg P(x) \vee \neg R(x))
\end{array}$$

Figure 2 shows a tableau T with unification for Φ. The nodes of the tableau are numbered starting from 5 (the numbers 1–4 refer to the formulas in Φ); an expression $[\mathbf{i}; j]$ is in front 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 can be closed; a closure is indicated by an arc between its complementary literals, labeled with the required MGU. Observe that MGUs are applied to *all* nodes in the tree. For example, the MGU of nodes 12 and 15 is the identity, because x_3 is instantiated during unification of nodes 6 and 14. Convince yourself that the tableau is well-defined.

In the following we say just 'tableau' instead of 'tableau with unification', if it is clear from the context that the latter is meant.

Occasionally, we speak of the size of a tableau. Formally, the *size* of a tableau is the sum of the sizes of the formulas occurring in it.

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 there exists a closed tableau for Φ iff Φ is unsatisfiable. We fixed already that a tableau represents the disjunction of its branches which in turn are considered as conjunctions of their labels. By a standard argument then, the equivalence above is

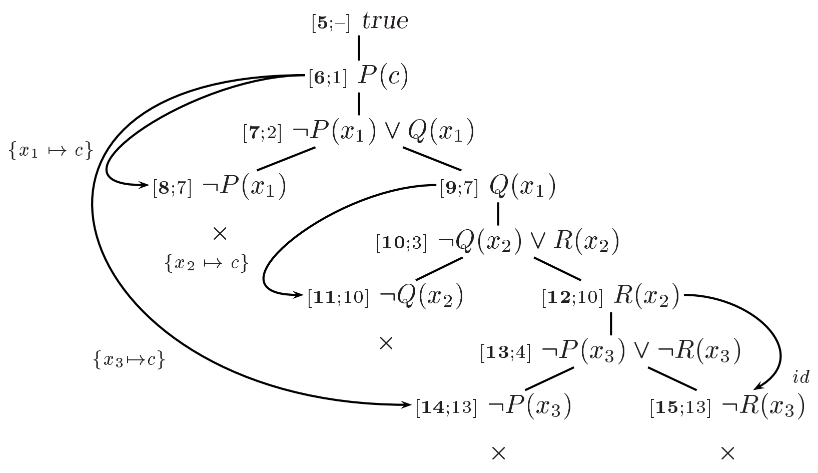


Figure 2: Tableau proof for Φ from Example 3.6.

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.

3.7. 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.

3.8. Definition. A term domain structure $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ is *canonical* iff for all variable assignments μ and all $\delta(x) \in \mathcal{L}_{\Sigma^*}$: 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.

3.9. 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 $\mathbf{M}^* \models \Phi$.*

3.10. 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 $\delta_1(sko_{\delta(x)}(x_1, \ldots, x_n))$, or by (b) replacing a negative occurrence of $\overline{\delta(x)}$ in ϕ by $\overline{\delta_1(sko_{\delta(x)}(x_1, \ldots, x_n)}$, where $x_1, \ldots, x_n$ are the free variables in $\delta(x)$. Then $(\mathbf{M}^*, \mu) \models \phi$ implies $(\mathbf{M}^*, \mu) \models \phi'$.*

3.11. Lemma. *Any tableau T with unification constructed for a satisfiable set of $\mathcal{L}_\Sigma$-sentences is satisfiable.*

Proof. By definition of tableaux with unification, 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 3.9, 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 easy (see [Fitting 1996] for details) and all cases, but the δ-rule case are straightforward. The latter, however, holds by 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:

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

Completeness is stated and proven in Section 3.4.

3.2.5. Universal and Rigid Variables

In general, different instances of the variables in the scope of a universal quantifier are needed in order to close a branch (or a subtableau). In tableaux with unification 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[4], 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 $\psi = (\forall x)(P(x) \vee Q(x)) \rightarrow ((\forall x)P(x) \vee (\forall x)Q(x))$ that would be possible if free variables were not handled rigidly.

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 $\Phi = \{\neg P(c) \vee \neg P(d), (\forall x)P(x)\}$ that consists of two branches, one containing $P(x_1)$ and $\neg P(c)$, and the other containing $P(x_1)$ and $\neg P(d)$. 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 new instance of $(\forall x)P(x)$. In this example, $(\forall x)P(x)$ is a logical consequence of Φ and the formulas already on the tableau (in a sense made

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

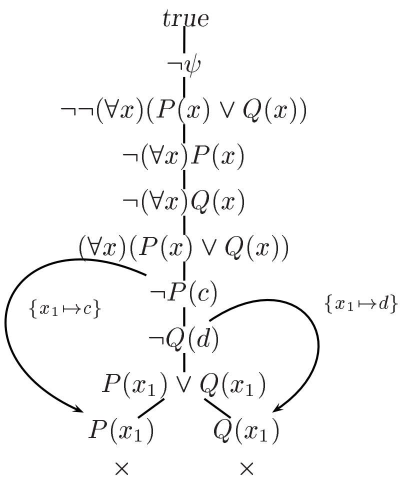


Figure 3: Unsound tableau proof due to non-rigid treatment of free variables.

precise in Definition 3.13), hence, $(\forall x)\phi(x)$ 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 using more general closing substitutions, yields shorter tableau proofs, and in many cases reduces the search space.

3.13. 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 $x \in \text{Var}$. Formula ϕ is called *universal* on B with respect to x if every model of T is also a model of T'.[5] Denote with $\text{UVar}(\phi, B)$ the variables with respect to which ϕ is universal on B.

Instead of designing a closure rule that takes universal variables into account (Definition 3.3(3)), we generalize the concept of a unifier:

3.14. Definition. A substitution σ is a *unifier* of formulas ϕ, ϕ' on a branch B 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 $U = \text{UVar}(\phi, B) \cap \text{UVar}(\phi', B)$ 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 calculus of tableaux with unification 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

[5]When obvious, a formula ϕ being universal on a branch B with respect to a variable x is just referred to as "the universal formula ϕ," and x as "the universal variable x."

proofs drastically): 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 with respect to x, formally:

3.15. Lemma. *A formula ϕ on a branch B of a tableau T is universal with respect to 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 with respect to x; or*
3. *a β-rule to a formula β that is universal on B with respect to x, and x does not occur in any $\beta_i \neq \phi$.*

The proof of soundness of tableaux with unification (Theorem 3.12) can accommodate the universal formula technique. Bibel [1982] 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 the n-ary rule cannot polynomially simulate tableaux based on (3.1) with respect to the minimal proof size, see also Section 6. In the present paper I work with the n-ary rule to achieve maximum uniformity among clausal and non-clausal tableaux. There is no loss of generality in doing so, because it is obvious that rule (3.1) can polynomially simulate the n-ary rule.

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 $\bigvee_{\psi\in\Phi}\overline{\psi}$. There 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 can be answered affirmative, for the second, a fully satisfactory answer is not yet available. This requires some explanation.

Definition 3.3 consists of a bunch of rules that define how to construct a tableau. In Section 3.4 it is shown that there exists a closed tableau with unification for any given unsatisfiable set of sentences. This property of a calculus is called *completeness*. There is only a finite number of rules that can be applied to each given tableau, so it is a routine task to breadth first search for tableau proofs.

It would be much better, of course, if there were no need for search. Define a *tableau proof procedure* to be a tableau calculus equipped with a function F that, given a set of sentences Φ and a tableau T, computes in deterministic polynomial time (in the size of Φ and T) the next rule to be applied on T. It can be thought of as a "deterministic calculus": its rules allow to construct at most one successor tableau from any given tableau and set of sentences. If the tableau proof procedure computes a tableau proof for any given unsatisfiable sentence, it is called *strongly complete*. The function F is called a *computation rule*. Let me point out why it is a difficult problem, to find strongly complete tableau proof procedures.

Usually, a great number of rules is applicable to any given tableau. More precisely, one must first 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 nondeterminism with the phrases *select branch*, *select mode*, *select formula*, and *select pair* in the following. In the propositional case no substitutions occur and, to arrive at a strongly complete tableau proof procedure, it suffices to select each non-literal formula exactly once on each branch in any order.

In the first-order case, one needs to apply rules more than once to certain formulas (otherwise, first-order logic were decidable). Making an arbitrary choice for a computation rule in the first-order case, however, results in general 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 propositional formula indefinitely. In an obvious way, the formula $Q \wedge \neg Q$ is treated *unfair*. This motivates the following definition.

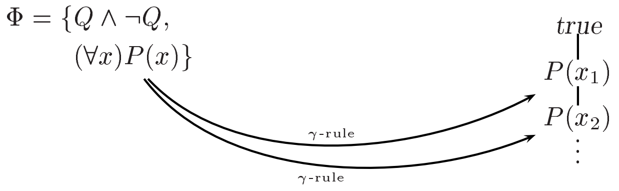


Figure 4: Incompleteness caused by unfair *select formula*.

3.16. Definition. The set of tableaux with unification for a given set of sentences $\Phi \subset \mathcal{L}_\Sigma$ is partially ordered relative to a computation rule F, where the successor of a tableau T for Φ is the tableau computed from T by F. This defines a (possibly infinite, if Φ contains at least one type γ formula) ascending chain starting with the initial tableau T_0 for Φ and supremum T_∞ (which exists by Zorn's lemma).

A computation rule F is *fair* if for all Φ the following holds for all branches B in tableau T_∞ for Φ:

1. All formulas of type α, β, and δ occurring on B or in Φ were used to expand B (by applying the appropriate expansion rule)
2. All type γ formulas occurring on B or in Φ were used infinitely often to expand B (by applying the γ-rule).

It is simple to construct a fair computation rule, but this is not sufficient for strong completeness, because the above notion of fairness says nothing about closure. Combining fair application of the expansion rules with fair application of the closure rule, however, is a difficult problem, because tableaux with unification are *destructive*:

3.17. Definition. A tableau calculus is *non-destructive* if all tableaux that can be constructed with the help of its rules 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. Alas, it is 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.

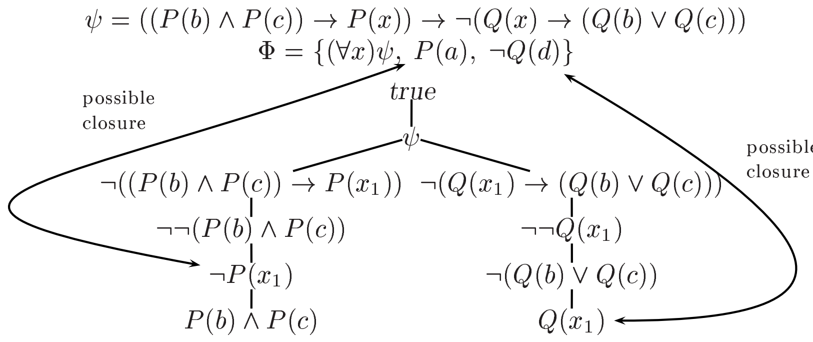

$\psi = ((P(b) \wedge P(c)) \to P(x)) \to \neg(Q(x) \to (Q(b) \vee Q(c)))$
$\Phi = \{(\forall x)\psi,\ P(a),\ \neg Q(d)\}$
possible closure
true
ψ
$\neg((P(b) \wedge P(c)) \to P(x_1))$
$\neg(Q(x_1) \to (Q(b) \vee Q(c)))$
possible closure
$\neg\neg(P(b) \wedge P(c))$
$\neg\neg Q(x_1)$
$\neg P(x_1)$
$\neg(Q(b) \vee Q(c))$
$P(b) \wedge P(c)$
$Q(x_1)$

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

Tableau with unification are (trivially) not destructive for propositional logic and for quantifier-free sets of sentences.

