Efficient Representation and Computation of Tableaux Proofs

Klaus Schneider, Ramayya Kumar and Thomas Kropf

Institute of Computer Design and Fault Tolerance, University of Karlsruhe
(Prof. Dr. D. Schmid) P.O. Box 6980, W-7500 Karlsruhe, Germany,
e-mail: {schneider, kumar, kropf}@ira.uka.de

Abstract
The current first-order automatic prover FAUST, embedded in HOL, is based on a sequent calculus which is quite slow and memory intensive. In this paper, an improved version of FAUST using a modified form of tableau calculus called Tableau Graph Calculus is presented which overcomes the well-known inefficiencies of the traditional tableau calculus to a large extent. This calculus works on a compact representation of analytic tableaux called tableau graphs which are obtained by a preprocessing step which covers most of the rule applications of usual tableau calculus. This representation retains the clarity of the input formula and furthermore, its size is linear with respect to the length of the input formula. As a result of this preprocessing, our calculus has only one single rule which is repeatedly applied to obtain a proof. Many optimizations for the rule applications to effectively prune the search space are presented as well and are currently being implemented in a new version of FAUST.

Keyword Codes: I.2.3; F.4.1
Keywords: Deduction and Theorem Proving; Mathematical Logic

1 Introduction
The use of the theorem proving assistant HOL can be greatly enhanced by embedding an automatic first-order prover within it. Especially in the domain of Hardware Verification, the goals to be proven — when appropriate transformations have been undertaken — are mostly solvable using first-order techniques [ScKK91a], [KuKS91a]. To this effect, a prover called FAUST based on a modified form of sequent calculus has been integrated within HOL [ScKK91b], [KuKS91b]. This prover was quite effective in automating the process of Hardware Verification to a large extent. However, when the subgoals which result from the Hardware Verification Environment called MEPHISTO were very large,
the proofs were slow and highly memory intensive. This motivated us to improve the efficiency of FAUST. This has been achieved by using a calculus based on a modified tableau calculus, as reported in this paper.

The first-order prover to be embedded should be based on a calculus which lends itself to an easy integration in HOL. Sequent and tableau calculi allow easy translations of obtained proofs to natural deduction on which HOL is based, since their rules reflect the natural semantics of the connectives of the logic. Having experimented with a sequent calculus implementation [ScKK92], we decided to use an alternative approach based on tableau calculus.

Tableau calculi construct a tree labeled with signed formulae of an initial formula set whose inconsistency has to be shown. The rules of tableau calculi which are used to extend paths, can be classified into four types — $\alpha$, $\beta$, $\gamma$ and $\delta$-rules. $\alpha$- and $\beta$-rules are used to eliminate propositional connectives of the logic. $\beta$-rules branch a path but $\alpha$-rules do not. $\delta$-rules eliminate quantifiers by instantiating an arbitrary new variable. Thus $\alpha$, $\beta$ and $\delta$-rules can be applied only finitely often.

Since the tableaux as given in [Beth55],[Simul68],[Fitt90] correspond to trees, the number of nodes can grow exponential in terms of the length of the input formula. This effect takes place whenever several $\beta$-rules can be applied at the same stage, as shown in figure 1 (left side). After the application of one of those $\beta$-rules, e.g. the first one, the other $\beta$-rules have to be applied at each open path\(^1\).

\[ \text{Figure 1: Avoiding redundancy due to $\beta$-explosions} \]

However, the application of the first $\beta$-rule might have doubled the number of open paths\(^2\) thus enforcing the duplication of the remaining subformulae by further $\beta$-rule applications. In fact, the number of these copies can become exponentially large in analytic tableaux. It can however be observed that tableaux need not be stored completely, since they are highly redundant. The redundancy arising from the existence of simultaneously

\(^1\)There are two possibilities how rules can be applied: The first retains the formulae on which the rule is applied in the tableau and generates new subformulae, while the second one removes it from the tableaux (except for $\gamma$-rules). Using the second form of rule application, it is necessary to extend each open path by copies of subformulae resulting from other rule applications.

\(^2\)The number of open paths is not always doubled, since one of the paths could have been closed after the application of a $\beta$-rule.
applicable $\beta$-rules can be avoided by referring to the original subformulae instead of using copies as illustrated in figure 1 (right side).

In this paper, we define a calculus which is similar to tableau calculus, but works on such efficient representations of formulae and proofs. As this representation is even more flexible than usual tableau trees, further enhancements which speed up the proof process are possible. Some of these enhancements stem from connection calculi ([Andr81],[Bibe87]) to which the presented calculus is also related to. Thus, our calculus combines advantages of tableau calculus — namely the clarity and naturalness — and connection calculus — namely its efficient implementations. As the various features presented here are mainly independent from each other, one can choose an appropriate subset to implement a suitably efficient prover.

