*Journal of Universal Computer Science, vol. 5, no. 3 (1999), 113-134*
*submitted: 1/11/98, accepted: 12/3/99, appeared: 28/3/99 © Springer Pub. Co.*

# Proof Transformations from Search-oriented into Interaction-oriented Tableau Calculi

Gernot Stenz
(Munich University of Technology, Germany
stenzg@in.tum.de)

Wolfgang Ahrendt
(University of Karlsruhe, Germany
ahrendt@ira.uka.de)

Bernhard Beckert
(University of Karlsruhe, Germany
beckert@ira.uka.de)

**Abstract:** Logic calculi, and Gentzen-type calculi in particular, can be categorised into two types: search-oriented and interaction-oriented calculi. Both these types have certain inherent characteristics stemming from the purpose for which they are designed. In this paper, we give a general characterisation of the two types and present two calculi that are typical representatives of their respective class. We introduce a method for transforming proofs in the search-oriented calculus into proofs in the interaction-oriented calculus, and we demonstrate that the difficulties arising with devising such a transformation do not pertain to the specific calculi we have chosen as examples but are general. We also give examples for the application of our transformation procedure.

**Key Words:** Automated deduction, tableau calculi, proof transformation, proof presentation, proof search.
**Category:** I.2.3

## 1 Introduction

Historically, tableaux and other Gentzen-type calculi were developed as proof theoretic tools. Since then they have been improved in many ways and for different purposes, in particular proof search and automated deduction on the one hand and proof presentation and user interaction on the other hand.

In this paper we describe and discuss the two classes of calculi in general [Section 2] and present a proof transformation from a search-oriented calculus into an interaction-oriented calculus—both calculi are typical representatives of their classes. The two calculi—a Smullyan-style ground tableau calculus augmented by a non-analytic cut rule and a free variable tableau calculus using the universal variable technique—are defined in [Sections 3 and 4], respectively. In [Sections 5 and 6] we describe the transformation and give examples, with a particular focus on the translation of (multiple) instantiations of free variables. Our transformation has been implemented in Prolog [Stenz, 97] as part of the theorem prover $_3T\!AP$ [Beckert *et al.*, 96].

The main application of transformations between search-oriented and interaction-oriented calculi is the integration of automated and interactive theorem proving; recent years have seen new efforts to bring these two braids of computer supported reasoning back together to combine the advantages of both approaches [Ahrendt *et al.*, 98, Bjørner *et al.*, 98]. Other applications include (1) the translation of proofs into natural language, for which the transformation into an interaction-oriented calculus is an intermediate step; and (2) checking proof correctness, because interaction-oriented proofs are typically easier to check by a machine for the same reasons that they are easier to understand and validate by a human user.

Applications of our proof transformation are described in [Section 7]; and finally, in [Section 8] we draw conclusions from our work.

To place our work in the general context of proof transformations, we briefly classify different types of transformations. The first type are translations between inherently different calculi—for example, between semantic tableaux and resolution [Wolf, 94] or between natural deduction and resolution [Pfenning, 84].[1] [Eder, 92] addresses the question of p-simulation between different first-order calculi. Such transformations between different calculi are relevant for the relation between automated proof search and human interaction in case the calculi are well suited for these different purposes [Miller, 84, Pfenning, 84]. A second type of transformations are normalisation and optimisation of proofs within the same calculus (without changing the set of available rules). A further type of transformations—one of the most important fields in proof theory—is the elimination or introduction of rules (resp. their applications) such as, for example, cut elimination. In the context of presenting automatically generated proofs to humans, the *introduction* of cuts (or lemmata [Pfenning and Nesmith, 90]) is of interest.

In this paper, we present a transformation of a different kind. It is a translation between different members of the same family of calculi (semantic tableaux). One calculus can be seen as a *refinement* of the other, but it is not just a sub- oder a superset. To make proofs easier to understand for humans, refinements tailor-made for automated search are eliminated.

## 2 The Two Paradigms

### 2.1 Enhancements and Improvements of Tableau Calculi

Below, the two main classes of techniques for improving tableau calculi are described in general. Only those techniques are considered that require the calculus to actually be changed; heuristics and techniques for organising the proof search are not discussed as they do not affect the form of the constructed proofs and, thus, do not affect their transformation.

*Strengthening the calculus.* A calculus is strengthened if it is changed in such a way that shorter proofs for at least some formulae exist. In most cases, strengthening adds non-determinism to a calculus, i.e., there are more possibilities to proceed at each tableau expansion step. Thus, there is a trade-off between the advantage of shorter proofs and the disadvantage that these short proofs may be harder to find as there are more choice points in the search space.

[1] [Pfenning, 84] actually uses *expansion trees*, which can be seen as an abstraction of resolution proofs. [Miller, 84] describes a translation in the other direction, from expansion trees into natural deduction (but in the framework of higher order logic).

*Post-poning choice points.* Often, it is possible to uniformly represent different tableaux—and thus different parts of the search space—by a single tableau using additional syntactical devices. A typical example is the *rigid variable* technique [see Section 4.1], where tableaux that are identical up to the replacement of terms by other terms are all represented by a single tableau in which a free (or dummy) variable is used as a place holder for the different terms.

### 2.2 Interaction-oriented Calculi

Interaction-oriented calculi are used if a deduction system has to interact with a human user, where the reason for interaction may either be that the system is not (fully) automated or that a proof is communicated to the user.

Humans prefer calculi with simple rules. Rule applications should be syntactically (not necessarily semantically) simple, i.e., each application should be easy to validate, the pre-conditions should be easy to check, and effects of a rule application easy to understand.[2] In particular, rule applications should not have any non-local side effects, as humans tend to modularise a problem and focus on proving one sub-problem at a time.

On the other hand, a human user's intuition and meta-knowledge about the problem domain eliminates the need to minimise the search space. Highly non-deterministic rules may be used—as long as they are simple in the sense that their applications are easy to validate; an example is the non-analytic cut rule that imposes no restrictions on the cut formulae that can be introduced.

Techniques for post-poning choice points are often not suitable for enhancing interaction-oriented calculi as they typically result in rules that are syntactically more complicated inference rules and have non-local side effects (such as, for example, the instantiation of rigid variables).

### 2.3 Search-oriented Calculi

For automated deduction, minimising the size of the search space is the main objective in designing a calculus. To achieve this goal, non-determinism is eliminated whenever possible. Though short proofs are in general easier to find than long proofs, the possibility of finding short proofs may be sacrificed for the sake of a smaller search space. Consequently, because strengthening a calculus often increases its non-determinism, weaker versions with fewer choice points are often preferable for automated deduction.[3]

Strengthening the calculus by adding a non-analytic cut rule, which allows to deduce the conclusion "$\phi$ or $\neg\phi$" from the empty premiss for all formulae $\phi$,

[2] Calculi for interactive *higher order* theorem proving often have rules that are *not* simple in that sense. The reason why these rules are acceptable for a human user—despite their complexity—is that they constitute comparatively large proof steps. But even in these higher-order calculi, at least the rules for handling the *first-order* part of the logic are simple.

[3] In search-oriented tableau calculi, techniques are frequently used that restrict the search space without affecting the form of the constructed proofs—such as ordering and connectedness conditions [Beckert and Hähnle, 98, Letz, 98]. They sacrifice the possibility to find shorter proofs for the sake of less non-determinism. Such techniques are not considered here, as using them means changing the proof search without changing the calculus itself.

is a typical example where the disadvantage of additional non-determinism outweighs the advantage of the existence of shorter proofs. Therefore, in automated deduction systems the cut rule is *never* used unrestrictedly. There are, however, restricted versions of the cut rule, such as the generation of local lemmata [see Section 4.3], that still can reduce the size of the smallest proofs exponentially but do not lead to too many additional choice points.

A method for strengthening tableau calculi that is very useful for proof search is the introduction of *universal variables* [see Section 4.2]; this technique can lead to considerably smaller proofs, and adds only very few additional choice points.

Techniques that postpone choice points are always of advantage for proof search if the only measure considered is the number of expansion steps that have to be executed to find a tableau proof. The disadvantage is that additional syntactical devices and bookkeeping mechanisms are required, which can be difficult to implement and lead to computational overhead.

In a certain sense, post-poning choice points leads to a breadth-first search where different parts of the search space are investigated simultaneously, until additional information has been gathered that allows to make an informed decision about which part of the search space should be further investigated. For example, if closing a tableau branch requires a rigid variable $X$ to be instantiated with a certain term $t$, then the decision to instantiate $X$ with $t$ is informed in the sense that the instantiation is known to be useful as a branch closure exists which is not possible without the instantiation.