A non-destructive tableau calculus equipped with a fair computation rule gives a strongly complete proof procedure. Examples of non-destructive tableau calculi are Smullyan's [1995] ground tableaux and Fitting's [1996] tableaux with delayed instantiation rule, see below. As mentioned already, tableaux with unification *are* destructive. The culprit is the closure rule, Definition 3.3(3).

Independently of being destructive, a complete tableau calculus may fail to be proof confluent:

3.18. Definition. A tableau calculus 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 calculus contains no "dead ends", from where no proof can be found. A strongly complete tableau proof procedure is trivially a proof confluent tableau calculus. Thus, proof confluence is a necessary prerequisite for strong completeness.

A destructive tableau proof procedure still might be strongly complete, but as witnessed by the example in Figure 5, it might as well be not. At the present time, no strongly complete, destructive tableau proof procedure is known that works well in practice (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 tableaux with unification into a strongly complete proof procedure is to separate application of expansion and closure rules. Under a fair computation rule, delay the application of the closure rule 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 inefficiencies: 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 application, the whole tableau must be tested for closure, which is very redundant. If more sophisticated data structures were used and the different MGUs available to close each branch were maintained as a tableau-wide constraint system that can be incrementally tested, then this approach might still be worth a try. There is experimental evidence to support this [Giese 2000].

Another option for implementing tableau proof search, which was mentioned already, comes from the observation that its nondeterminism is locally finite—from each tableau only a finite number of successor tableaux can be constructed. Envisage tableau proof search as a, possibly infinite, search tree whose nodes are tableaux. The root node contains the trivial tableau. The successors of a node are all the tableaux that can be constructed from it with one of the available tableau rules. Nodes that contain a closed tableau are success nodes. Even though the whole search tree is infinite, success nodes occur at finite depth and can be searched for in a breadth first manner. This approach is impractical, however, 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 much more space efficient [Korf 1985].

DFID tableau proof search is based on a mapping m from $\mathbb{N}$ to subsets of the tableaux that can be constructed with a given calculus, such that $\bigcup_{i \in \mathbb{N}} m(i)$ contains all these tableaux. Common choices[6] for m, which is called *completion mode*,

[6]These options are implemented, for example, in the provers $_3T^AP$ [Beckert, Hähnle, Oel and

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

Now the parts of the search space containing the tableaux in $m(1), m(2), \ldots$ are successively enumerated by depth first search with backtracking. To increase efficiency of the search it is important to get rid of as many nodes in the search space 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 computation rule that gets rid of *select formula* [Fitting 1996]. This leaves the—destructive—branch closure.

Tableau proof search based on DFID with backtracking over nodes corresponding to *select mode* and *select pair* is elegant and fast, when implemented in logic programming languages [Stickel 1992, Baumgartner and Furbach 1994, Beckert and Posegga 1995].

3.4. Tableau Completeness

The preceding discussion shows that it is difficult to ensure strong completeness of tableaux with unification. 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) may contain an infinite number of formulas whose order is irrelevant. A model can be immediately constructed for any Hintikka set.

3.19. 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 $t \in \mathcal{T}^0_{\Sigma^*}$;
5. if $\delta(x) \in H$, then $\delta_1(t) \in H$ for some $t \in \mathcal{T}^0_{\Sigma^*}$.

3.20. Lemma (Hintikka). *Every Hintikka set is satisfiable.*

Proof. An Herbrand model over the signature Σ^* is simply defined by setting $P^{\mathbf{I}}(t_1, \ldots, t_k) = \mathit{true}$ iff $P(t_1, \ldots, t_k) \in H$ for $P(t_1, \ldots, t_k) \in \mathcal{A}^0_{\Sigma^*}$. By induction on the structure of formulas in H it is easy to prove that $\mathbf{M} \models H$. □

Sulzmann 1996] and Setheo [Moser et al. 1997].

3.21. 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 sequence of tableaux for Φ constructed with a fair computation rule, containing no closure rule applications, and with limit T_∞. We define a particular grounding 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 $\mathcal{T}^0_{\Sigma^*}$.

If we want to extract a model of Φ from B_k, then the instances of the 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, 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. The tree $T_\infty\sigma_\infty$ is finitely branching and the distance of all formulas involved in closures to the root node is finite. Then, by König's Lemma[7], 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. This is done with a standard lifting argument: $\sigma_\infty = \sigma \circ \sigma_r \circ \sigma_{r-1} \circ \cdots \circ \sigma_1$, 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 quite a 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

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

expansion step is uniquely determined once the position, where it is to be applied, is fixed. In the case of formulas in negation normal form (NNF), the situation is even simpler: up to variable instantiation, any formula occurring in a tableau for a formula ψ in NNF is just a subformula of ψ. Therefore, in the case of quantifier-free sentences in NNF, a tableau branch can be viewed as a sequence of positions of certain subformulas.

Such representations were first suggested independently by Davydov [1973], Bibel and Schreiber [1975], and Andrews [1976]. Bibel [1979] and Andrews [1981] defined procedures to check the validity of first-order formulas in NNF (in this case, one must record substitutions as well).

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 not containing *true*, *false*, written in a two-dimensional notation: the immediate subformulas of a disjunctive formula are stacked vertically onto each other, while the immediate subformulas of a conjunctive formula are written in a horizontal row and enclosed between square brackets; literals are unchanged. To simplify things we start with propositional logic.

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

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

A *path* through a matrix M is a set $\pi = \{M_1, \ldots, M_r\}$ of occurrences of submatrices (which can be literals) defined inductively:

1. For every matrix M, $\{M\}$ is a path through M (note that M can be a literal).
2. If M consists of rows $M_1, \ldots, M_r$ and π is a path through some M_i for $i = 1, \ldots, r$, then π is a path through M.
3. If M consists of columns $M_1, \ldots, M_r$ and π_i is a path through M_i for all $i = 1, \ldots, n$, then $\pi_1 \cup \ldots \cup \pi_n$ is a path through M.

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

Consider any branch of B of any non-trivial tableau for a propositional NNF formula ψ. Let π be those formulas of B that were not used as a premise of a rule application on B. Then it is fairly easy to prove by induction that π is a path through the matrix of ψ and, vice versa, each path through ψ is contained in a branch of some tableau for ψ. Thus, complementary formulas on branches are just complementary submatrices in paths. A pair of complementary formulas on a branch is called *connection* by Bibel and *mated* 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 $\mathbf{C}$ is called *spanning* by Bibel [1981] who was the first to give this kind of matrix characterization of unsatisfiable formulas (without making use of the mentioned correspondence between matrices and tableaux):

3.23. 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 certain tableau procedures.

An exact tableau counterpart to the non-clausal connection method with some additional restrictions is 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 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]. Its main advantage is that it generalizes to logics not permitting skolemization, such as intuitionistic logic [Voronkov 1996].

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 new instances, which is closely related to the Herbrand complexity in Theorem 2.3. 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 tableaux with unification.

Matrix methods are closer to data structures allowing efficient implementation than tableaux. This positive feature, on the other hand, makes their the formal presentation of matrices very technical. I suspect that this a main reason why many refinements were conceived within the more abstract—and redundant—tableau formalism. Even worse, the less redundant structure of matrices can actually be in the way of extensions or optimizations: for example, certain rules needed to deal efficiently with formulas that contain equalities, add a new literal to a branch that is a logical consequence of a literal set C on the same branch (for example, the basic superposition calculus of [Degtyarev and Voronkov 1998])—this cannot be done in an obvious way within a matrix framework, because the paths containing C are not explicitly represented. Similarly, simplification as discussed in Section 3.5.3 below, cannot be easily incorporated into a matrix framework.

But there is an important merit of matrix formulations compared to 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.

An extensive overview over various calculi from the point of view of matrices is [Bibel and Eder 1992].

Finally, it should be mentioned that apart from matrices further formula representations exist that try to avoid redundancy: 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 and BDDs are closely related to tableaux [Posegga 1993, Preiß 1998]. They share, however, the drawback that their 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 expanded 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 below B (define an extension β_i to be *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 take advantage 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*, that is, the effects of the β-rule application are undone, see Figure 6.

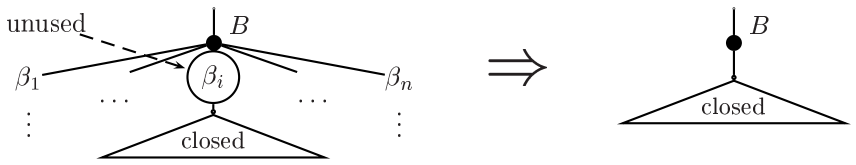


Figure 6: Pruning irrelevant parts of a tableau proof.

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 propositional 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 $true \vee \theta \Rightarrow true$, etc. In contrast to branching, simplification is an inexpensive operation and can be computed in (low) polynomial time in the size of formulas on branches. The well-known unit resolution and pure literal rule subprocedures of the Davis-Putnam-Loveland-Logeman procedure [Davis, Logemann and Loveland 1962] (see also Section 5.4) are special cases of Massacci's [1998*b*] simplification rule who demonstrated its effectiveness for (modal) propositional logic. It remains to be seen, however, if a useful variant for first-order tableau with unification will emerge.

4. Clause Tableaux

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

- Simplified notation leads to easier detection of new refinements
- Efficient implementability, for example, by compilation to abstract machines
- Completeness proofs stay manageable
- Comparability (most deduction procedures are implemented on the clause level)

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

- Some applications (such as software verification) expect proofs on the non-clausal level: back-translation 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 the models of a formula to be preserved, and then in the worst case its CNF has exponential size

In summary, there is considerable incentive to generalize the results that follow (partially this has been done, for example, in [Hähnle and Klingenbeck 1996, Hähnle, Murray and Rosenthal 1997]), but I believe the material to be 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 in [Baaz et al. 2001, Nonnengart and Weidenbach 2001] (Chapters 5 and 6 of this Handbook).

4.2. Clause Tableau Proofs, Soundness, Completeness

4.2.1. Clause Tableaux

Let us start by stating suitably simplified versions of Definitions 3.3 and 3.4.

4.1. Definition. Given a set S of clauses from $\mathcal{L}_\Sigma$, a *clause tableau for S* is defined as a tableau constructed with 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 $L_1 \vee \cdots \vee L_r$ a new 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 (that is, $T' = T\sigma$), then T' is a tableau for S and branch $B\sigma$ is marked as *closed* (closure rule).

Clauses are first-order formulas, so the extension rule is composed of several applications of the expansion rule 3.3(2), which justifies the change in terminology.

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. The formula set of Example 3.6 can be transformed in a CNF that consists of the following clauses:

$$\begin{array}{ll} (1) & P(c) \\ (2) & \neg P(x) \vee Q(x) \\ (3) & \neg Q(x) \vee R(x) \\ (4) & \neg P(x) \vee \neg R(x) \end{array}$$

A clause tableau for this clause set is displayed in Figure 7. Observe that the nodes are a subset of the nodes of the tableau in Figure 2.

In the following, closed branches are not indicated by arrows between participating literals anymore, but merely by a horizontal bar and the closing MGU below their leaf.

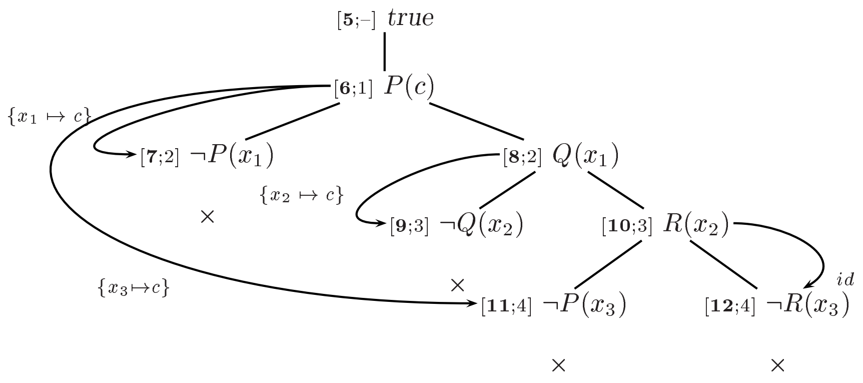


Figure 7: Clause tableau proof of Example 4.3.