The paper is organized as follows: First, we describe an improvement of the representation of analytic tableaux developed by [Beth55],[Smul68] on which our tableau graph calculus is based. The definitions and the related soundness and completeness proofs are also given. The next section describes various algorithms for computing proofs within this calculus. Implementational details, mainly concerning extension-rules are briefly sketched in section 4. A short section concerning the implementation of our calculus in HOL90 with experimental results then follows. Finally, the paper is concluded with a summary.

## 2 The Tableau Graph Calculus $\mathcal{C}_{TG}$

In this section, we define the tableau graph calculus $\mathcal{C}_{TG}$, which uses graphs as shown in figure 1 as the underlying representation of the input formula. Before we formalize these ideas, we list the notation which is used (for detailed definitions see any logic textbook, for example [Fitt90]):

- $V$ set of object variables
- $T_{\Sigma}$ set of first order terms over a signature $\Sigma = (P_\Sigma, F_\Sigma, \alpha_\Sigma)$, where $P_\Sigma, F_\Sigma$ denote the set of predicate symbols and functions symbols, respectively and $\alpha_\Sigma : P_\Sigma \cup F_\Sigma \rightarrow \IN$ is the arity function
- $For_{\Sigma}$ set of first order formulae over a signature $\Sigma = (P_\Sigma, F_\Sigma, \alpha_\Sigma)$
- $T, F$ signs of tableau nodes
- $true, false$ truth values
- $S For_{\Sigma}$ set of signed formulae over a signature $\Sigma = (P_\Sigma, F_\Sigma, \alpha_\Sigma)$
- $[\varphi]_x^\alpha$ substitution of the variable $x$ in the term or formula $\varphi$ by $\tau$
- $\xi^\alpha_x$ modified assignment: $\xi^\alpha_x(x) := d$ and $\xi^\alpha_x(y) := \xi(y)$ if $x \neq y$
- $\omega^{D, I, \xi}$ evaluation function of an interpretation $(D, I)$ and an assignment $\xi$
- $\varphi(M)$ set of all subsets of a set $M$
- $\text{mgu}$ abbreviation for most general unifier
- $id$ the identity substitution
- $\circ$ composition of functions, that is $(f \circ g)(x) := f(g(x))$

The tableau graph calculus $\mathcal{C}_{TG} = (SS_{\Sigma}, CSS_{\Sigma}, \{\vdash^\alpha\})$ consists of a formal language $SS_{\Sigma}$, a subset $CSS_{\Sigma} \subset SS_{\Sigma}$ of it called the axioms and a rule $\vdash^\alpha$ which maps elements of the language $SS_{\Sigma}$ to other elements of it. In general, a proof in $\mathcal{C}_{TG}$ will proceed as follows:
First, for the given formula the corresponding element of \( \mathcal{SS}_\Sigma \) has to be computed, then the rule \( \vdash^\Sigma \) has to be applied until an element of \( \mathcal{CSS}_\Sigma \) is obtained. Implementations of \( \mathcal{C}_{TG} \) however work more sophisticatedly than described above.

First, we define the language \( \mathcal{SS}_\Sigma \) of our calculus:

**Definition 2.1 (Structure Sets \( \mathcal{SS}_\Sigma \))** The following rules define the set of structure sets \( \mathcal{SS}_\Sigma \) over a given signature \( \Sigma \):

1. \( \{\} \in \mathcal{SS}_\Sigma \)
2. \( \{\varphi\} \cup S \in \mathcal{SS}_\Sigma \), where \( \varphi \in SFor_\Sigma \) and \( S \in \mathcal{SS}_\Sigma \)
3. \( \{(S_1, S_2, \beta)\} \cup S \in \mathcal{SS}_\Sigma \), where \( S_1, S_2, S \in \mathcal{SS}_\Sigma \)
4. \( \{(S_1)^\mu\} \cup S \in \mathcal{SS}_\Sigma \), where \( S_1, S \in \mathcal{SS}_\Sigma \)

\((S_1, S_2, \beta)\) is called a \( \beta \)-structure and \((S)^\mu\) is called a \( \gamma \)-structure, where \( \mu \) is an arbitrary metavariable\(^3\). Elements of a structure set are called rows.