Note that, in contrary to interaction-oriented calculi, it is acceptable for search-oriented calculi to have many specialised inference rules, (syntactically) complex rules, and rules with non-local side effects (such as the instantiation of rigid variables).

## 3 An Interaction-oriented Tableau Calculus

In this section, we define a *ground* tableau calculus for first-order predicate logic, which is a typical representative of the class of interaction-oriented calculi. This version of tableaux is called *ground* because universally quantified variables are instantiated by ground terms when the $\gamma$-rule is applied. Except for a more sophisticated $\delta$-rule for handling existential quantification (see below), we use the standard ground tableau rules (as defined by [Smullyan, 68]). As the calculus is meant to be used interactively and not automatically, an unrestricted *cut rule* is included.

The definition of the tableau expansion rules makes use of the *unifying notation* given in [Tab. 1], where the left-most columns assign the types $\alpha$, $\beta$, $\gamma$, and $\delta$ to formula patterns.

[Tab. 2] contains the rule schemata for our interaction-oriented calculus (premisses and conclusions are separated by a horizontal bar, while vertical bars in the conclusion denote different *extensions*). The right-most schema is the non-analytic *cut rule*, in which $\phi$ can be any formula.

Both the $\gamma$-rule and the cut rule are highly non-deterministic, which is typical for an interaction-oriented calculus and allows the user to make use of his intuition and domain knowledge.

The expansion rule schemata in [Tab. 2] all have a very simple form and are easy to understand. Moreover, they can be seen as defining the calculus com-

| $\alpha$ | $\alpha_1$ | $\alpha_2$ |
|---|---|---|
| $\phi_1 \wedge \phi_2$ | $\phi_1$ | $\phi_2$ |
| $\neg(\phi_1 \vee \phi_2)$ | $\neg\phi_1$ | $\neg\phi_2$ |
| $\neg(\phi_1 \rightarrow \phi_2)$ | $\phi_1$ | $\neg\phi_2$ |
| $\neg\neg\phi$ | $\phi$ | $-$ |

| $\beta$ | $\beta_1$ | $\beta_2$ |
|---|---|---|
| $\phi_1 \vee \phi_2$ | $\phi_1$ | $\phi_2$ |
| $\neg(\phi_1 \wedge \phi_2)$ | $\neg\phi_1$ | $\neg\phi_2$ |
| $(\phi_1 \rightarrow \phi_2)$ | $\neg\phi_1$ | $\phi_2$ |

| $\gamma$ | $\gamma_1(x)$ |
|---|---|
| $(\forall x)(\phi(x))$ | $\phi(x)$ |
| $\neg(\exists x)(\phi(x))$ | $\neg\phi(x)$ |

| $\delta$ | $\delta_1(x)$ |
|---|---|
| $\neg(\forall x)(\phi(x))$ | $\neg\phi(x)$ |
| $(\exists x)(\phi(x))$ | $\phi(x)$ |

**Table 1:** Unifying notation.

$$\frac{\alpha}{\alpha_1} \atop \alpha_2 \qquad \frac{\beta}{\beta_1 \mid \beta_2} \qquad \frac{\gamma}{\gamma_1(t)} \qquad \frac{\delta}{\delta_1(c)} \qquad \frac{}{\phi \mid \neg\phi}$$

where $t$ is any ground term. where $c = sko(\delta)$.

**Table 2:** Rule schemata for the ground version of tableaux.

pletely; no additional explanations are needed (in particular, closing a branch does not change the tableau). This makes the calculus very user-friendly. Definition 1 merely formalises the definition of tableaux for a given set $\Phi$ of formulae that is implicitly already contained in [Tab. 2].

**Definition 1.** A tableau is a finitely branching tree whose nodes are first-order formulae. A branch in a tableau $T$ is a maximal path in $T$. Given a set $\Phi$ of sentences[4], the *tableaux for* $\Phi$ are (recursively) defined by:

1. The tree consisting of the one branch $\Phi$ is a tableau for $\Phi$ (initialisation).[5]
2. Let $T$ be a tableau for $\Phi$, $B$ a branch of $T$, and $\psi$ a formula in $B$. If the tree $T'$ is constructed by extending $B$ by as many new branches as the tableau expansion rule corresponding to $\psi$ has extensions, where the nodes of the branches are labelled with the formulae in the extensions, then $T'$ is a tableau for $\Phi$ ($\alpha$-, $\beta$-, $\gamma$-, $\delta$-expansion).
3. Let $T$ be a tableau for $\Phi$, $B$ a branch of $T$, and $\phi$ an arbitrary formula. If the tree $T'$ is constructed by extending $B$ by two new single node branches, one containing the formula $\phi$ and the other containing the formula $\neg\phi$, then $T'$ is a tableau for $\Phi$ (cut-expansion).

**Definition 2.** Given a tableau $T$, a branch $B$ of $T$ is *closed* iff $B$ contains a pair $\phi, \neg\phi$ of complementary formulae; otherwise it is *open*. A tableau is closed if *all* its branches are closed.

**Definition 3.** A *tableau proof* for (the unsatisfiability of) a set $\Phi$ of sentences consists of a tableau $T$ for $\Phi$ that is closed.

[4] A *sentence* is a first-order formula not containing any free variables.
[5] We identify the set $\Phi$ with a branch whose nodes are the formulae in $\Phi$.

So far we have not explained the meaning of the operator *sko* used in the $\delta$-rule schema. For skolemisation, we use symbols from a special infinite set $F_{\text{sko}}$ of *Skolem constant symbols*, which is disjoint from the set $F_\Sigma$ of function symbols for any given *first-order signature* $\Sigma = \langle P_\Sigma, F_\Sigma \rangle$. The extended signature $\langle P_\Sigma, F_\Sigma \cup F_{\text{sko}} \rangle$ is denoted by $\Sigma^*$.

**Definition 4.** Given a signature $\Sigma = \langle P_\Sigma, F_\Sigma \rangle$, the function *sko* assigns to each $\delta$-formula $\delta$ over $\Sigma^*$ a symbol $sko(\delta) \in F_{\text{sko}}$ such that $sko(\delta) > c$ for all $c \in F_{\text{sko}}$ occurring in $\delta$, where $>$ is an arbitrary but fixed ordering on $F_{\text{sko}}$.

Note, that the symbol $c = sko(\delta)$ may already occur in the context (on the tableau branch) where our $\delta$-rule is applied to replace an existentially quantified variable with $c$. In contrast to that, the Skolem symbol introduced by an application of the classical (ground) $\delta$-rule is always *new* w.r.t. the context. The intuitive reason why our $\delta$-rule is sound although the Skolem symbol $c$ may not be new is the following: In general, a (ground) $\delta$-rule is sound if it guarantees the dependency relation between Skolem symbols is acyclic. That dependency relation is the transitive closure of the relation $\leadsto$ defined by: For all Skolem symbols $c$ and $d$, $c \leadsto d$ iff $c = sko(\varphi(d))$. An easy (though not optimal) way to ensure that the dependency relation is acyclic, is to demand that each Skolem symbol is new w.r.t. the context where it is used by the $\delta$-rule. Our improved $\delta$-rule is more liberal but ensures nevertheless that the dependency relation is acyclic.

Such more liberal rules were first used in free variable calculi [see Section 4.1], but they are useful for ground calculi as well. Our $\delta$-rule may not be as syntactically simple as the classical rule; that disadvantage is, however, made up for by the fact that our rule is strictly local (whereas the concept of a "new" symbol is inherently global). An additional benefit of using an improved rule in the ground calculus as well is that the transformation between the free variable (search-oriented) and the ground (interaction-oriented) calculus gets easier.

## 4 A Search-oriented Tableau Calculus

Here, we introduce a tableau calculus that is designed for automated proof search. We use the concepts of rigid variables, universal variables, and local lemmata. For an overview of these and other advanced techniques for tableau-based automated deduction we refer to [Beckert and Hähnle, 98].

### 4.1 Rigid Variables

The concept of rigid variables[6] is based on the fact that the $\gamma$-rule allows to derive conclusions of the form $\gamma_1(t)$ for *all* terms $t$. Therefore, instead of guessing terms when they are introduced, they can be represented by a rigid variable $X$. Later, when the proof procedure has good reason to use a particular instance $t$, the variable $X$ is instantiated "on demand"; that means, all occurrences of $X$ are replaced by $t$. Usually, this is done when at least one branch can be closed (in

[6] In the literature—in particular on calculi for first-order predicate logic—, several other names have been used for free and, in particular, for rigid variables, including *parameter*, *dummy variable*, and *meta variable*.

the sense of Definition 2) provided that $X$ is replaced by $t$. Unification is used to find a *most general* substitution that allows to close a branch. Using rigid variables reduces the number of choice points in the construction of a tableau proof and thus the size of the search space.