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 clauses from the input set and branch literals. This simplifies some definitions.

4.2.2. Soundness and Completeness

Soundness of clause tableaux follows immediately from Theorem 3.12 by observing that clauses are particular first-order formulas and extension rule 4.1(ii) can be composed of several applications of rule 3.3(2).

Completeness could be obtained easily by suitable simplification of the proof of Theorem 3.21, but in the clausal case a more modular approach is useful. Following Robinson [1965], 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 completeness proofs of the following tableau refinements and either is obvious or at most requires a sketch. 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 a clause set, $\widehat{S}$ a 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:

If T' is a clause tableau, τ a substitution such that $T'\tau$ is closed, and ρ an MGU that closes any branch of T', then $T'\rho\tau = T'\tau$. (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 a new instance of the clause $C \in S$ of which $\widehat{C}$ is a ground instance. Obviously, $T^0\tau = \widehat{T}$ for a suitable grounding 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 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.[8]

As to completeness, let us look first at the ground case. To minimize iterated efforts, we proceed in a schematic way. The following *ground completeness schema* for any given clause tableau restriction, let us call it X-tableau, is proven:

If the finite ground clause set S is unsatisfiable, then there is an X-tableau proof for S. (4.2)

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.

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

[8]When the computation rule of *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 $T\tau$ appears as a subset of the nodes of $\widehat{T}$.

Proof (*Schema*). Herbrand's Theorem 2.3 provides a finite, unsatisfiable set $\widehat{S}$ of ground instances of S. By a suitable instance of (4.2) 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 prove a suitable instance of (4.2), whenever 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 view 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 were pioneered by Andrews [1981] and Bibel [1982*b*].

4.3.1. Connection Tableaux

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.

4.6. 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 8. 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.

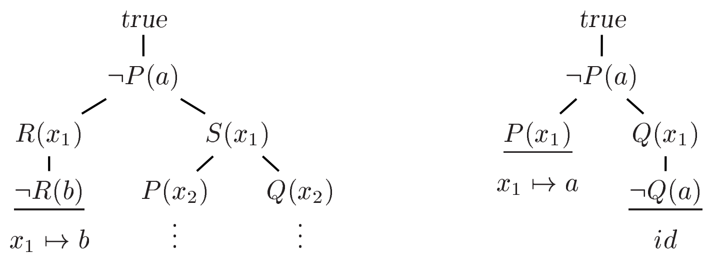


Figure 8: Redundant nodes in a tableau.

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

The tableau on the right in Figure 8 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 $T \neq \mathit{true}$ 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 new 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, $L_1 \vee \cdots \vee L_r$ a new 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 *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. This justifies the change of terminology.

It is important to note that while clause tableaux are proof confluent, connection tableaux are not:

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

Proof. Consider $S = \{P, \neg P, Q\}$ and let Q be the clause used in the initial step. It is 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 extension 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[9] shows that even this results in incompleteness:[10]

4.9. 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 leads to greedy reduction with $P(x)$ (and $\neg P(a)$) and the proof is stuck at the open branch containing $Q(a)$. With a different starting clause, a proof with greedy reduction 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.

[9]Personal communication.

[10]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 reducing greedily without noticing the incompleteness problem.

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 partially restores proof confluence:

(ii$''$) Let T be a tableau for S, B a branch of T *containing L not necessarily as leaf*, $L_1 \vee \cdots \vee L_r$ a new 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 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.*

4.10. 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 is 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 constructing certain redundant proofs is *regularity*:

4.11. Definition. A clause tableau is *regular*, if none of its branches contains more than one occurrence of the same literal.

4.12. 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 9. 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 $L(t_1, \ldots, t_m)$ and $L(t'_1, \ldots, t'_m)$ 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 $x_2 \mapsto 0$.

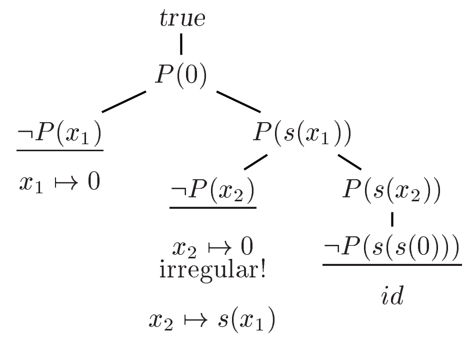


Figure 9: Advantage from regularity.

The following standard lemma is needed in the proof 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 below was first proven (differently) in [Letz 1993]. The present proof is from [Hähnle et al. 1997].

4.13. 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 $D \neq D'' \in S''$.*

4.14. Theorem (Completeness). *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 clause $C_j \in S$, that is not a unit clause, there is a closed regular connection tableau for S whose initial step uses C_j. If 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 10) 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 consisting just of an initial step that uses C_j (upper middle part of Figure 10).

For all $i \in \{1, \ldots, n\}$ let $C'_j = L_i$ and $S_{L_i} = (S - \{C_j\}) \cup \{C'_j\}$. By Lemma 4.13(i), C'_j is contained in an mu subset S'_{L_i} of S_{L_i}. Hence, $\overline{L_i}$ occurs in a clause C^i of S'_{L_i}. Moreover, S'_{L_i} contains less literals than S.

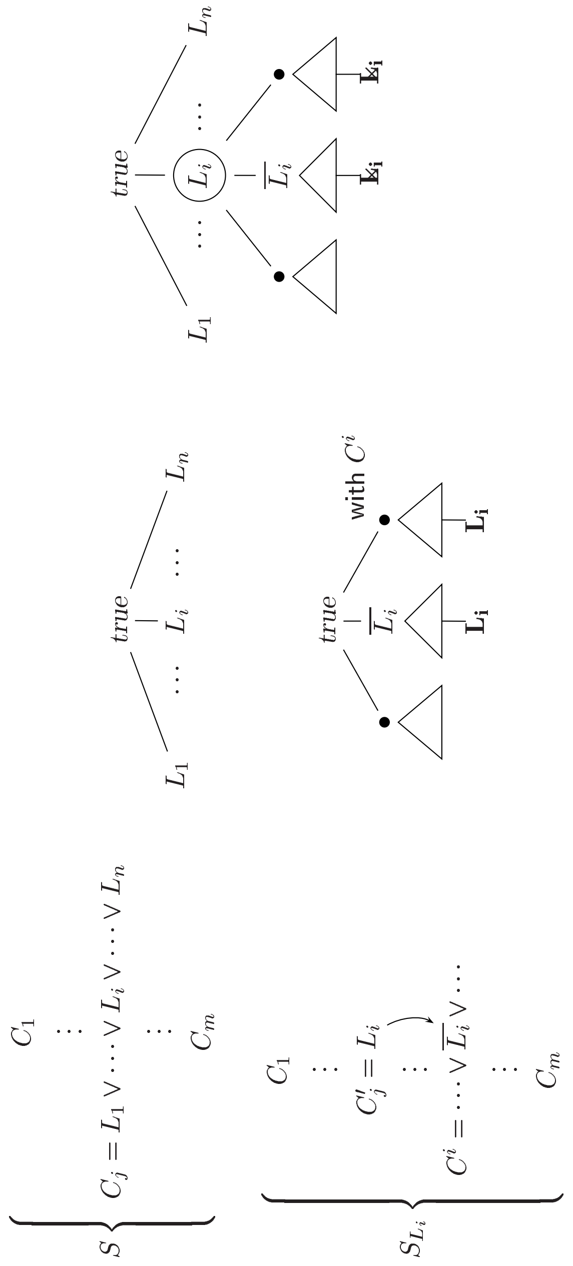


Figure 10: Illustration of the proof of Theorem 4.14

.

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'_{L_i} yields a closed regular connection tableau T_i for $S_{L_i} \supseteq S'_{L_i}$, where the first extension step uses C^i (lower middle part of Figure 10).

By Lemma 4.13(ii), L_i occurs at most in C'_j and is therefore only used in extension steps with the unit clause 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 extension steps with unit clause L_i simply can be replaced by reduction steps with the circled occurrence of L_i. Because of $S_{L_i} - \{C'_j\} \subseteq S$, the result is a regular connection tableau for S. $\square$

The proof has an interesting consequence resulting in a restriction for initial clauses, which is of importance later on in Section 5.3: each mu set of (not necessarily ground) clauses S trivially contains a negative clause (if not, it can be satisfied by the constantly true interpretation). Hence, one of the negative clauses of S is relevant and, therefore, the initial extension step can be restricted to negative clauses.

4.15. Corollary. *Regular connection tableaux are complete even when the first extension step must use a negative clause.*

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 first-order clause tableau is rendered irregular by an extension step of branch B with a non-tautologous 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, therefore, amounts to avoiding 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* with respect to 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.

If $\mathbf{I}_B$ is not yet a model of S (that is, B is not a saturation itself), then there must be a reason for it in the form of clauses $C \in S$ not satisfied by $\mathbf{I}_B$. We try to complete $\mathbf{I}_B$ to a model of all clauses in S by adding to it selected literals from the unsatisfied clauses. These clauses are, by definition, no tautologies and are regular on B.

This idea is formalized in the following section.

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:

$$\widetilde{B}_f = B \cup \bigcup_{\substack{C \in S \\ \mathbf{I}_B \not\models C}} f(C) \tag{4.3}$$

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

$$\{C \mid C \in S, \mathbf{I}_B \not\models C \text{ and } (\overline{f(C)} \cap \widetilde{B}_f \neq \emptyset \text{ or } f(C) = \emptyset)\} \tag{4.4}$$

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 131 (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 11).

For the leftmost branch $B = \{\neg Q\}$ one obtains $\widetilde{B}_f = B \cup \{S, R, \neg P\}$, 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 down-sizing the search space with selection functions.

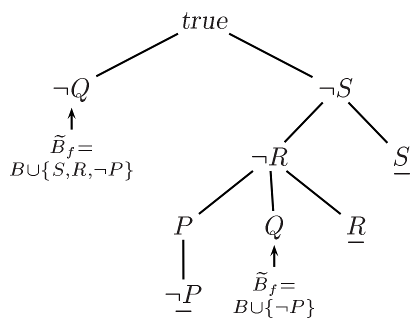


Figure 11: Tableau with selection function.

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. □

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

$$f_<(C) = \{L \mid L \text{ maximal in } C \text{ with respect to } <\} \ .$$

The restriction $\mathbf{I}_B \not\models C$ in (4.3), (4.4) 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 connected extension steps.

In ordered tableaux, by definition, $f(C) \neq \emptyset$; hence, only restart clauses of the kind that are connected via f can occur. It is a natural question, if one can get rid of restart clauses altogether. In [Hähnle and Klingenbeck 1996] it is shown that ordered tableaux without restart steps (that is *each* extension step is weakly connected via f) are incomplete for certain total orderings. It is not obvious, for which selection functions a calculus without restart clauses might be complete (besides the trivial selection function defined by $f(C) = C$ for all $C \in S$, which gives regular clause tableaux without any further restriction).

On the other hand, one can impose a restriction, which is complementary to that of ordered tableaux, in the sense that one permits *only* restart clauses of the kind, where $f(C) = \emptyset$:

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

Whenever f is consistent with respect to S, only restart steps with clauses that have no selected literals are possible. The special case when S is consistent, $|f(C)| \leq 1$, and there is exactly one restart clause in S, is a complete calculus 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. More precisely, call f *stable with respect to 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'. It is obvious from the discussion following 4.4 that tableaux with selection functions that are stable with respect to substitutions lift without problems.