The next definition shows how structure sets can be computed for signed formulae:

**Definition 2.2 (Structure Set of a signed formula) \( \exists : SFor_\Sigma \rightarrow \mathcal{SS}_{\Sigma_{sh}} \) assigns to each signed formula \( \Phi \in SFor_\Sigma \), a structure set \( \exists(\Phi) \) according to the following rules\(^4\) \((\Sigma_{sh} \text{ is obtained by the introduction of new skolem function symbols in } \Sigma)\):

\[
\begin{align*}
\exists(TP(\tau_1, \ldots, \tau_n)) & := \{TP(\tau_1, \ldots, \tau_n)\} \\
\exists(FP(\tau_1, \ldots, \tau_n)) & := \{FP(\tau_1, \ldots, \tau_n)\} \\
\exists(T\varphi) & := \exists(F\varphi) \\
\exists(F\neg \varphi) & := \exists(T\neg \varphi) \\
\exists(T\varphi \land \psi) & := \exists(T\varphi) \cup \exists(T\psi) \\
\exists(F\varphi \land \psi) & := \{(\exists(F\varphi), \exists(F\psi))\} \\
\exists(T\varphi \lor \psi) & := \{(\exists(T\varphi), \exists(T\psi))\} \\
\exists(F\varphi \lor \psi) & := \{(\exists(F\varphi), \exists(F\psi))\} \\
\exists(T\forall x. \varphi) & := \{(\exists(T[\varphi]_x^\mu)\} \\
\exists(F\forall x. \varphi) & := \{(\exists(F[\varphi]_x^\mu)\} \\
\exists(T\exists x. \varphi) & := \{(\exists(T[\varphi]_x^\mu)\} \\
\exists(F\exists x. \varphi) & := \{(\exists(F[\varphi]_x^\mu)\} \\
\end{align*}
\]

In the rules above, \( \{\mu_1, \ldots, \mu_n\} \) is the set of metavariables of \( \varphi \), \( f \) is a new skolem-function symbol and \( \mu \) is a new metavariable.

Structure sets are a set-representation of the skolemized negation normal form of the given formula. Thus, provided that no equivalence or antivalence operator occurs, the following lemma holds:

**Lemma 2.1 (Size of Structure Sets)** The number of atomic formulae of the structure set \( \exists(\varphi) \) of each formula \( \varphi \in SFor_\Sigma \) is less or equal to the number of atomic formulae in \( \varphi \).

\(^3\)Unification can be introduced in tableau calculus by using Metavariabes (see [Fitt90], [Recev87], [ScKK91b], [ScKK92]). They can be seen as placeholders which are introduced by a \( \gamma \)-rule and are instantiated with suitable terms found with unification later on. Substitutions which assign terms to metavariables are called metasubstitutions.

\(^4\)Here we use the liberalized \( \delta \)-rule presented in [HaSc92] which takes only the free variables of the formula as arguments for the skolem function, instead of the free variables of a considered path.
Proof:
Easy by induction with Definition 2.2. As $\exists (T \psi \land \psi) = \exists (T \psi)$, the number of atomic formulae of the structure set $\exists (\varphi)$ can also be less than the number of atomic formulae of the input formula.

Example 2.1 (Structure Sets) 1. Let $\Phi_1$ be the formula:

$$\neg(-h \land (-[g \land m] \rightarrow [-g \land h]) \land ([\{g \rightarrow m\} \land \{\neg z \rightarrow h\}] \lor z)
\land (z \rightarrow [-g \land m]) \land (-h \rightarrow g))$$

$\Phi_1$ has the following structure set $\exists (\Phi_1)$:

$$\{\{Fh,\{Tg, Tm\}\}, \{\{Fh, Fg\}\}_{\beta}, (\{\{Fg\}, \{Tm\}\}_{\beta}, \{\{Tz\}\}_{\beta}, \{\{Tz\}\}_{\beta},
(\{Fz\}, \{Fg, Tm\}\), (\{Th\}, \{Tg\}\}_{\beta})\}
$$

2. Let $\Phi_2$ be the formula:

$$\neg([\forall x. (S(x, a) \lor R(b)) \land (Q(c, a) \lor U(x))]$$
$$\land [R(b) \lor [S(a, a) \land S(b, a)]) \rightarrow (-Q(c, a) \land \neg [U(a) \land U(b)])])$$

The structure set $\exists (\Phi_2)$ contains a $\gamma$-structure and a $\beta$-structure at the topmost level:

$$\{\{(\{TS(\mu, a)\}, \{TR(b)\})_{\beta}, (\{TQ(c, a)\}, \{TU(y)\})_{\beta}\}^\gamma,
(\{FR(b), \{FS(c, a)\}, \{FS(b, a)\}\})_{\beta},
\{FQ(c, a), ((FU(a))\}, \{FU(b)\})_{\beta}\}\}$$

To define the set of axioms $\mathcal{CS}_\Sigma$, we need the notion of the path set of a given structure set. To define it, the following operator is introduced: $\{p_1, \ldots, p_n\} \cup p := \{p_1 \cup p, \ldots, p_n \cup p\}$. $\cup$ extends all paths which are the elements of its left argument with the path which is its right argument.