Intuitively, a formula containing a rigid variable $X$ stands for a single (but unknown) element of the set of all formulae that are the result of replacing $X$ by some term. All occurrences of a rigid variable in a tableau have to be instantiated by the same term (which is why these variables are called "rigid").

The possibility to instantiate rigid variables with Skolem symbols leads to difficulties if a $\delta$-rule is used whose soundness is based on demanding the Skolem symbol that is introduced to be new. Consider, for example, the tableau shown on the left in [Fig. 1]. The last expansion rule applied has been a $\delta$-rule (introducing the Skolem symbol $c$).[7] When the closing substitution $\{X/c\}$ is applied, the tableau becomes ground (shown on the right in [Fig. 1]. This ground tableau cannot be constructed using a *ground* calculus with the classical $\delta$-rule that demands Skolem symbols to be new. But it can be constructed using the ground calculus defined in [Section 3], which has a more liberal $\delta$-rule. This example illustrates why a tableau proof transformation from a rigid variable calculus into a ground calculus using a liberal $\delta$-rule is easier than a transformation into a calculus using the classical $\delta$-rule.

$$
\begin{array}{c}
(\forall x)(p(x) \wedge (\exists y)\neg p(y)) \\
| \\
p(X) \wedge (\exists y)\neg p(y) \\
| \\
p(X) \\
| \\
(\exists y)\neg p(y) \\
| \\
\neg p(c)
\end{array}
\qquad\qquad
\begin{array}{c}
(\forall x)(p(x) \wedge (\exists y)\neg p(y)) \\
| \\
p(c) \wedge (\exists y)\neg p(y) \\
| \\
p(c) \\
| \\
(\exists y)\neg p(y) \\
| \\
\neg p(c) \\
* \\
\text{closed by } \{X/c\}
\end{array}
$$

**Figure 1:** A tableau that cannot be constructed using the classical ground $\delta$-rule.

### 4.2 Universal Variables

Under certain conditions, there is an alternative use of free variables for strengthening a tableau calculus. Instead of using a free variable to represent a single but unknown term, it can be used as well to represent *all* terms. Then, a formula containing such a free variable $X$ stands for the set of *all* formulae that are the result of replacing $X$ by some term. Intuitively, these free variables can be seen as being universally quantified on the meta-level; accordingly they are called *universal variables*. The advantage of using universal variables is the following:

---

[7] For this example, it is not important exactly which $\delta$-rule is used. It can be the rule we define below, but earlier versions of the $\delta$-rule [Hähnle and Schmitt, 94, Fitting, 96] lead to the same result (however, the rule given in the *first edition* of [Fitting, 96] leads to a different tableau).

Often several different instances of a tableau formula containing free variables have to be used to close a branch (or a sub-tableau). In rigid variable tableaux, the mechanism to do so is to apply the $\gamma$-rule more than once to the same premiss, introducing new rigid variables to generate variants of a formula. Suppose a tableau branch $B$ contains a formula $\phi(X)$. When $X$ is instantiated in the tableau (to close a branch), then $\phi(X)$ is instantiated as well. In particular situations, however, it is possible to protect $X$ in $\phi(X)$ from being instantiated. This is sound whenever $\phi(t)$ is a logical consequence of the formulae on $B$ *for all* terms $t$. This is undecidable in general, but many of the protectable free variables can be recognised easily. For this purpose we introduce the notion of *decorated formulae.*

**Definition 5.** A *decorated formula* $\mathcal{U} : \Phi$ consists of a set $\mathcal{U}$ of free variables and a formula $\phi$. The variables in $\mathcal{U}$ are called the *universal variables* of $\phi$; and the variables in $\mathrm{Free}(\phi) \setminus \mathcal{U}$ are called the *rigid variables* of $\phi$ (where $\mathrm{Free}(\phi)$ is the set of all free variables in $\phi$).

In the search-oriented calculus defined below, the tableau nodes are decorated formulae. Intuitively, $\mathcal{U} : \phi(X)$ represents the different instances $\phi(t)$ provided that $X \in \mathcal{U}$; they can all be used without first generating variants of $\phi(X)$. Using the concept of universal variables yields shorter tableau proofs, and in most cases reduces the search space. If both universal and rigid variables are used in a calculus, then closing substitutions are only applied to the rigid occurrences of a variable.

Before defining our search-oriented calculus formally, we give simple examples that demonstrate the usefulness of universal variables.

*Example 1.* We consider tableaux for the set $\Phi = \{(\forall x)(p(x)), \neg p(a) \vee \neg p(b)\}$ of formulae. The top left tree in [Fig. 2] is a *rigid variable tableau* for $\Phi$ that cannot be closed immediately as no single substitution for $X$ allows to close both branches. To find a proof, the expansion rule has to be applied again to the $\gamma$-formula $(\forall x)(p(x))$ to add a variant $p(X')$ of $p(X)$. Then, the closed tableau (top right in the figure) can be deduced.

The bottom left tableau in the figure is a *mixed variable tableau* for $\Phi$ (only universal variables are used in this example). It contains the decorated formula $\{X\} : p(X)$ instead of $p(X)$. This tableau can be expanded to a closed tableau (bottom right in the figure) without applying a substitution, because $\{X\} : p(X)$ represents all formulae of the form $p(t)$, including $p(a)$ and $p(b)$.

An additional advantage of universal variables is that they help to avoid redundancies inherent to rigid variable calculi. If, for example, a rigid variable tableau branch contains the formulae $p(X_1), p(X_2), \neg p(a)$, and $\neg p(b)$, then there are four different possibilities to close the branch. If, however, the branch contains the decorated formula $\{X\} : p(X)$ instead of $p(X_1)$ and $p(X_2)$, then there is only one possibility, namely closing the branch without instantiating any variable.

The concept of universal variables was first described in [Beckert and Hähnle, 92]; it has later been improved [Beckert, 98, Beckert and Hähnle, 98].

### 4.3 Local Lemmata

A simple and in many cases useful way of strengthening a tableau calculus is to make sure that the extensions of a conclusion do not *intersect semantically.*

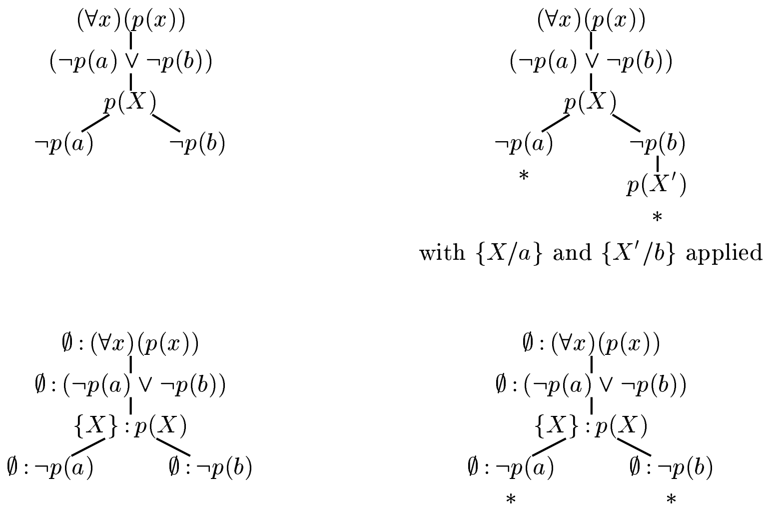


**Figure 2:** Tableaux for Example 1

For example, the two extensions $p$ and $q$ of the conclusion for the premiss $p \vee q$ intersect semantically, as $p$ and $q$ can both be satisfied by a single interpretation.

The application of an expansion rule using a conclusion with semantically intersecting extensions adds, in some sense, less information to the tableau. Intuitively, to close a tableau, one has to show that no interpretation satisfies any of its branches. If there are branches that may be satisfied by the same interpretation, then such an interpretation has to be considered and excluded more than once. Extensions can be made intersection-free by adding tableau formulae; this may require the use of additional extensions, and one has to be careful to preserve soundness.