Implementation of first-order tableaux with selection function poses similar problems as regularity. In addition to regularity constraints, selectedness constraints are derived from (4.4) and take the form $L \in f(C)$ [Pape 1996, Hähnle and Pape 1997].

Checking selectedness constraints for satisfaction can be expensive (NP-complete). If one decides to suppress their generation, then the resulting calculus can 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 used in a tableau already. 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 at once that selection functions need not be stable with respect to arbitrary substitutions in tableaux with input selection function, rather, stability with respect to variable renamings is sufficient [Hähnle and Pape 1997].

4.6. Hyper Tableaux

Recently, tableau calculi based on hyper extension rules gained considerable attention [Bry and Yahya 1996, Baumgartner et al. 1996, Shults 1997, Baumgartner 1998]. This is not surprising, because hyper-resolution [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 is discussed below.

It is well-known that hyper-resolution 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 (4.4) on clause candidates for extension 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:

$$\{C \mid C \in S, \mathbf{I}_B \not\models C \text{ and } \overline{f(C)} \subseteq B\} \tag{4.5}$$

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:

$$\overbrace{\overline{L_1} \wedge \cdots \wedge \overline{L_m}}^{\text{selected literals}} \rightarrow \overbrace{L_{m+1} \vee \cdots \vee L_n}^{\text{not selected literals}} \tag{4.6}$$

The premise of C viewed as a rule (that is, the set of literals $\{\overline{L_1}, \ldots, \overline{L_m}\}$) is equivalent to $\{\mathit{true}\}$, when there are no selected literals; likewise, the conclusion of C viewed as a rule (that is, the set of not selected literals $\{L_{m+1}, \ldots, L_n\}$) is equivalent to $\{\mathit{false}\}$, when there are only selected literals. Note that the premise of a rule contains *complements* of selected literals.

As *true* occurs on every tableau branch, the following reformulation of (4.5) is possible:

$$\{C \mid C \in S,\, \mathbf{I}_B \not\models C \text{ and all literals of the premise of } C \text{ occur in } B\} \tag{4.7}$$

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: $\{\underline{Q} \wedge \underline{S} \to \mathit{false},\, \underline{R} \to S,\, \mathit{true} \to P \vee Q \vee R,\, \underline{P} \to \mathit{false}\}$. The only possible hyper tableau for f_N and S_2 is as follows:

$$\begin{tikzcd} & \mathit{true} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\ P \arrow[d, dash] & Q & R \arrow[d, dash] \\ \underline{\mathit{false}} & & S \end{tikzcd}$$

Note that the left branch is closed. A hyper tableau for S_2 must begin with the only restart clause. Closure of the leftmost branch and extension of the rightmost one are both mandatory. No other 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.[11]

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 an open branch B in a hyper tableau for S, in which all possible extension steps were made, and let $\widetilde{B}_f$ be the extension of B, see (4.3). The set $\widetilde{B}_f - B$ is consistent, because f is consistent, so when $\widetilde{B}_f$ 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 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

[11]In fact, for consistent selection functions that select at most one literal per clause conditions (4.4) and (4.5) become identical.

contradicting the assumption that all possible extension steps were made. So at least one literal in $f(C)$ is still present in B', thus $\mathbf{I}_{B'} \models C$. □

The set B' that induces the model $\mathbf{I}_{B'}$ in the previous proof can be defined more directly as $B' = B \cup \{L \in f(C) \mid C \in S, \overline{L} \notin B\}$.

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 Example 4.21 one reads off the models $\{Q, \underline{\neg S}, \underline{\neg R}, \underline{\neg P}\}$ and $\{R, S, \underline{\neg Q}, \underline{\neg P}\}$.

Selection function f_N is an example of a *complete selection function*[12], which selects at least one of $P, \neg P$ for each ground atom in the signature of ground clause set S.[13] In the presence of complete selection functions reduction steps are not required, because they would occur with two complementary, not selected literals whose existence is exactly what has been ruled out;[14] 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. Completeness of the selection function is a necessary condition in the corollary: consider the clause set $S = \{\mathit{true} \to P, \mathit{true} \to \neg P\}$ in which nothing is selected (that is, its corresponding selection function is consistent). Both clauses are restart clauses, so P and $\neg P$ are present on any branch via extension, 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:

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

[13]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*?

[14]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.

(ii‴) Let T be a hyper tableau for S and consistent selection function f, B a branch of T, $C = \overline{L_1} \wedge \cdots \wedge \overline{L_m} \rightarrow L_{m+1} \vee \cdots \vee L_n$ a new instance of a clause in S. If there is an MGU σ such that all literals of the premise of $C\sigma$ occur in $B\sigma$ (see also (4.7)), $\mathbf{I}_{\exists B\sigma} \not\models \forall C\sigma$, 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.

As before, selection functions must be liftable, that is, stable with respect to substitutions to ensure completeness. As can be seen from (ii‴), in addition to the regularity and tautology check (that is, $\mathbf{I}_{\exists B\sigma} \not\models \forall C\sigma$), one must compute a set $B_0 \subseteq B$ as well as a simultaneous MGU of $\{\{L, L'\} \mid L \in B_0, L' \in \overline{f(C)}\}$ to perform a hyper extension step on branch B with clause C. This is easily seen to be an instance of the first-order clause subsumption problem, which is NP-complete [Garey and Johnson 1979]. This complexity is implicitly present in proof search without hyper steps as well, although on a different level. It does not indicate inferiority of hyper tableaux as a calculus for proof search.

A more interesting question takes up the discussion of Section 3.3: how deals one in the context of hyper tableaux with the difficulties to define combined fair clause and substitution selection in destructive calculi? For positive hyper tableaux, several strategies are found in the literature:

The first option is to fix a computation rule and accept incompleteness. Typically, one minimizes $n-m$ in (ii‴) and/or the size of terms in MGUs. For specific problem domains, where specific heuristics could be developed, success is reported in [Brown 1978, Shults 1997]. Another form of incompleteness (in syntactical expressivity of the logic) 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 in the conclusion of C (that is, in not selected literals) occur also in its premise (that is, in selected literals).

Observe that range-restricted clauses with no selected literals (premise 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] (by Corollary 4.23 reduction steps are not necessary). These calculi are further discussed in Section 4.6.3 below.

EP-tableaux are a variant of positive hyper tableaux for non-clausal first-order logic without function symbols that can detect finite satisfiability [Bry and Torge 1998]. The key ingredients are (i) a non-clausal version of range-restrictedness called *positive formulas with restricted quantifications* (PRQ formulas)—in particular, type γ formulas are of the form $(\forall\vec{x})(\phi(\vec{x}) \rightarrow \psi)$ and are only used on a branch

B with $\mathbf{I}_B \models \phi\sigma$ for some σ; (ii) only formulas ψ with $\mathbf{I}_B \not\models \psi$ are added to a branch B, (iii) a modified δ-rule that explicitly enumerates the current model domain:

$$\begin{array}{c|c|c|c} \multicolumn{4}{c}{\delta(x)} \\ \hline \delta_1(c_1) & \cdots & \delta_1(c_m) & \delta_1(sko_\delta) \end{array}$$

$c_1, \ldots, c_m$ are all
constants on the branch.

Because of (i), EP-tableaux have no free variables, so they are not destructive. In the function-free case, finite term domain models have a finite representation on branches; (iii) ensures that enough information for explicit model construction is present. Together with (ii) and a fair computation rule, this is sufficient to detect all finite term domain models on finite EP-tableau branches.

Back to general clause sets, call a variable violating range-restrictedness of a clause *and* occurring in more than one literal of the conclusion of this 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 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 new instances of input clauses (in other words, not unification, but merely matching is employed). If an instance $L\sigma$ (with critical variables) of a literal occurrence L on a tableau branch is required to perform an extension or reduction step, 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.[15] Needless to say, lifting is not trivial in such a calculus.

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 proof search.

In principle, a strongly complete, *destructive* calculus could be obtained if one compiled the information contained in these "outside" clauses (and their fair selection) into a clause selection rule. Consequences for such a calculus would be: (I) clause selection is not defined branch-local, because "outside" clauses touch on several branches; (II) one needs to identify clause instances that are identical up to variable renaming to ensure finiteness; (III) a suitable ordering on literals must be used to enumerate instances of literals for closure in a fair manner. All three ingredients are actually present in recent suggestions for strongly complete, destructive proof procedures in [Beckert 1998, Beckert 2000] and [Baumgartner, Eisinger

[15]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.

and Furbach 1999, Baumgartner, Eisinger and Furbach 2000], 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. $\mathcal{M}$ is an inductively defined set of interpretations called **model candidates** each of which is represented by a consistent set of ground literals.

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

Model Extension: $\mathbf{I} \in \mathcal{M}$ is a model candidate, $D \to E$ a new 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

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

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

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, if selection functions are suitably defined on the first-order level:

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 occurring 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 of the form

$$P(x_1, \ldots, x_n) \wedge \neg P(x_1, \ldots, x_n) \to \mathit{false} \tag{4.8}$$

for each n-ary predicate symbol $P \in P_\Sigma$. The rules (4.8) were called *integrity constraints* by Shirai and Hasegawa.

Whether constraint model generation is complete for a given problem S was unclear in [Shirai and Hasegawa 1995]. From the tableau perspective, Theorem 4.22 ensures completeness for each problem S with a consistent set of rule premises (with the exception of the problem-independent rules (4.8)). The quasi-group problems discussed in [Shirai and Hasegawa 1995] are a practically relevant case.

If necessary, any clause set can be made consistent with respect to any given selection function f: assume there is a rule $R = L \wedge C \rightarrow D$ such that L is unifiable with $\overline{L'}$ occurring in the premise 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 first-order tableau calculi cannot easily be turned into a strongly complete proof procedure so that one usually retracts to DFID search, although backtracking is not necessary, in principle, within proof confluent calculi.

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

Baumgartner's [1998] idea, briefly discussed in the previous section, is to record MGUs outside of the tableau in the form of a dynamically growing input clause set. Beckert [1998] has a different approach not modifying the input clause set, but the tableaux themselves.[16]

Whenever a substitution σ must be applied to close a branch in a tableau T, the smallest subtableau T' of T affected by σ (that is, containing variables x with $\sigma(x) \neq x$) is *reconstructed*. This can be done trivially by copying T' below each open branch of $T'\sigma$ as depicted in Figure 12, 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.

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).

[16]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.

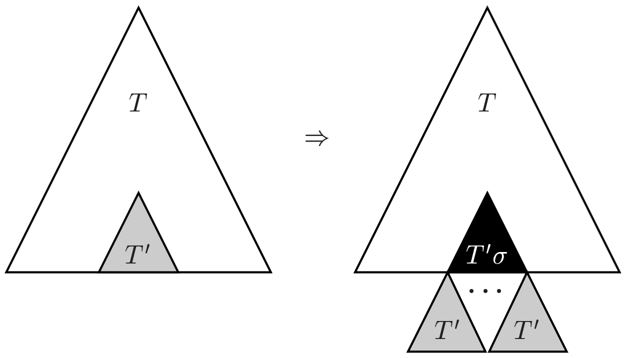


Figure 12: Reconstructing a subtableau "destroyed" by substitution.

4.28. Example. Let $|L|$ denote the number of symbols in L, where variables are counted twice. Then define a well-order $<$ by $L < L'$ 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 clause instance used for extension;
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 at least one branch of T'.