The $\beta$-rule of the interaction-oriented calculus from [Section 3] allows to derive conclusions that are not intersection-free. An alternative $\beta$-rule schema is

$$\frac{\beta}{\beta_1 \;\Big|\; \begin{array}{l}\beta_2\\ \neg\beta_1\end{array}}$$

which produces intersection-free conclusions (the formula $\neg\beta_1$ that is added to the right extension can be considered to be a *local lemma*). Using this new schema leads to a non-elementary reduction in the size of the shortest proofs for certain classes of formulae [Egly, 98].

The unrestricted *cut rule* of the interaction-oriented calculus [see Section 3] is also an intersection-free rule. However, for reasons discussed in [Section 2.3], it is not included in our search-oriented calculus.

### 4.4 The Calculus

We now have everything at hand to define our search-oriented tableau calculus. It is a *mixed variable calculus*, i.e., it is working with both *rigid* and *universal* variables. For this purpose, we use the decorated formulae from Definition 5. The calculus also uses the $\beta$-rule schema from [Section 4.3], which generates local lemmata. The rule schemata are given in [Tab. 3]. The square brackets in the $\beta$-rule schema indicate that adding the local lemma is optional.

| | | | |
|---|---|---|---|
| $\dfrac{\mathcal{U}:\alpha}{\begin{array}{c}\mathcal{U}\cap \mathrm{Free}(\alpha_1):\alpha_1\\ \mathcal{U}\cap \mathrm{Free}(\alpha_2):\alpha_2\end{array}}$ | $\dfrac{\mathcal{U}:\beta}{\emptyset:\beta_1 \;\Big|\; \begin{array}{c}\emptyset:\beta_2\\ {[\emptyset:\neg\beta_1]}\end{array}}$ | $\dfrac{\mathcal{U}:\gamma}{\mathcal{U}\cup\{X\}:\gamma_1(X)}$ where $X$ is a new free variable. | $\dfrac{\mathcal{U}:\delta}{\mathcal{U}:\delta_1(f(X_1,\ldots,X_n))}$ where $f = sko(\delta)$ and $\{X_1,\ldots,X_n\} = \mathrm{Free}(\delta)$. |

**Table 3:** Rule schemata for the mixed variable version of tableaux.

For the free variable version of the $\delta$-rule, the definition of the operator *sko* must be changed slightly. Now, $F_{\mathrm{sko}}$ is an infinite set of Skolem *function* symbols.

**Definition 6.** Given a signature $\Sigma = \langle P_\Sigma, F_\Sigma\rangle$, the function *sko* assigns to each $\delta$-formula $\delta$ over $\Sigma^*$ a function symbol $sko(\delta) \in F_{\mathrm{sko}}$ such that (a) $sko(\delta) > f$ for all $f \in F_{\mathrm{sko}}$ occurring in $\delta$, where $>$ is an arbitrary but fixed ordering on $F_{\mathrm{sko}}$, and (b) for all $\delta$-formulae $\delta, \delta'$ over $\Sigma^*$, the symbols $sko(\delta)$ and $sko(\delta')$ are identical if and only if $\delta$ and $\delta'$ are identical up to renaming of free variables.

In mixed variable tableaux, rigid variables are instantiated by substitutions that result from unifying pairs of potentially complementary formulae. Universal variables are protected from being instantiated, reflecting the fact that every possible instance of the corresponding formula *could* be derived. Accordingly, universal variables can be renamed to avoid occur check clashes when formulae containing the same universal variables are unified. However, instead of actually renaming universal variables in tableaux, we define the unification of decorated formulae appropriately.

**Definition 7.** A substitution $\sigma$ is a *unifier* of decorated formulae $\mathcal{U}:\phi$ and $\mathcal{U}':\phi'$ in a tableau $T$ if it is the restriction of a substitution $\tau$ with the property $(\phi\pi)\tau = (\phi'\rho)\tau$ to the set $\mathrm{Free}(\phi)\setminus\mathcal{U})\cup(\mathrm{Free}(\phi')\setminus\mathcal{U}'$ of variables where $\pi$ is a renaming of the variables in $\mathcal{U}$ and $\rho$ is a renaming of the variables in $\mathcal{U}'$ with variables new to $T$.

**Definition 8.** Let $\Sigma$ be a first-order signature. A tableau (over $\Sigma$) is a finitely branching tree whose nodes are *decorated* formulae over $\Sigma^*$. A branch in a tableau $T$ is a maximal path in $T$. Given a set $\Phi$ of sentences over $\Sigma$, the *tableaux for* $\Phi$ are (recursively) defined by:

1. The single branch consisting of the decorated formulae in $\{\emptyset:\phi \mid \phi \in \Phi\}$ is a tableau for $\Phi$ (initialisation).
2. Let $T$ be a tableau for $\Phi$, $B$ a branch of $T$, and $\mathcal{U}:\psi$ a decorated formula in $B$. If the tree $T'$ is constructed by extending $B$ by as many new branches as the tableau expansion rule corresponding to $\mathcal{U}:\psi$ has extensions, where the nodes of the branches are labeled with the decorated formulae in the extensions, then $T'$ is a tableau for $\Phi$ (expansion).

3. Let $T$ be a tableau for $\Phi$, $B$ a branch of $T$, and $\mathcal{U}:\psi$ and $\mathcal{U}':\neg\psi'$ literals in $B$. If $\sigma$ is a unifier of $\mathcal{U}:\psi$ and $\mathcal{U}':\psi'$ (according to Definition 7), and $T' = T\sigma$, i.e., $T'$ is constructed by applying $\sigma$ to all formulae in $T$, then $T'$ is a tableau for $\Phi$ (closure).

## 5 The Transformation of Free Variable Instantiations

A transformation from one calculus into another often results in a proof containing additional redundancies. It is desirable to minimise these redundancies while preserving the original proof structure as much as possible. In this section the technique for avoiding the introduction of redundant proof parts is described that is part of our transformation.

The main difficulty in transforming proofs from the search-oriented calculus from [Section 4] into the interaction represented calculus from [Section 3] is that the former uses free variables (both rigid and universal), whereas the latter is a ground calculus without free variables.

Assume, for example, that in a free variable tableau proof the formula $\phi(X)$ is derived applying the $\gamma$-rule to the formula $(\forall x)(\phi(x))$ and that later a successor formula $\psi(X)$ of $\phi(X)$ is used to close a branch $B$ unifying it with some formula $\neg\psi(a)$. Then, in the corresponding ground proof, the formula $\phi(a)$ has to be derived from $(\forall x)(\phi(x))$ using the ground $\gamma$-rule, and subsequently $\psi(a)$ has to be derived from $\phi(a)$ mimicking the derivation of $\psi(X)$ from $\phi(X)$, which then finally allows to close the branch in the ground tableau that corresponds to $B$. Of course, the situation is in general much more complex—in particular because universal variables can be instantiated multiply. Mechanisms for extensive bookkeeping are needed to keep track of how formulae are derived and which instances are needed (to derive an instance $\psi(a)$ in the ground calculus, the instance $\phi(a)$ of the predecessor formula has to be derived first).

In addition, to minimise the number of formulae that are generated in the ground version of a proof, the order in which different free variables occurring in a formula have been introduced by $\gamma$-rule applications has to be taken into account. This order induces a dependency relation between free variables, as the following example illustrates (a similar situation is described in Example 4):

*Example 2.* Assume that two instances $p(a, b)$ and $p(a, c)$ of some decorated formula $\{X, Y\}:p(X, Y)$ are used in a free variable tableau proof to close branches.

If this formula has been derived from the $\gamma$-formula $\phi = (\forall x)(\forall y)(p(x, y))$, i.e., if $X$ has been introduced first, then in the ground calculus one can derive the formula $\psi = (\forall y)(p(a, y))$ and with two $\gamma$-rule applications to $\psi$ the ground instances $p(a, b)$ and $p(a, c)$. In that case, one can make use of the fact that the ground instances coincide in the instantiation of $X$ by applying the $\gamma$-rule only once to $\phi$.

If, however, the $\gamma$-formula is of the form $\phi' = (\forall y)(\forall x)(p(x, y))$, i.e., if $Y$ has been introduced first and $X$ depends on $Y$, then in the ground calculus one has to derive both $\psi'_1 = (\forall x)(p(x, b))$ and $\psi'_2 = (\forall x)(p(x, c))$, from which then the two ground instances can be derived. In that case, the fact that the ground instances coincide in the instantiation of $X$ does not lead to a reduction in the number of generated formulae.

In the following, we use (sets of) *instantiation trees* to represent (sets of) instantiations of the free variables occurring in a formula. The order that is inherent to the trees reflects the dependency relation between the free variables (the instantiations of the variable introduced first are contained in the roots of the trees). The instantiation trees corresponding to the instantiations described in Example 2 are depicted in [Fig. 3].