A branch B of T *subsumes* a branch B' of T' if for all sets Φ' of literals on B' with at most two elements there is a set Φ of literals on B such that

1. $\Phi\pi = \Phi'$ (where π is a variable renaming)
2. for each literal L in T that has variables in common with Φ there is an L' in T' such that $L\pi = L'$ up to renaming of free variables not occurring in Φ'.

The reason for $|\Phi| \leq 2$ is that a single rule application involves either one (extension) or two (reduction) branch literals. The second, rather technical, condition is required to guarantee that a deduction that is possible in T', can be mimicked in T, and has the same effect on instantiations and order. 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 applied to a tableau T when*

1. *the $<$-maximal elements of $R(T)$ are $<$-minimal in* $\bigcup_{R' \text{ applicable on } T} \max R'(T)$
2. *T does not subsume the result of applying R to T.*

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 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 left hand tableau in Figure 13 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 $\{x \mapsto b\}$ 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 (right hand in bottom row) is subsumed by the top right tableau: the unchanged branch is trivially subsumed; *both* gray branches subsume the same gray branch in the subsumed tableau.

This leaves closure with $\{x \mapsto a\}$ 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, only have extremely long tableau proofs. Already on the ground level there exist classes of formulas S_n such that the size of their 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

$$S_n = \{L_1 \vee \cdots \vee L_n \mid L_i \in \{P_i, \neg P_i\},\ i \in \{1, \dots, n\}\} \tag{4.9}$$

Obviously, S_n is unsatisfiable. 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.

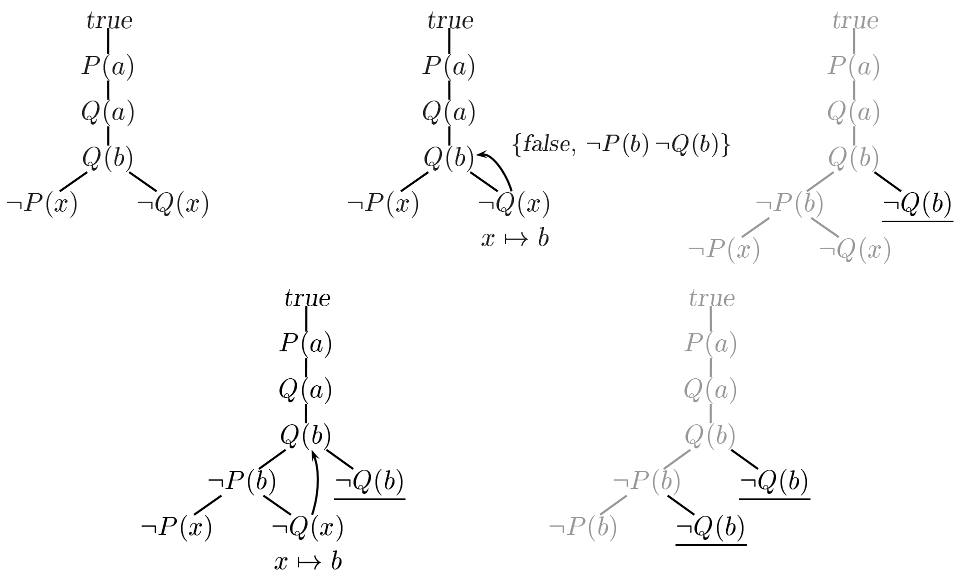


Figure 13: Illustration of Example 4.32.

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 14). 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.

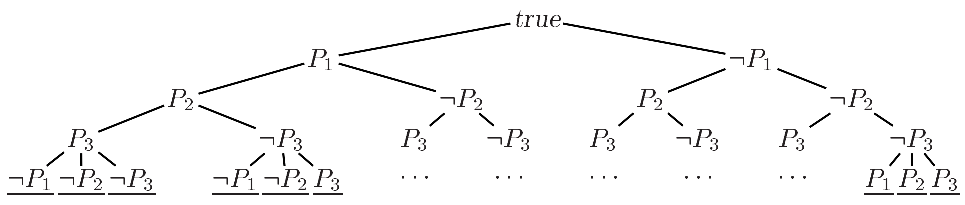


Figure 14: Clause tableau proof for $S_3 \cup T_3$.

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

$$\frac{}{P \mid \neg P} \tag{4.10}$$

called *atomic cut* rule. It is closely related to the well-known cut rule found in sequent calculi [Gentzen 1935]. This is what we look at next.

Clausal Sequent Calculus	Clausal Tableau Calculus
Axiom rule	Branch closure rule
$\vee$-left rule	Tableau extension
Atoms/unit clauses left of '$\Rightarrow$'	Positive branch literals
Atoms right of '$\Rightarrow$'	Negative branch literals
Sequent proof trees interpreted as logical conjunction of their leaf 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.

4.8.1. Tableaux and Sequent Calculi

Sequent calculi [Gentzen 1935] are direct ancestors of tableaux; see [Avron 1993, Smullyan 1995] for full accounts of their relationship, which I sketch here for the clausal case.

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 rule schemata:

$$\frac{\Gamma, L_1, \Gamma' \Rightarrow \Delta \;\Big|\; \cdots \;\Big|\; \Gamma, L_n, \Gamma' \Rightarrow \Delta}{\Gamma, L_1 \vee \cdots \vee L_n, \Gamma' \Rightarrow \Delta}\;\vee\text{-left}$$
$$\frac{\Gamma, \Gamma' \Rightarrow P, \Delta}{\Gamma, \neg P, \Gamma' \Rightarrow \Delta}\;\neg\text{-left} \qquad \frac{\times}{\Gamma, P, \Gamma' \Rightarrow \Delta, P, \Delta'}\;\text{axiom} \tag{4.11}$$

Sequent proof trees have sequents as their nodes. A sequent proof tree for a sequent $\Gamma \Rightarrow \Delta$ has this sequent as its root and is extended by applying suitable instances of rule schemata (4.11) to leaves.

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

Literal occurrences may be shared among several tableau branches, but are duplicated in sequent proof trees. In the light of this and the duality between sequent and tableau proofs, rule (4.10) corresponds exactly to the usual cut rule of sequent calculi, if the *cut formula* ϕ is restricted to being an atom:

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

$$\frac{\Gamma, \Gamma' \Rightarrow \Delta, \phi, \Delta' \;\Big|\; \Gamma, \phi, \Gamma' \Rightarrow \Delta, \Delta'}{\Gamma, \Gamma' \Rightarrow \Delta, \Delta'} \tag{4.12}$$

4.8.2. Tableaux with Lemmas

We saw that the atomic cut rule (4.10) can lead to exponentially shorter tableau proofs, however, its application is completely unrestricted—one can use it anytime during tableau construction. This may well cause longer proofs than shorter ones. It is not obvious when to apply cut advantageously. A first restriction is obtained by permitting its use only if the extension rule application immediately following it during tableau construction is a connected extension step (see Figure 15).

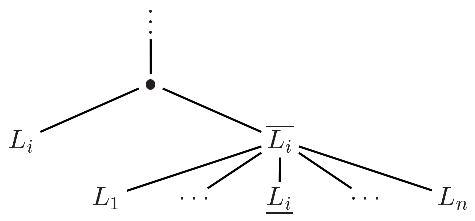


Figure 15: Cut rule application followed by connected extension step.

With this restriction in place, not more branches are generated than if one had performed only the extension step without the preceding cut. The cut has the effect that in all $n-1$ branches that contain a literal $L_j \neq L_i$ in addition the literal $\overline{L_i}$ is present. If one applies $n-1$ atomic cuts to $n-1$ different literals from $\{L_1, \dots, L_n\}$ before an extension step and writes the resulting proof tree as a "macro rule" (not displaying closed branches) yields the *extension rule with local lemmas* displayed in Figure 16.

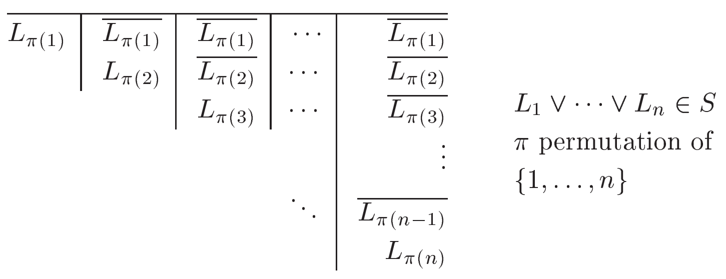


Figure 16: Extension rule with local lemmas.

The complemented literals are called *local lemmas*: for example, $\overline{L_{\pi(i)}}$ is obtained as a lemma on its sibling branches after the branch B is closed with the help of

$L_{\pi(i)}$. (Recall that closed branches are unsatisfiable, hence, a branch that can be closed and where $L_{\pi(i)}$ occurs, proves $B \models \overline{L_{\pi(i)}}$.) 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 [Letz 1993]. 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 17. Assuming that *select branch* is left-first, after closure of branch $B = \{P, Q, R\}$ one knows that $S \cup \{P, Q\} \models \neg R$. This lemma is useless, though, because the branch $\{P, Q, \neg Q\}$ to the left of B is closed already, and B has no right sibling. Inspection of the partial proof shows, however, that even $S \cup \{P\} \models \neg R$ holds, because Q was not involved in the closure of B.

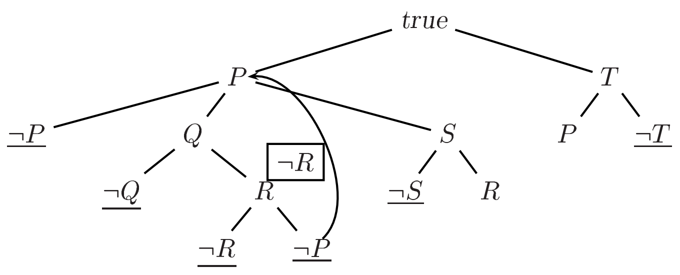


Figure 17: Illustration of Example 4.34.

In clause tableaux with *folding up rule* each literal L that is a local lemma in the sense of the local lemma rule in Figure 16 may be moved up towards the root as follows: we say that a literal L' on the path between L and the root was *used to prove* L if L' is involved in the closure of a branch through L'; now the folding up rule permits to move L immediately above the lowest literal used to proof L [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 above P and lemma $\neg P$ from the right subtree is moved up above *true* (the latter is also possible with a suitable variant of the local lemma rule). In Figure 18 new lemma positions are boxed, upward moves are indicated by dashed arrows, while additional closures made possible by moved lemmas are indicated by solid arrows.

One can show that the folding up rule does not change the *nondeterministic power* of tableau with local lemmas, that is, the length of shortest proofs does not

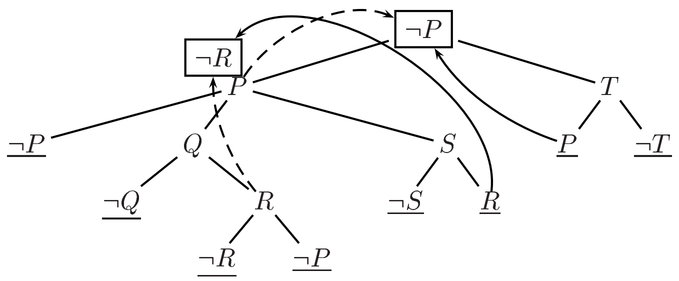


Figure 18: Tableau with folding up rule.

change [Letz 1993]. In a concrete tableau proof search the shortest proof is generally not found, therefore, folding up can speed up proof search considerably: an ill choice of *select clause* might 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]. Assume branch B is not yet closed, but there is a "more specific" branch B' closed already. Intuitively, this justifies to close B as well (as long as one does not go around in circles). Formally, a branch B' is *more specific* than a branch B iff there is a substitution σ such that $B' \subseteq B\sigma$. 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 directly; alternatively, 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 19 the left branch was closed first by extension with $P(a)$ (only $P(b)$ would lead to success). This forces extension with $\neg Q(a)$ 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.

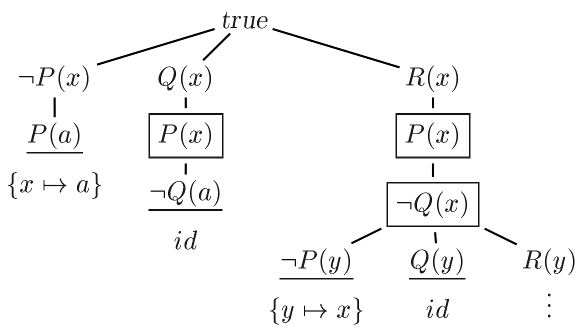


Figure 19: Search space increase caused by local lemmas.

On the other hand, local lemmas are not strong enough on the first-order level. In the clause tableau for $S = \{Q(c) \vee Q(d),\ \neg Q(x) \vee P,\ \neg P\}$ in Figure 20, for example, the lemma $\neg Q(c)$ can be folded up to the *true* node, but it cannot be used to close the open branch on the right, because a different instance than c is required. Closer inspection of S, however, shows that it is in fact justified to derive the stronger lemma $(\forall x)\neg Q(x)$. A sufficient condition for lemma generalization is reached when the proof of (that is, the tableau below) the node giving rise to the lemma does not instantiate any variables that occur elsewhere in the tableau. It remains to be seen whether such an optimization can be efficiently implemented and whether it 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 for automated deduction can be uniformly presented within a tableau framework and that this deepens the understanding of these calculi. In Section 4.6 I put model generation into a tableau perspective. The relationship between tableaux and the connection method [Bibel 1982*b*] is discussed in Section 3.5 above for the general, non-clausal case. Therefore, these calculi are not discussed again in the following.

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 an occurrence of $\neg P$ is required. The type B literal P turns into a type A literal 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 block 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.

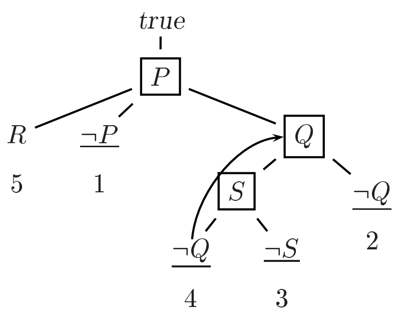


Figure 21: Partial regular connection tableau corresponding to the model elimination proof in Example 5.4.

Model elimination is closely linked to the connection tableau calculus. This is demonstrated by the tableau in Figure 21 which simulates lines 0 to 5 in the model elimination proof above. The number below each branch indicates which line in the model elimination proof corresponds to its closure. Type A literals correspond to inner tableau nodes. Type B literals occurring between two type A literals in chains correspond to 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 changes the focus from closed branches without open siblings to the next open branch to the left. Hence, a chain is simply a linear format to represent a partial connection tableau. This is possible, because *select branch* in model elimination 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 necessary number of extensions 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 connection tableaux, of course.

Model elimination can be interpreted as a systematic way to exclude certain interpretations as possible models of a clause set [Loveland 1969, p. 362], but it should be stressed that those type A literals in a non-empty chain to which no rule can be applied in general do *not* constitute a partial model (or have a saturation, in the terminology of Section 4.5), because model elimination, just like connection tableaux, is not proof confluent. To summarize, model elimination (without the lemma generation mechanism) can be conceived as a notational variant of regular connection tableaux (although it preceded the latter, of course).

5.2. Linear Resolution

Linear resolution [Loveland 1968*a*, Luckham 1968] and its refinements do not fit directly into the framework of standard resolution [Robinson 1965], because their definition, like that of tableau calculi with connection condition, relies on the form of the derivation. Moreover, in first-order SL-resolution [Kowalski and Kuehner 1971, Reiter 1971] (discussed on page 69 in Chapter 2 of this Handbook) variables are treated rigidly like in tableaux with unification. Indeed, as Loveland [1972] observed, SL-resolution 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 connection tableaux.

More formally, let S be an unsatisfiable set of Horn clauses. Corollary 4.15 guarantees the existence of a connection tableau proof T for S starting with a *query* Q. Regularity is not enforced. By a trivial induction on the depth of T, using that S is Horn and connectedness of T, one obtains that the positive literals of T are exactly the literals at leaf nodes. As a consequence, 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 page 128 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 resolution* refutation of $\{Q\} \cup (S - \{Q\})$ in the usual sense [Lloyd 1987]. This remains true on the first-order level.

5.5. EXAMPLE. To approximate the usual notation of logic programming, we employ rule notation of clauses as introduced in Section 4.6.[19]

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

One of several possible connection tableaux for S starting with the only query $P(x) \to \mathit{false}$ in S is displayed on the left in Figure 22.

The open branches are the ones ending with $\neg T(z)\{z \mapsto c\} = \neg T(c)$ 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 connected extension steps are made). Thus a less redundant notation for connected Horn clause tableaux 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 in Figure 22 on the right: each node represents a 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 $P(x) \to \mathit{false}$ in common logic programming notation [Lloyd 1987]. Another valid point of view is to interpret the figure as a model elimination proof with left-first branch selection, where only the rightmost block of type B literals in a model elimination chain is represented.

In summary, although the Prolog procedure is often presented as a resolution refinement, it can be completely and naturally fitted into the connection tableau

[19]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 $P \to \mathit{false}$.

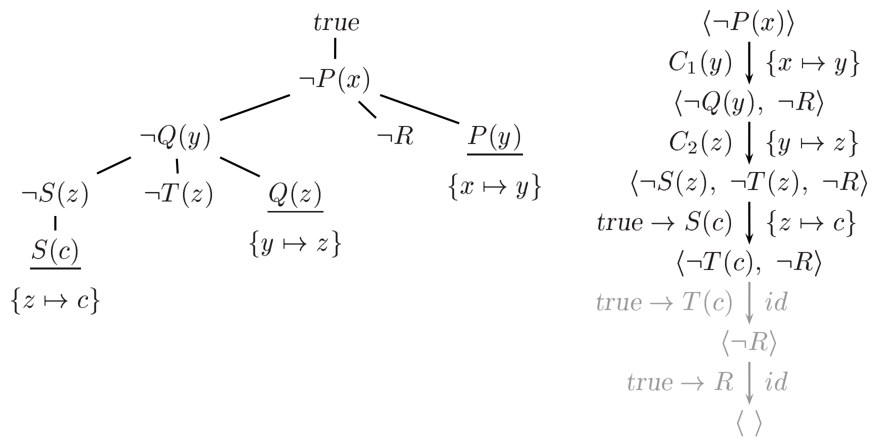


Figure 22: Illustration of Example 5.5.

(and hence: model elimination) framework. In the remainder of this section I intend to show that it is even advantageous to do so.

The following terminology comes handy: given a clause set S in rule notation, consider a clause tableau T for S. An occurrence of a literal L in T that was introduced in an extension step using rule $C \in S$ such that L occurs in the premise or body of C is called a *body literal*. Similarly, literal occurrences in T that stem from conclusions or heads of rules in S are called *head literal*. Further, in a reduction step in a clause tableau that involves L as the leaf literal, we say that the reduction *is from* L. Finally, the literal $L \in C$ in a (weakly) connected extension step using clause C in branch B such that $\overline{L}$ occurs on B is referred to with the phrase *extension via* L.

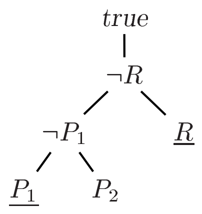
Disjunctive logic programming, instances of which are *Non-Horn clause logic programming* [Plaisted 1988] and *Near-Horn logic programming* [Loveland 1987], is the attempt to lift the Horn restriction from Prolog while retaining the efficient implementability of Prolog-SLD resolution. One line of development is to implement model elimination (that is: regular connection tableaux) for arbitrary clauses directly via an extended Prolog Abstract Machine [Letz et al. 1992]. This is discussed in detail in [Letz and Stenz 2001] (Chapter 28 of this Handbook). 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, and in others more general than connection tableaux and model elimination.[20] We start our discussion by two examples that highlight some differences.

[20]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].

5.6. Example. Consider $S_1 = \{\mathit{true} \to P \vee Q,\ P \wedge Q \to \mathit{false},\ P \to Q,\ Q \to P\}$. In the Prolog-SLD restriction of connection tableau no reduction steps are allowed. Obviously, no 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 example 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 $S_2 = \{P_1 \to R,\ P_2 \to R,\ R \to \mathit{false},\ \mathit{true} \to P_1 \vee P_2\}$. Below is a partial connection tableau proof for S_2:



In connection tableaux the open branch ending in P_2 can be extended with $P_2 \to R$ which completes the proof with one more extension using $R \to \mathit{false}$ (the case when $P_2 \to R$ is used before $P_1 \to R$, 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 is not represented. It is, of course, possible to extend the linear proof format to accommodate head literals (see [Reed and Loveland 1992*a*] and Example 5.8), but I prefer to use the clause tableau format, where all required information is present and subgoal dependencies are easy to see. Some authors prefer to use a sequent-style format [Plaisted 1990, Reed and Loveland 1992*b*].

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

$$\{\overline{L_1} \wedge \cdots \overline{L_{i-1}} \wedge \overline{L_{i+1}} \wedge \cdots \wedge \overline{L_n} \to L_i \mid 1 \leq i \leq n\} \ .$$

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.

For the rest of Section 5.3 we assume that clause sets S are given in rule notation and *no other contrapositives than the explicitly given rules can be used in tableau extension steps.*

How does one render 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.[21] It does not come as a surprise then, that the remedy is similar: certain *restart steps*, that is, extension steps without a connection are permitted to regain completeness.

A general framework of clause tableau calculi for disjunctive logic programming is, therefore, given by the following coordinates:

1. the first extension step is with a query, see Corollary 4.15;
2. connected extensions steps, generalizing the Prolog-SLD resolution rule, are permitted;
3. a list of permitted contrapositives of clauses is either explicitly given in rule notation or exactly the positive literals are head literals and the negative literals are body literals;
4. reduction steps (called *ancestor cancellation* rule by Reed and Loveland [1992*b*] who derive the name from ancestor resolution) are only allowed from body literals;
5. restart extension steps are only allowed below head literals.

The last two items constitute the parameterizable part. Disjunctive logic programming calculi are not proof confluent as the example $S = \{\mathit{true} \rightarrow C, C \rightarrow A, B \rightarrow A, A \rightarrow \mathit{false}\}$ shows: any tableau starting with the last clause, followed by a extension step with the second but 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 four ingredients are classified as the *ancestry family* of deductive procedures by Reed and Loveland [1992*b*]. In the remainder of the section I define some of the more 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, see 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:

[21] Tableaux with selection function select the *body* literals of a clause, but Prolog proceeds "top-down" from the query and extension steps are via head literals.

each head literal above a restart step must be used in a reduction step with no restart step in between.