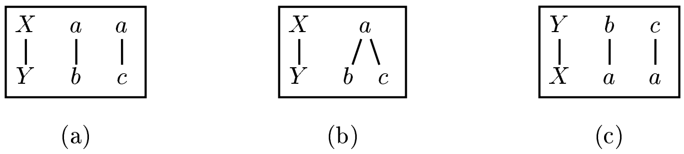


Figure 3: (a) A set of instantiation tree that can be merged. (b) The result of merging these instantiation trees. (c) A set of instantiation tree that cannot be merged (see Example 2).

*Example 3.* In this example, we consider a slightly more complicated case where instantiation trees are merged that consist of more than one branch. Assume that instantiations of some decorated formula $\phi = \{X, Y, Z\} : p(X, Y, Z)$ have already been collected and are represented by the instantiation tree shown in [Fig. 4 (a)]. Assume further that the instance $p(a, b, c)$ is later used to close a tableau branch, i.e., the instantiation tree has to be merged with the one shown in [Fig. 4 (b)]. The result of the merging process is shown in [Fig. 4 (c)]. In the ground tableau proof, a total number of six $\gamma$-rule applications is needed to generate the three required ground instances of $\phi$ (a naive transformation would generate nine $\gamma$-rule applications).

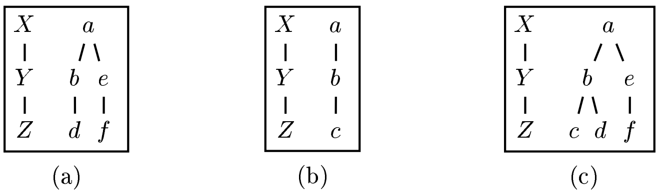


Figure 4: The instantiation tree (c) is the result of merging the trees (a) and (b) (Example 3).

Our transformation algorithm, which has been implemented in Prolog as part of the tableau-based theorem prover $_3T^AP$, works in three steps:

1. Compute and collect all instantiations of free variables (both rigid and universal) required to close all branches of the given free variable tableau proof;

in this first step only the instantiations of variables occurring in literals actually used for branch closure are considered. The instantiations are represented by instantiation trees attached to these literals.

2. Compute the necessary instantiations of the free variables in *all* formulae, including the non-literal formulae. The set of instantiation trees of a formula $\phi$ is computed by merging the sets of instantiation trees of all successor formulae of $\phi$.
3. Construct the ground proof. It contains one formula for each path in any of the instantiation trees computed in Step 2 (it may contain additional redundant formulae). In addition, at this stage, the complex Skolem terms of the free variable proof are replaced by Skolem constants.

The size of the ground tableau proof that is the result of this transformation is polynomial in the size of the original free variable proof. However, since the increase in proof size heavily depends on the nesting level of universal quantifiers in the proof, the increase in proof size is in practice only slightly super-linear. Note that our transformation does not make use of the cut rule—which is available in the ground calculus—to translate multiple instantiations of universal variables. A different transformation that makes use of the cut rule was suggested by [Egly, 98]. It can lead to shorter ground proofs (which depending on the purpose of the proof transformation can be important); however, it changes the structure of the proof drastically. Such a change of the proof structure may be tolerable in certain environments, but as our aim was to preserve the proof structure as much as possible for the benefit of a human user, Egly's method is not suitable for our purpose.

Due to space limitations we cannot describe the transformation algorithm in more detail here (a detailed description [in German] is available [Stenz, 97]). Instead we give three examples to illustrate the difficulties arising with the transformation and how they are solved by our method.

*Example 4.* [Fig. 5 (a)] shows a free variable tableau proof after the first step of the transformation, i.e., instantiation trees have been computed for all literals used to close one of the branches. Only the instantiation trees of the non-ground literal $p(X, Y)$ are depicted in the figure; they correspond to its instances $p(a, b)$, $p(a, d)$, and $p(c, d)$, which are used to close the three branches of the tableau ($p(X, Y)$ can be used with different instantiations for the variables $X$ and $Y$ as they are universal in this literal). All other literals in this tableau are ground (the empty instantiation tree is implicitly attached to ground literals). Also, in this and the following figures, the lists of free variables being universal in a formula are not shown for ground formulae (for which the list is always empty).

The result of the second step of the transformation is shown in [Fig. 5 (b)]. The two instantiation trees of $p(X, Y)$ with root $a$ have been merged into one tree. Note, that it is not possible to merge the two trees with the identical leaf $d$ as they have incompatible roots $a$ and $c$. In addition, the set of instantiation trees for the non-literal formula $(\forall y)(p(X, y))$ has been computed.

In [Fig. 5 (c)] the final result of the transformation (a ground tableau proof) is shown. The dotted arrows indicate the correspondence between the paths of the instantiation trees in [Fig. 5 (b)] and the ground formulae in [Fig. 5 (c)].

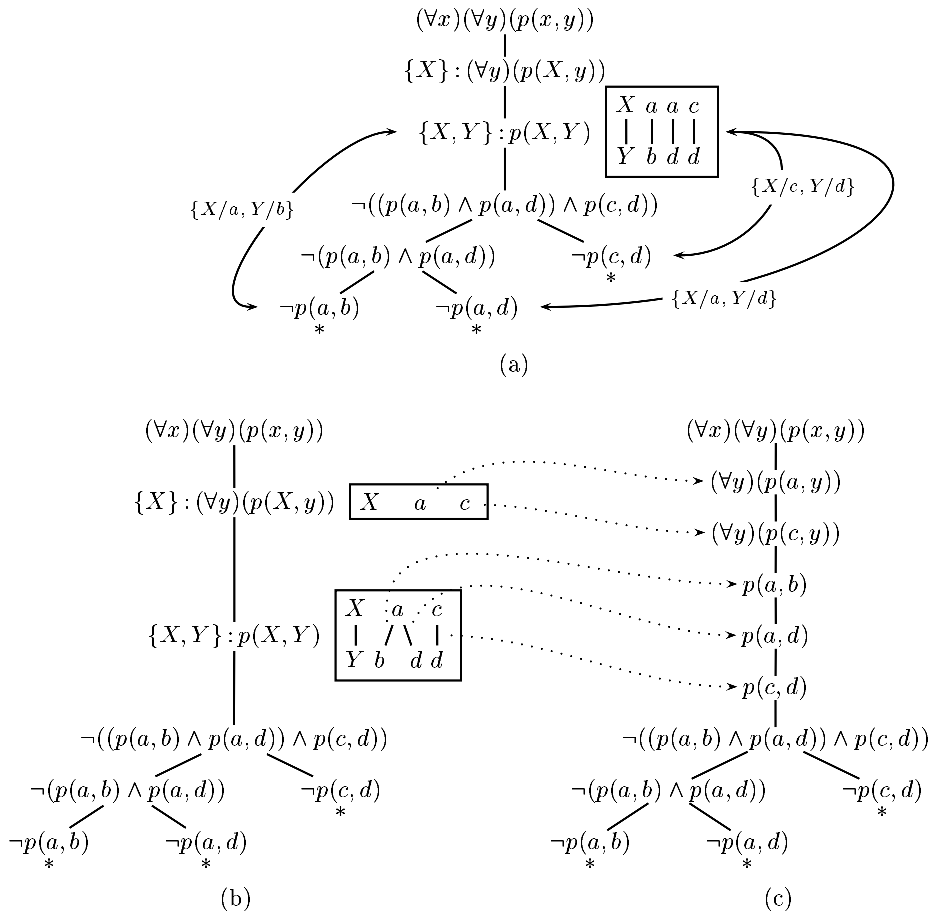

Figure 5: An example for the transformation where instantiation trees contain only *ground* terms (Example 4).

*Example 5.* The free variable tableau proofs shown in [Figs. 7 (a) and 8 (a)] illustrate the difficulties arising if instantiation trees contain non-ground terms. The symbol $\otimes$ occurring in instantiation trees represents an arbitrary instantiation (note that this is different from the empty instantiation tree).

The free variable tableau proofs in these two figures are very similar. The main difference is that in the first case [see Fig. 7] the two instantiations $f(a)$ and $f(Y)$ of the variable $X$ can easily be merged, because the variable $Y$ has to be instantiated with $a$ anyway. In the second case [see Fig. 8], however, there is no single optimal result of the transformation. In this case, the instantiations $f(a)$ and $f(Y)$ of $X$ are not merged by our algorithm. It generates the instance $q(b)$ of $q(Y)$ and the instances $p(f(a))$ and $p(f(b))$ of $p(X)$ [see Figure 8 (b)]. The result of the transformation using these instantiations is shown in [Fig. 8 (c)].