5.8. Example. As its name suggests, near-Horn Prologs are generalizations of Prolog-SLD resolution. We stress this by rewriting Example 5.7 in a Prolog-like notation [Reed and Loveland 1992*b*]. The separator '#' is followed by a list of active heads available for reduction: the head literals in the current tableau branch (only the most recently introduced head literal, in the case of UnH-Prolog). Next is the list of deferred subgoals (leaves of open branches), surrounded by curly brackets. The tableau in the Example 5.7 corresponds to the following derivation, which is both an UnH- and an InH-Prolog derivation:

$$\begin{array}{llll} ?\text{-} & \mathit{false} & \# & \\ ?\text{-} & \mathrm{R} & \# & \\ ?\text{-} & \mathrm{P}_1 & \# & \\ ?\text{-} & & \# & \{\mathrm{P}_2\} \end{array}$$

In UnH-Prolog one can continue with a restart step at R and the only deferred subgoal P_2:

$$\begin{array}{llll} ?\text{-} & \mathrm{R} & \# & \mathrm{P}_2 \\ ?\text{-} & \mathrm{P}_2 & \# & \mathrm{P}_2 \\ ?\text{-} & & \# & \mathrm{P}_2 \end{array}$$

This time, clause $P_2 \to R$ is used, which makes reduction with P_2 possible. As there are no more deferred subgoals, the proof is finished.

In InH-Prolog only a restart step at *false* is possible after the first block, 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, hence, of the clausal version of the connection method (Section 3.5), even without using contrapositives of clauses provided that the input is in goal normal form. The reason is simply that the query clause of a restart step in an InH-Prolog proof is weakly connected to the root literal in this case. Therefore, an InH proof (which does not use contrapositives) is at the same time a weakly connected clause tableau proof.

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 to occur below a head literal. Up to this feature and the cancellation pruning rule (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 as generalizations of Horn clause logic programming, the motivation of *restart model elimination* for Baumgartner and Furbach [1994] was to obtain a specialized version of connection tableaux/model elimination that is complete without contrapositives.

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.

Within the restart model elimination framework the restriction of reduction steps to be only from body literals employed in InH-Prolog is called *strict restart model elimination* in [Baumgartner and Furbach 1994]. This leaves the cancellation pruning rule and left-first branch selection strategy as the only remaining difference between InH-Prolog and strict restart model elimination. Left-first branch selection is standard in PTTP technology-based implementations such as the one reported in [Baumgartner and Furbach 1994]. The gap is closed by [Baumgartner and Furbach 1998] where strict restart model elimination with cancellation pruning is considered.

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. The following refinements were investigated:

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 on the same branch with the same literal are admissible to maintain completeness, but in between subsequent restarts regularity can be enforced.

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

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].

5.3.3. Restart Tableaux

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

Restart tableaux [Pape and Hähnle 1997] can be motivated from the observation that the combination of selection functions and enforcing the weak connection condition for every extension step via a selected literal in general results in an incomplete calculus (Section 4.5.3). Adding restart steps in the sense of Section 4.5.2 gives completeness back. Restart tableaux restrict their restart steps to occur only in situations when no weakly connected extension step via a selected literal is possible, but, in contrast to near-Horn Prolog and restart model elimination, these restart steps may also occur below body literals. This relaxation makes restart tableaux a *proof confluent* calculus.

Restart tableaux admit to select an arbitrary subset S_c of the input clause set S such that only connected extension steps can be performed with clauses from S_c. Depending on the degree of "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 is a generalization of strict restart model elimination.

5.4. Davis-Putnam Procedure and related methods

Clause tableaux with the cut rule (4.10) 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 the presence of cut can blow up the search space considerably. Mondadori [1988] designed a tableau system with cut called *KE* for full first-order logic, where every rule, including the cut rule, is essential for completeness. Together with D'Agostino, KE 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 is a close variant of weak connection tableaux (Section 4.3.2) and can be succinctly described as follows: take the atomic cut rule (4.10), the initialization rule of clause tableaux (Definition 4.1(i)), and these extension rules (compare with (ii$''$) on page 131) are:

(ii$''''$) Let T be a KE-tableau for S, B a branch of T containing L, $L_1 \vee \cdots \vee L_r$ *is on* B *or* a new 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 $\min\{2, r\}$ new subtrees, where the nodes of the new subtrees are L_i and, if

$r \geq 2$, $L_1 \vee \cdots \vee L_{i-1} \vee L_{i+1} \vee \cdots \vee L_r$, then $T'\sigma$ is a KE-tableau for S, in which the branch ending with $L_i\sigma$ is marked as closed. (This is called a *KE extension step.*)

(iii'''') Let T be a tableau for S, B a branch of T, L a new instance of a unit clause in S. Then the tableau constructed from T by extending B with L is a KE-tableau for S. (This is called a *unit extension step.*)

Formally, the resulting calculus is not an instance of the clause tableau framework, but of non-clausal tableaux, because nodes may be clause instances (not just literals). Application of the extension rules of KE-tableaux does never increase the number of open branches, so the system is incomplete without the atomic cut rule, which is called *principle of bivalence* by Mondadori [1988].

Rule (ii'''') is better known under the name *unit resolution*. Its propositional version originates in [Davis and Putnam 1960], where it is called *elimination of one-literal clauses*. The *affirmative-negative* rule in [Davis and Putnam 1960], better known as *pure clause rule* avoids all extensions with clauses that contain an unconnected literal; it is only partly realized in 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-Logeman procedure* [Davis and Putnam 1960, Davis et al. 1962] are very similar and closely related to weak connection 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 non-clausal KE system as a non-clausal version of the Davis-Putnam-Loveland procedure (up to the pure clause rule). Within the non-clausal tableau framework with *analytic cut* (the cut formula can be 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:

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

Mondadori [1989*b*] also suggested a system called *KI* consisting of "inverted" tableau rules that build up complex formulas from simpler ones. At the propositional level they are:

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

These rules are restricted to analytic use in the sense that their 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), or alternatively, they form yet another complete system together with *atomic cut*.

Another tableau calculus and relative of KE is *Stålmarck's procedure* [Sheeran and Stålmarck 2000], which is underlying a commercially successful satisfiability checker for propositional logic, the *Prover Plug-In*. The calculus consists of (non-branching) KE-like rules for a restricted propositional language, namely, formulas of the form $P \leftrightarrow (Q \rightarrow R)$, where P, Q, R are atoms or logical constants. They are called *triplets* and form the tableau nodes. In addition, there is the so-called *dilemma rule*, a generalization of atomic cut: after applying the cut, each branch is saturated with respect to the non-branching rules; if neither branch is closed or gives a model, then both branches are recombined by computing the intersection of their triplets. The nesting depth of dilemma rule applications can be limited to a pre-set bound.

6. Comparing Calculi

In this section I suggest some coordinates along which the plethora of existing tableau-like calculi (and others) can be cartographed. It is kept short, because most of the material is not specific to tableau-like calculi.

A useful qualitative distinction are the properties *proof confluence* and *branching*. The latter encompasses calculi, where the derivation process branches or forks, see also [Bachmair and Ganzinger 2001] (Chapter 2 in this Handbook). In Table 4 are typical representatives of each kind of calculus.

	branching	non-branching
proof confluent	tableaux with unification	Robinson resolution
not proof confluent	connection tableaux	Prolog SLD-resolution

Table 4: Qualitative classification of calculi.

Calculi that are not proof confluent cannot be used to find models of satisfiable formulas. They make no sense to use with propositional formulas either, because backtracking is expensive and can easily be avoided in the propositional case. Implementation of branching calculi demands more general data structures than of non-branching ones. In practice, implementation-oriented branching calculi are often formulated sequentially, for example, model elimination. Another property that severely affects the choice implementation techniques is destructiveness. For example, it is not straightforward to build strong redundancy criteria such as subsumption or regularity into destructive calculi, see Section 4.4.

Deeper insight into similarities and differences among calculi is often gained by presenting them within a unifying theoretical framework, such as the tableau framework of the present chapter. The point of view of matrices is taken in

[Bibel 1982*b*, Bibel and Eder 1992]; *consolution* combines aspects from resolution and the connection method and can be used to analyze a broad class of calculi [Eder 1992, Baumgartner and Furbach 1993].

Another objective criterion for evaluation of calculi is provided by an investigation into their complexity.

First of all, one can determine the complexity of the decision problem of various subtasks associated with (tableau) proof search. It is well-known that clause subsumption is an NP-complete problem [Garey and Johnson 1979]; the same is true, for example, for computing hyper tableau extension steps in first-order logic (see Section 4.6.2) or for solving constraints arising in ordered tableaux (see Section 4.5.4) from certain literal orders [Pape 1996].

A more basic problem is encountered by the realization that tableau-like methods provide a constructive proof of Herbrand's Theorem 2.3. A tableau T can be considered as a formula ϕ_T, where branches form conjunctions over their literals and ϕ_T is a disjunction over the branches of T. Let $x_1, \ldots, x_r$ be the free variables of ϕ_T. By tableau soundness (Theorem 3.12), T can be closed iff $(\forall x_1, \ldots, x_r)\phi_T$ is unsatisfiable and has Herbrand complexity $m = 1$. The problem, whether the sentence in Herbrand's Theorem is unsatisfiable with multiplicity $m = 1$, is called the *formula instantiation problem* in [Voronkov 1998]; it is shown to be decidable and Σ_2^p-complete, if the signature of ϕ_T has at least two symbols.

A completion mode used in tableau-like proof procedures (see Section 3.3) can be seen as a strategy to approximate the required multiplicity of a formula to show its unsatisfiability with Herbrand's Theorem. In [Voronkov 1998] such strategies are discussed and analyzed in an abstract framework. All of the mentioned problems become substantially more complex, if part of the formula signature is interpreted relative to theories (such as equality), see [Degtyarev and Voronkov 2001*a*] (Chapter 10 in this Handbook).

It is important to know the complexity of each part of a proof search procedure, but one would also like to have results about the relative overall performance of various refinements. One criterion that received much attention is *minimal proof length*, the size of a minimal proof for a formula ϕ, based on a suitable notion of proof length. Our definition of tableau size is an adequate measure. For propositional resolution, one can take the sum of the sizes of generated clauses (it was shown in [Letz 1993] that there can be no adequate measure for first-order resolution). Cook and Reckhow [1979] suggested to compare calculi based on the relative length of minimal proofs:

6.1. Definition. Let $\mathcal{P}$ and $\mathcal{Q}$ be sound and complete calculi. $\mathcal{P}$ can *p-simulate* $\mathcal{Q}$ iff there is a polynomial p such that for all formula sets Φ: if there is a proof for Φ in $\mathcal{Q}$ of length n, then there is a proof of Φ in $\mathcal{P}$ of at most length $p(n)$. $\mathcal{P}$ and $\mathcal{Q}$ are *p-equivalent* iff they can p-simulate each other.

There is a large number of results on p-simulation for tableau and other calculi, for example, [Eder 1992, Letz 1993]. Because of technical difficulties, the vast majority are for the propositional, clausal case, but there are exceptions such as

[Baaz and Fermüller 1995, Egly 1997] showing that non-elementary differences in complexity exist on the first-order level for even closely related calculi. There are some surprises, for example, clause tableau cannot p-simulate truth table checking (see Example 4.33 and [D'Agostino 1992]) or n-ary type β tableau rules cannot p-simulate binary ones (see Section 3.2.6 and [Massacci 1998*a*]). Regular connection tableaux (and, hence model elimination) are pretty weak calculi in this hierarchy. For a number of reasons, relative proof length complexity is of limited use to predict the practical behaviour of calculi:

- relative proof length complexity measures the *nondeterministic* power of calculi, nothing is said about how long it takes to *find* a proof; for this, the search space size would be a more suitable measure;
- the stronger the nondeterministic power of a calculus, the larger is often the search space (witness tableaux with lemmas, Section 4.8.5);
- the notion of p-simulability is often too coarse, for example, all variants of tableaux with lemmas are p-equivalent, regardless of regularity or connectedness;
- the behaviour of calculi on the first-order level can easily override any differences on the propositional level.