One could as well generate the two instances $q(a)$ and $q(b)$ of $q(Y)$, in which case the two instantiations of $X$ can be merged and generating the single instance $p(f(a))$ of $p(X)$ is sufficient. Then, the resulting ground proof would be different from the one shown in [Fig. 8 (c)] but it would be of the same size.

The main reason for introducing instantiation trees was to avoid the introduction of redundancies into the resulting ground proof. However, as the following example demonstrates, our transformation does not always find an optimal solution (for which all instantiations in the whole tableau would have to be considered simultaneously).

*Example 6.* Assume that two instances of a decorated formula $\{X\}:p(X)$ have been used for branch closure: $p(a)$ and $p(Y)$ where $Y$ is a free variable also occurring on some other branch of the tableau. Our transformation does not merge the two instantiations of $X$ although that is useful if $Y$ is instantiated with $a$ to close other branches of the tableau (this situation is shown in [Fig. 6].

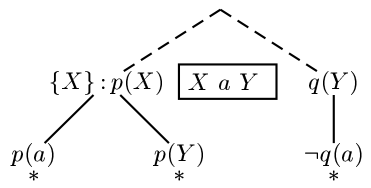


**Figure 6:** The situation described in Example 6.

*Example 7.* The problem illustrated by the final example is that closing a tableau branch may require to instantiate a free variable $Y$ with a term $g(Y)$ containing $Y$. That does not cause problems if the universal variable technique is used (and $Y$ is universal). However, special care has to be taken when a free variable tableau containing such an instantiation is transformed into a ground tableau; the instantiation has to be split into two separate instantiations. This situation arises in the free variable tableau proof shown in [Fig. 9 (a)]. In this proof, $g$ is a Skolem function symbol for which $g = sko(\delta)$ where $\delta = \neg(\forall x)(f(Y) \supset f(x))$.

In [Fig. 9 (b)], which shows the result of the second step of the transformation, the constant $c$ has been chosen to be the arbitrary term that is represented by $\otimes$ in [Fig. 9 (a)]. Note, that two instantiation trees are attached to each of the non-literal formulae, indicating that two separate instances of these formulae have to be generated in the ground version of the proof.

The ground proof, which is shown in [Fig. 9 (c)], contains two different instances of the $\delta$-formula $\psi(Y) = \neg(\forall x)(f(Y) \supset f(x))$, namely $\psi(c)$ and $\psi(d)$; accordingly, two different Skolem constants are introduced. The first of these constants is $d = sko(\psi(c))$ (which corresponds to the term $g(c)$ in the free variable proof) and the second constant is $e = sko(\psi(d))$ (which corresponds to the term $g(g(c))$ in the free variable proof). Since $e$ "depends" on $d$, we have $e > d$ where $>$ is the ordering on $F_{\text{sko}}$ that ensures the dependency relation on Skolem constants to be acyclic (Def. 4).

Since $d$ already occurs in the proof at the time it is used as a Skolem constant in a $\delta$-rule application, this proof could not be constructed using the classical ground $\delta$-rule that demands Skolem constants to be new.

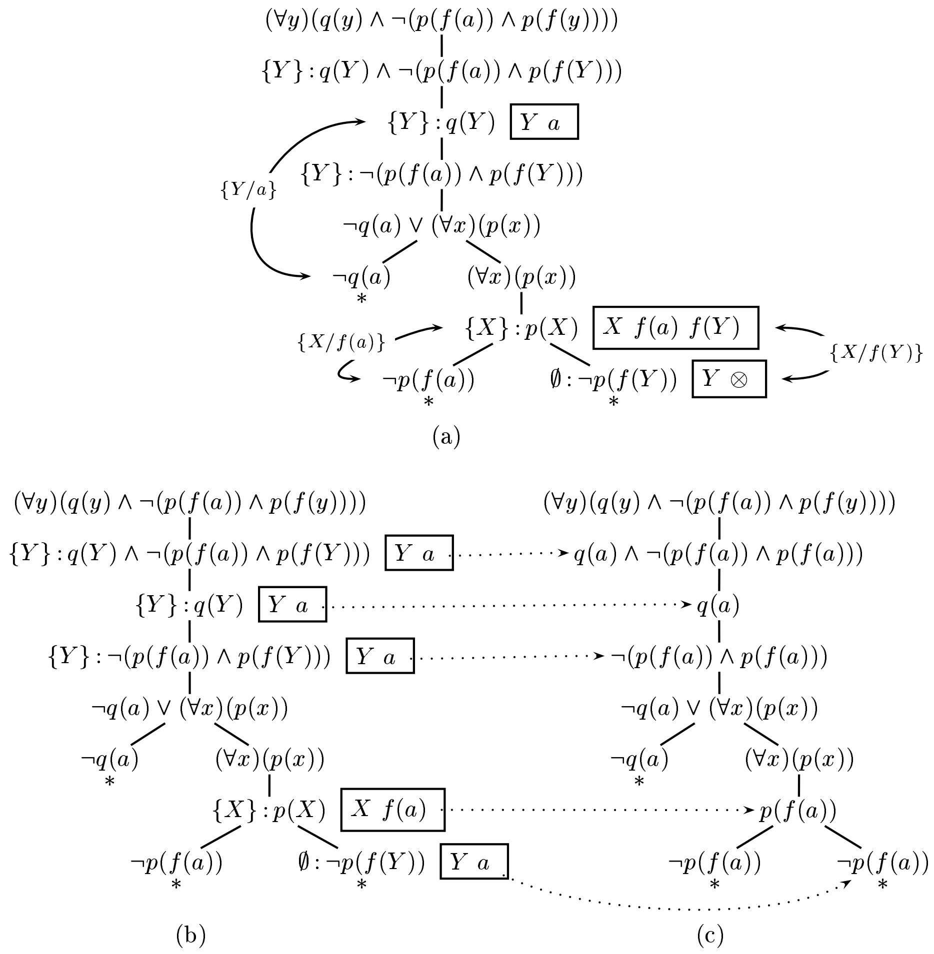


Figure 7: The first example for the transformation where instantiation trees contain free variables (Example 5).

A translation of the ground proof into natural language is shown in [Section 7.2].

## 6 The Transformation of Local Lemmata

Applications of the $\beta$-rule in the search-oriented calculus [see Tab. 3], which may introduce local lemmata, require attention when they are transformed, because the $\beta$-rule of the target calculus [see Tab. 2] does not generate lemmata.

Fortunately, the introduction of local lemmata can be seen as a restricted cut rule application and can therefore easily be simulated since the cut rule is available in the target calculus. A $\beta$-rule application that generates a local lemma