An attempt to formalize the search space associated with calculi is made by Plaisted and Zhu [1997]. The search space is formalized as a directed graph whose nodes represent states in a proof and arcs define reachability among these states. Plaisted and Zhu [1997] analyze width, depth and size of nodes of search space graphs for a number of calculi and propositional Horn clauses. Satisfiability of the latter is well known to be decidable in linear time [Dowling and Gallier 1984], but many calculi turn out to be exponential along at least one dimension of their search space. As the Horn fragment is practically important, one can argue that the performance of a calculus there is significant also for its overall behaviour to a certain extent.

Apart from theoretical investigations one can and does perform experimental evaluation based on concrete implementations. There is an extensive problem collection (the TPTP library) for first-order theorem provers [Suttner and Sutcliffe 1999] and an annual competition [Sutcliffe and Suttner 1999]. This is a good way to test correctness and base speed of the implementation of an automatic theorem prover, but it does not necessarily tell everything on the usability of a system. The latter is perhaps best evaluated within larger case studies or applications. It is unlikely that the same deduction paradigm suits all needs. In practice, often a combination of several techniques is successful: see [Weidenbach 2001] (Chapter 27) of this Handbook for a point in case. There is also indication that the usefulness of fully automated systems is limited, whereas they have a large potential in connection with interactive systems [Joyce and Seger 1993, Ahrendt et al. 1998, Dahn and Schumann 1998].

7. 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 much 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. 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 [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,[22] where many of the relevant results are now published. Part I of [Bibel and Schmitt 1998] contains survey articles on various aspects of state-of-the-art research on tableau methods. The Handbook of Tableau Methods [D'Agostino, Gabbay, Hähnle and Posegga 1999] contains extensive information, not limited to automated reasoning.

Acknowledgments

Uli Furbach and Philippe de Groote served as second readers of this chapter. Their careful reading and constructive criticism resulted in numerous large and small improvements. In addition, Andrei Voronkov proofread most of this chapter and provided many corrections and useful suggestions. Thanks are also due to Bernhard Beckert, Jean Goubault-Larrecq, Ryuzo Hasegawa, and Reinhold Letz for discussions, helpful observations, and suggestions. Reinhold Letz contributed Example 4.9. Jeff Lansing corrected a number of errors in my English.

[22] `i12www.ira.uka.de/TABLEAUX/`

Bibliography

Ahrendt W., Beckert B., Hähnle R., Menzel W., Reif W., Schellhorn G. and Schmitt P. H. [1998], Integration of automated and interactive theorem proving, *in* W. Bibel and P. Schmitt, eds, 'Automated Deduction: A Basis for Applications', Vol. II, Kluwer, chapter 4, pp. 97–116.

Andrews P. [1971], 'Resolution in type theory', *Journal of Symbolic Logic* **36**, 414–432.

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

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

Avron A. [1993], 'Gentzen-type systems, resolution and tableaux', *Journal of Automated Reasoning* **10**(2), 265–281.

Baader F. and Snyder W. [2001], Unification theory, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 8, pp. 445–532.

Baaz M., Egly U. and Leitsch A. [2001], Normal form transformations, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 5, pp. 273–333.

Baaz M. and Fermüller C. G. [1995], Nonelementary speedups between different versions of tableaux, *in* P. Baumgartner, R. Hähnle and J. Posegga, eds, 'Proc. 4th Workshop on Deduction with Tableaux and Related Methods, St. Goar, Germany', Vol. 918 of *LNCS*, Springer-Verlag, pp. 217–230.

Baaz M., Fermüller C. and Salzer G. [2001], Automated deduction for many-valued logics, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 20, pp. 1355–1402.

Bachmair L. and Ganzinger H. [2001], Resolution theorem proving, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.

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], A confluent connection calculus, *in* H. Ganzinger, ed., 'Proc. 16th International Conference on Automated Deduction, CADE-16, Trento, Italy', Vol. 1632 of *LNCS*, Springer-Verlag, pp. 329–343.

Baumgartner P., Eisinger N. and Furbach U. [2000], A confluent connection calculus, *in* S. Hölldobler, ed., 'Intellectics and Computational Logic — Papers in Honor of Wolfgang Bibel', Kluwer.

Baumgartner P. and Furbach U. [1993], 'Consolution as a framework for comparing calculi', *Journal of Symbolic Computation* **16**, 445–477.

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.

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.

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

Beckert B. [2000], Depth-first proof search without backtracking for free-variable clausal tableaux, *in* P. Baumgartner and H. Zhang, eds, 'Proc. Third Int. Workshop on First-Order Theorem Proving (FTP), St. Andrews, Scotland', Fachberichte Informatik 5/2000, University of Koblenz, Institute for Computer Science, pp. 44–55.

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., Hähnle R., Oel P. and Sulzmann M. [1996], The tableau-based theorem prover $_3T^AP$, version 4.0, *in* M. McRobbie and J. Slaney, eds, 'Proc. 13th Conference on Automated Deduction, New Brunswick/NJ, USA', Vol. 1104 of *LNCS*, Springer-Verlag, pp. 303–307.

Beckert B., Hähnle R. and Schmitt P. H. [1993], The *even more* liberalized δ-rule in free variable semantic tableaux, *in* G. Gottlob, A. Leitsch and D. Mundici, eds, 'Proceedings of the third Kurt Gödel Colloquium KGC'93, Brno, Czech Republic', Vol. 713 of *LNCS*, Springer-Verlag, pp. 108–119.

Beckert B. and Posegga J. [1995], 'leanT^AP: 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. [1979], 'Tautology testing with a generalized matrix method', *Theoretical Computer Science* **8**, 31–44.

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**(3), 269–293.

Bibel W. [1982*c*], Computationally improved versions of herbrand's theorem, *in* 'Logic Colloquium '81', North-Holland, pp. 11–28.

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 Eder E. [1992], Methods and calculi for deduction, *in* D. M. Gabbay, C. J. Hogger and J. A. Robinson, eds, 'Handbook of Logic in Artificial Intelligence and Logic Programming', Vol. 1: Logical Foundations, Oxford University Press, pp. 67–182.

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 Torge S. [1998], A deduction method complete for refutation and finite satisfiability, *in* J. Dix, L. F. del Cerro and U. Furbach, eds, 'Proc. 6th European Workshop on Logics in AI (JELIA)', Vol. 1489 of *LNCS*, Springer-Verlag, pp. 122–136.

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.

Cantone D. and Nicolosi Asmundo M. [1998], A further and effective liberalization of the delta-rule in free variable semantic tableaux, *in* R. Caferra and G. Salzer, eds, 'Proc. Second Int. Workshop on First-Order Theorem Proving, FTP'98', Technical Report E1852-GS-981, Technische Universität, Wien (Austria), pp. 86–96.

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.

Dahn I. and Schumann J. [1998], Using automated theorem provers in verification of protocols, *in* W. Bibel and P. Schmitt, eds, 'Automated Deduction: A Basis for Applications', Vol. III, Kluwer, chapter 8, pp. 195–224.

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.

Degtyarev A. and Voronkov A. [1998], 'What you always wanted to know about rigid E-unification', *Journal of Automated Reasoning* **20**(1), 47–80.

Degtyarev A. and Voronkov A. [2001], Equality reasoning in sequent-based calculi, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 10, pp. 609–704.

Dowling W. and Gallier J. [1984], 'Linear-time algorithms for testing the satisfiability of propositional Horn formulæ', *Journal of Logic Programming* **3**, 267–284.

Eder E. [1992], *Relative Complexities of First-Order Calculi*, Artificial Intelligence, Vieweg Verlag.

Egly U. [1997], Non-elementary speed-ups in proof length by different variants of classical analytic calculi, *in* D. Galmiche, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 158–172.

Fitting M. [1999], Introduction, *in* M. D'Agostino, D. Gabbay, R. Hähnle and J. Posegga, eds, 'Handbook of Tableau Methods', Kluwer, Dordrecht, pp. 1–43.

Fitting M. C. [1990], *First-Order Logic and Automated Theorem Proving*, Springer-Verlag, New York.

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].

Giese M. [2000], Proof search without backtracking using instance streams, position paper, *in* P. Baumgartner and H. Zhang, eds, 'Proc. Third Int. Workshop on First-Order Theorem Proving, St. Andrews, Scotland', Fachberichte Informatik 5/2000, University of Koblenz, Institute for Computer Science, pp. 227–228.

Giese M. and Ahrendt W. [1998], Hilbert's ϵ-terms in automated theorem proving, *in* N. V. Murray, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Saratoga Springs/NY, USA', number 1617 *in* 'LNCS', Springer-Verlag, pp. 171–185.

Hähnle R. [1999], Tableaux for many-valued logics, *in* M. D'Agostino, D. Gabbay, R. Hähnle and J. Posegga, eds, 'Handbook of Tableau Methods', Kluwer, Dordrecht, pp. 529–580.

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. Int. 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.

Herbrand J. [1930], Recherches sur la théorie de la démonstration, Thèse de doctorat, Université de Paris, France. Reprinted in [Herbrand 1971].

Herbrand J. [1971], *Jacques Herbrand: Logical Writings, edited by W. Goldfarb*, Reidel, Dordrecht.

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

Joyce J. J. and Seger C.-J. H. [1993], Linking BDD-based symbolic evaluation to interactive theorem-proving, Technical Report TR-93-18, UBC.

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, *in* N. Murray, ed., 'Position Papers, International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Saratoga Springs, NY, USA', Technical Report 99-1, Institute for Programming and Logics, Department of Computer Science, University at Albany – SUNY, pp. 101–116.

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* C. Kirchner and H. Kirchner, eds, '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.

Letz R. and Stenz G. [2001], Model elimination and connection tableau procedures, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 28, pp. 2015–2114.

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* E. Lusk and R. Overbeek, eds, '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.

Nonnengart A. and Weidenbach C. [2001], Computing small clause normal forms, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 6, pp. 335–367.

Ohta Y., Inoue K. and Hasegawa R. [1998], On the relationship between non-Horn magic sets and relevancy testing, *in* C. Kirchner and H. Kirchner, eds, '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.

PLAISTED D. A. AND ZHU Y. [1997], *The Efficiency of Theorem Proving Strategies: A Comparative and Asymptotic Analysis*, Vieweg Verlag, Braunschweig.

POSEGGA J. [1993], Deduktion mit Shannongraphen für Prädikatenlogik erster Stufe (in German), 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.

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.

SHEERAN M. AND STÅLMARCK G. [2000], 'A tutorial on Stålmarck's proof procedure for propositional logic', *Formal Methods in Systems Design* **16**(1), 23–58.

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. [1963], 'A unifying principle in quantification theory', *Procesdings of the National Academy of Sciences of the U.S.A.* **49**(6), 828–832.

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. [1999], 'The CADE-15 ATP system competition', *Journal of Automated Reasoning* **23**(1), 1–23.

Suttner C. B. and Sutcliffe G. [1999], The TPTP problem library, TPTP v2.2.0, Technical Report JCU-CS-99/02, Department of Computer Science, James Cook University.

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

Voronkov A. [1996], Proof search in intuitionistic logic based on constraint satisfaction, *in* P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi, eds, 'Proc. 5th International Workshop on Theorem Proving with analytic Tableaux and Related Methods (TABLEAUX), Terrassini, Italy', Vol. 1071 of *LNCS*, Springer-Verlag, pp. 312–329.

Voronkov A. [1998], Herbrand's theorem, automated reasoning and semantic tableaux, *in* 'Proc. 13th IEEE Symposium on Logic in Computer Science, LICS, Indianapolis, USA', IEEE Press, Los Alamitos, pp. 252–263.

Waaler A. [2001], Connections in nonclassical logics, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 22, pp. 1487–1578.

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.

Weidenbach C. [2001], Combining superposition, sorts and splitting, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 27, pp. 1965–2013.

Notation

Index