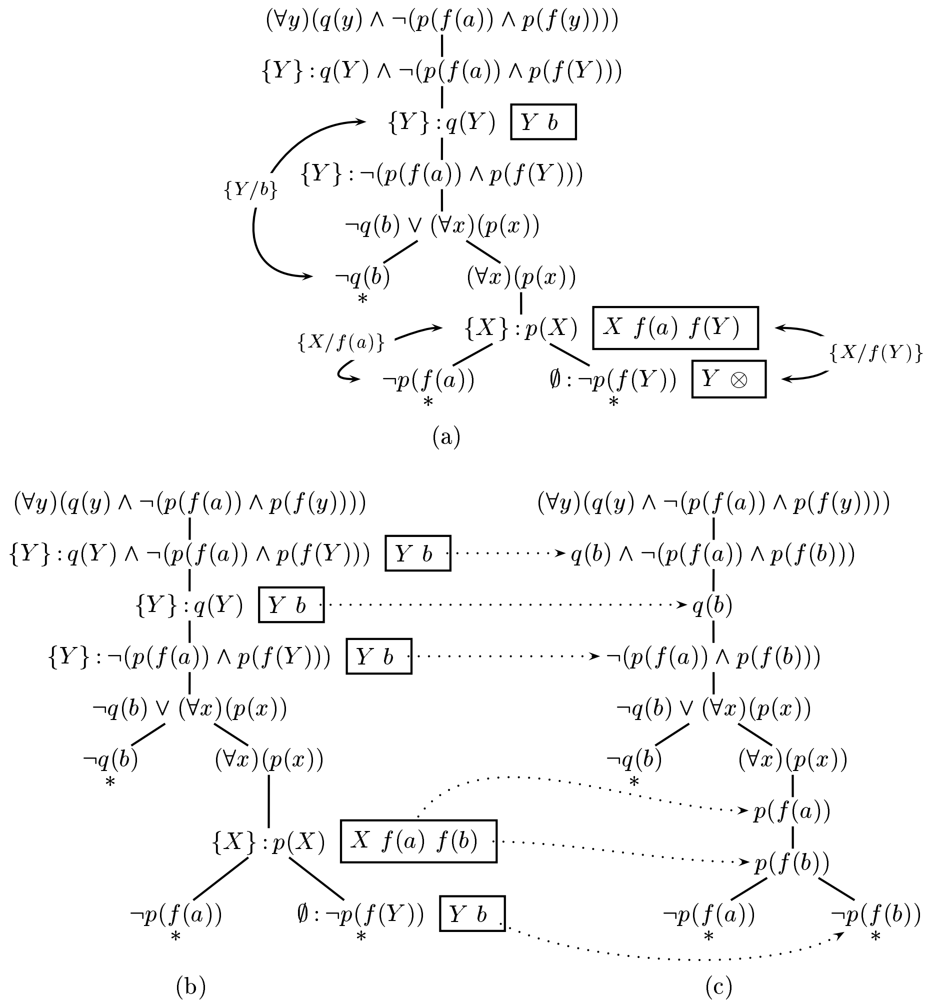

Figure 8: The second example for the transformation where instantiation trees contain free variables (Example 5).

is replaced by a cut rule application and a subsequent $\beta$-rule application (which results in a redundant branch that can immediately be closed); see [Fig. 10] for an example.

In the implementation of our transformation as part of the prover $_3T\!AP$, a simple language for defining translation patterns is made available, that allows to describe in which way applications of rules that do not exist in the target calculus are to be replaced by new sub-proofs consisting of one or several applications of the available rules. The translation of local lemmata (as described above) is an application of this flexible mechanism. Another example is the simulation of special rules in $_3T\!AP$'s search-oriented calculus that, for example, allow to derive from the premiss *false* $\vee\ \phi$ the conclusion $\phi$ in a single step.

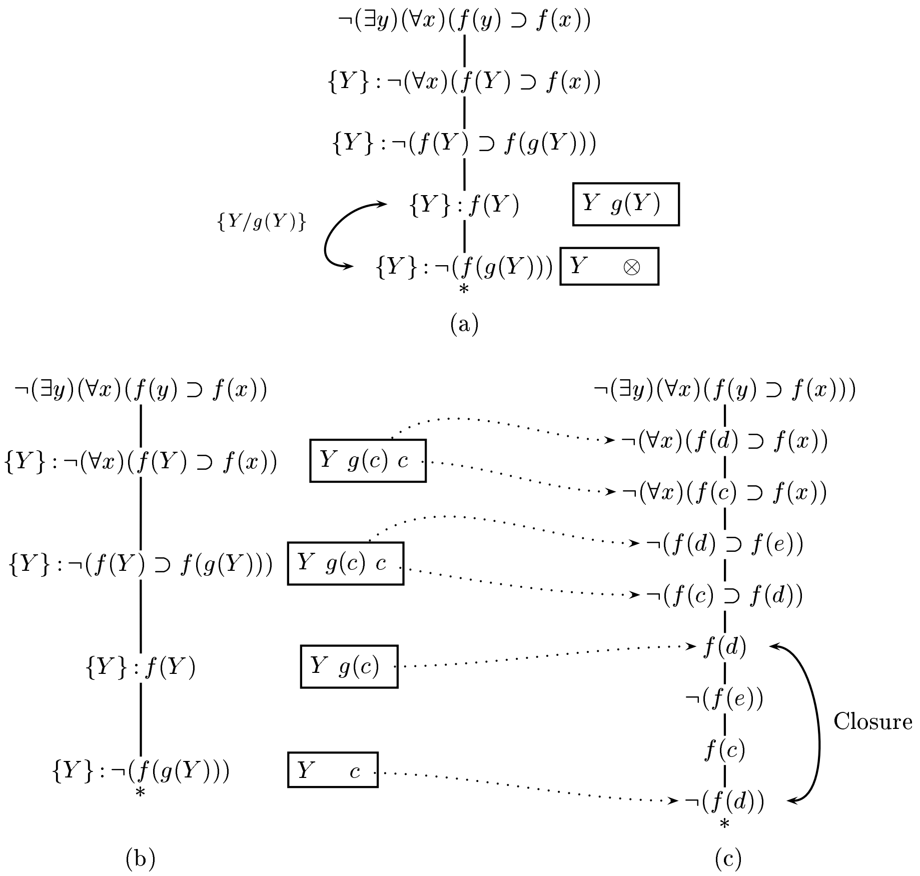


Figure 9: An example for the transformation where a (universal) variable $Y$ is instantiated with a term containing $Y$ (Example 7).

## 7 Applications

In this section, we present three applications of the transformation described in the previous sections to demonstrate its usefulness in practice. All these applications are based on the implementation of our transformation as part of the tableau-based automated deduction system $_3T^AP$.

### 7.1 Integrating Automated and Interactive Theorem Proving

The first application of our proof transformation is the integration of the automated theorem prover $_3T^AP$ [Beckert *et al.*, 96] and the interactive software verification system KIV (Karlsruhe Interactive Verifier) [Reif, 95] to form a single proof system as shown in [Fig. 11]. In this cooperation the KIV system handles all user interaction for constructing proofs, while $_3T^AP$ acts as a background reasoner for solving first-order sub-problems that have been isolated by KIV. Our

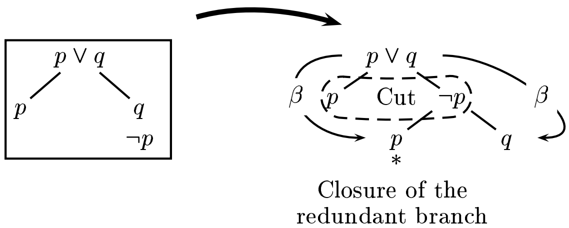


**Figure 10:** Simulating the introduction of a local lemma using the cut rule.

goal was to pass proofs found by $_3T^AP$ back to KIV, instead of only simple messages such as "success" or "no success". This has two main advantages. Firstly, the proof can be presented to the user and, secondly, KIV can integrate the results of $_3T^AP$ into its extensive proof management system, thus eliminating a lot of redundant work. The automated theorem prover $_3T^AP$ uses a search-oriented calculus that is nearly identical to the one described in [Section 4], while the interactive system KIV uses a sequent calculus for dynamic logic, the first-order fragment of which is similar to the interaction-oriented calculus presented in [Section 3].

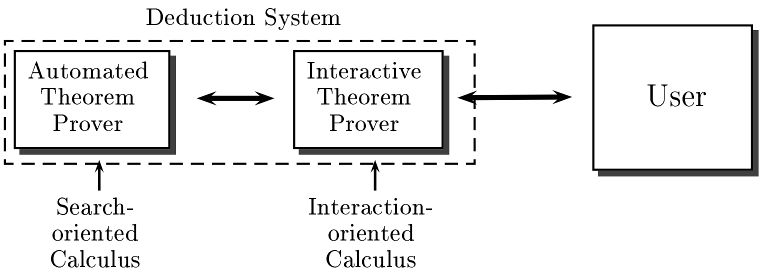


**Figure 11:** The integration of automated and interactive theorem proving.

### 7.2 Proof Presentation in Natural Language

Another integrated deduction system (as shown in [Fig. 11] has been developed in the framework of the ILF project [Dahn *et al.*, 97]. The ILF system has been designed as a generic interactive prover environment for integrating the abilities of human users on the one hand and any sort of automated deduction system on the other hand; $_3T^AP$ is one of the provers able to cooperate with ILF.

The ability to translate proofs from a calculus such as the interaction-oriented target calculus of our transformation into natural language is one of the main features of ILF [Dahn and Wolf, 96, Wolf, 98]. Therefore, the implementation of the proof transformation described in this paper permits the output of proofs found by $_3T^AP$ to be translated by ILF into natural language.

*Example 8.* ILF's translation of the ground tableau proof from Example 7 [see Fig. 9] into natural language is shown in [Fig. 12].

*Proof.* We show indirectly that

$$(\exists y)(\forall x)(f(y) \supset f(x)) \tag{1}$$

Let us assume that

$$(\exists y)(\forall x)(f(y) \supset f(x)) \text{ does not hold.} \tag{2}$$

We chose an arbitrary constant $c_1$ for $y$ in (2) and obtain that $(\forall x)(f(c_1) \supset f(x))$ does not hold. Hence we can introduce $c_2$ for $x$. This gives that

$$f(c_1) \supset f(c_2) \text{ does not hold.} \tag{3}$$

We chose an arbitrary constant $c_3$ for $y$ in (2) and obtain that $(\forall x)(f(c_3) \supset f(x))$ does not hold. Hence we can replace $x$ by the arbitrarily chosen $c_1$ and assume that $f(c_3) \supset f(c_1)$ does not hold. Hence $f(c_1)$ does not hold. By (3) $f(c_1)$. Therefore we have a contradiction. Thus we have completed the proof of (1).

**Figure 12:** A tableau proof translated by ILF into natural language (Example 8).

Note that, in the last paragraph of the proof, the constant $c_1$ replaces the (implicitly) existentially quantified variable $x$ in "$f(c_3) \supset f(x)$ does not hold" although $c_1$ already occurs. That is allowed because (1) $c_1$ was "arbitrarily chosen" when it was first used and because (2) the other constant $c_3$ occurring in the formula does not depend on $c_1$ (the dependency between Skolem constants is discussed in Section 3 below Definition 4). The (satisfied) dependency condition is not pointed out by ILF (it is arguable whether it should be mentioned in a natural language proof presentation).

### 7.3 Proof Checking

A further application for proof transformations is the possibility to check the correctness of proofs more easily.

Since automated theorem provers use complicated heuristics and proof procedures, their implementations are very complex. Therefore, the proofs that are constructed may not be sound in all cases, and it is highly desirable to be able to check the correctness of these proofs. This is made difficult by the fact that search-oriented calculi often employ tableau rules that are difficult to validate and that have non-local side effects [see Section 2.3].

The rules of an interaction-oriented calculus, however, are syntactically simple; for the same reasons that proofs in interaction-oriented calculus are easier to understand and validate for humans [see Section 2.2], they are easier to check for a computer. The proof checker can be implemented as a small and simple program and is therefore more trustworthy. Therefore, the proof transformation procedure is useful for generating the input for a proof checker.

Based on the implementation of our proof transformation as part of $_3T^AP$, we implemented a compact proof checker, which has been very helpful in detecting bugs in both $_3T^AP$ and the implementation of the proof transformation.

## 8 Conclusion

We have categorised logic calculi into two types according to the purpose they serve in theorem proving, either proof search or user interaction. Furthermore we have explained the inherent characteristics of such calculi and how these relate to the designated rôle of a calculus.

As examples we have defined one tableau calculus from each of those two categories, a search-oriented free variable calculus and an interaction-oriented ground calculus, and we have established a proof transformation procedure from proofs in the former into proofs in the latter calculus.

We believe that the problems we encountered in devising such a transformation procedure are not particular to our calculi but are invariably caused by the need to translate between a calculus placing the greatest emphasis on search space minimisation and a calculus designed with the syntactical simplicity of its rules as the main objective. Similar phenomena may, for example, occur if a resolution proof is transformed into a *ground* resolution proof, i.e., a proof where first a set of ground instances of the input clauses are listed, from which in a second step the empty clause is derived.

This work exemplifies and supports the fact that there is a non-trivial gap between calculi that serve different purposes even if they belong to the *same family*. Bridging this gap and thus bringing proofs back to the user requires considerable effort.

Such transformations are not only of theoretical value but they have applications and our implementation shows that they are indeed of practical value.

## References

[Ahrendt *et al.*, 98] Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, Wolfram Menzel, Wolfgang Reif, Gerhard Schellhorn, and Peter H. Schmitt. Integrating automated and interactive theorem proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction — A Basis for Applications*, volume II: Systems and Implementation Techniques, pages 97–116. Kluwer, Dordrecht, 1998.

[Beckert and Hähnle, 92] Bernhard Beckert and Reiner Hähnle. An improved method for adding equality to free variable semantic tableaux. In Depak Kapur, editor, *Proceedings, 11th International Conference on Automated Deduction (CADE), Saratoga Springs, NY, USA*, LNCS 607, pages 507–521. Springer, 1992.

[Beckert and Hähnle, 98] Bernhard Beckert and Reiner Hähnle. Analytic tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction — A Basis for Applications*, volume I: Foundations, pages 11–41. Kluwer, Dordrecht, 1998.

[Beckert *et al.*, 96] Bernhard Beckert, Reiner Hähnle, Peter Oel, and Martin Sulzmann. The tableau-based theorem prover $_3T^AP$, version 4.0. In *Proceedings, 13th International Conference on Automated Deduction (CADE), New Brunswick, NJ, USA*, LNCS 1104, pages 303–307. Springer, 1996.

[Beckert, 98] Bernhard Beckert. *Integration und Uniformierung von Methoden des tableau-basierten Theorembeweisens.* PhD thesis, Universität Karlsruhe, Fakultät für Informatik, 1998.

[Bjørner *et al.*, 98] Nikolaj S. Bjørner, Reiner Hähnle, Wolfram Menzel, Wolfgang Reif, and Peter H. Schmitt, editors. *Proceedings, Integration of Deductive Systems, Workshop at the International Conference on Automated Deduction (CADE), Lindau, Germany*, 1998.

[Dahn and Wolf, 96] Bernd I. Dahn and Andreas Wolf. Natural language presentation and combination of automatically generated proofs. In *Proceedings, Conference on Frontiers of Combining Systems (FroCos), München, Germany.* Kluwer, 1996.

[Dahn *et al.*, 97] Bernd I. Dahn, Jürgen Gehne, Thomas Honigmann, and Andreas Wolf. Integration of automated and interactive theorem proving in ILF. In W. McCune, editor, *Proceedings, 14th International Conference on Automated Deduction (CADE)*, LNCS 1249, pages 57–60. Springer, 1997.

[Eder, 92] Elmar Eder. *Relative complexities of first order calculi.* Vieweg, 1992.

[Egly, 98] Uwe Egly. Cuts in tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume I: Foundations, pages 103–131. Kluwer, Dordrecht, 1998.

[Fitting, 96] Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, second edition, 1996.

[Hähnle and Schmitt, 94] Reiner Hähnle and Peter H. Schmitt. The liberalized $\delta$-rule in free variable semantic tableaux. *J. of Automated Reasoning*, 13(2):211–222, 1994.

[Letz, 98] Reinhold Letz. Clausal tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume I: Foundations, pages 43–72. Kluwer, Dordrecht, 1998.

[Miller, 84] Dale A. Miller. Expansion tree proofs and their conversion to natural deduction proofs. In R. E. Shostak, editor, *Proceedings, 7th International Conference on Automated Deduction (CADE), Napa, USA*, LNCS 170. Springer, 1984.

[Pfenning and Nesmith, 90] Frank Pfenning and Daniel Nesmith. Presenting intuitive deductions via symmetric simplification. In M. Stickel, editor, *Proceedings, 10th International Conference on Automated Deduction (CADE), Kaiserslautern, Germany*, LNCS 449, pages 336–350. Springer, 1990.

[Pfenning, 84] Frank Pfenning. Analytic and non-analytic proofs. In R. E. Shostak, editor, *Proceedings, 7th International Conference on Automated Deduction (CADE), Napa, USA*, LNCS 170, pages 394–413. Springer, 1984.

[Reif, 95] Wolfgang Reif. The KIV-approach to software verification. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages, and Tools for the Construction of Correct Software — Final Report*, LNCS 1009. Springer, 1995.

[Smullyan, 68] Raymond M. Smullyan. *First-Order Logic.* Springer, Heidelberg, 1968. Second corrected edition published in 1995 by Dover Publications, New York.

[Stenz, 97] Gernot Stenz. Beweistransformation in Gentzenkalkülen. Diplomarbeit, Fakultät für Informatik, Universität Karlsruhe, August 1997.

[Wolf, 94] Andreas Wolf. Optimization and translation of tableau-proofs into resolution. *J. of Information Processing and Cybernetics (EIK)*, 30(5–6):311–325, 1994.

[Wolf, 98] Andreas Wolf. A step towards a better understanding of automatically generated model elimination proofs. In J. Cuena, editor, *Information Technologies and Knowledge Systems (IT&KNOWS'98) – Proceedings of the XV. IFIP World Computer Congress*, pages 415–428. Austrian Computer Society/International Federation for Information Processing, 1998.

## Acknowledgements

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) within the Schwerpunktprogramm *Deduktion*. We would like to thank Ingo Dahn, Uwe Egly, and Reiner Hähnle for fruitful discussions, anonymous referees for useful comments on an earlier version of this paper, and Thomas Honigmann for his help in implementing an interface to the ILF system.