Definition 2.3 (Path Set of a structure set) $\mathcal{PS} : \mathcal{SS}_\Sigma \rightarrow \wp(\wp(\mathcal{SFor}_\Sigma))$ assigns to each structure set $S \in \mathcal{SS}_\Sigma$ a set of paths $\mathcal{PS}(S)$. $\mathcal{PS}$ is defined as follows:

$$\mathcal{PS}(\emptyset) := \{\emptyset\}$$
$$\mathcal{PS}(\{p\} \cup S) := \{\{p\} \cup p \mid p \in \mathcal{PS}(S)\}$$
$$\mathcal{PS}(\{S_1, S_2\}) \cup S := \{\mathcal{PS}(S_1) \cup p \mid p \in \mathcal{PS}(S)\} \cup \{\mathcal{PS}(S_2) \cup p \mid p \in \mathcal{PS}(S)\}$$
$$\mathcal{PS}(\{(S_1)_\mu\} \cup S) := \{\mathcal{PS}(S_1) \cup p \mid p \in \mathcal{PS}(S)\}$$

Each element of $\mathcal{PS}(S)$ is called a path of the structure set. Given that $\Phi \in SFor_\Sigma$, then each element of $\mathcal{PS}(\exists (\Phi))$ is called a path of the formula $\Phi$.

Note that the path set of a structure set is a set of structure sets. The number of paths $\Pi(S)$ of a structure set $S$ can be computed as follows:
\[\Pi(\{\}\) := 1 \]
\[\Pi(\{\varphi\}) \cup S := \Pi(S) \]
\[\Pi(\{(S_1, S_2)_{\beta}\} \cup S) := (\Pi(S_1) + \Pi(S_2)) \cdot \Pi(S) \]
\[\Pi(\{(S_1)_{\gamma}\} \cup S) := \Pi(S_1) \cdot \Pi(S) \]

Example 2.2 (Path Sets) Some paths of \(3(\Phi_1)\) (see Example 2.1) are:

\[\{F_h, T_g, T_m, F_g, T_z, F_z, T_h\} \]
\[\{F_h, T_g, T_m, F_g, T_z, F_z, T_g\} \]
\[\{F_h, T_g, T_m, F_g, T_h, F_g, T_m, T_h\} \]

The set of axioms \(CSS_\Sigma\) of our calculus is characterized by the following definition:

Definition 2.4 (Closed Structure Set \(CSS_\Sigma\)) A structure set \(S\) is closed under the metasubstitution \(\sigma\) if and only if for each path \(p \in \mathcal{PS}(S)\) there exist signed formulae \(T\varphi_p, F\psi_p \in \sigma\) such that \(\sigma(\varphi_p) = \sigma(\psi_p)\). \((T\varphi_p, F\psi_p)\) is then called a complementary pair under \(\sigma\). The set of all closed structure sets of \(SS_\Sigma\) is denoted by \(CSS_\Sigma\).

![Diagram of structure sets into graphs and the resulting rows and columns](image)

Figure 2: Mapping of structure sets into graphs and the resulting rows and columns

Structure sets can be pictorially represented by graphs called tableau graphs. Therefore a function \(TG\) has to be defined which assigns a graph to a given structure set. To define \(TG\), a given structure set \(S \neq \{\}\) is split in the subsets \(\{\varphi_1, \ldots, \varphi_p\}\) of signed formulae, \(\{(\psi_1)_{\gamma_1}^{\varphi_1}, \ldots, (\psi_q)_{\gamma_q}^{\varphi_q}\}\) of \(\gamma\)-structures and \(\{((\xi_1, \eta_1)_{\beta_1}, \ldots, (\xi_r, \eta_r)_{\beta_r})\}\) of \(\beta\)-structures (which is always possible due to definition 2.1). Then figure 2 (left side) shows such a
pictorial representation of a tableau graph $\mathcal{T}\mathcal{G}(S)$ of $S$. If $S$ is empty, then $\mathcal{T}\mathcal{G}$ assigns the empty graph to it. It should be noted that the nodes corresponding to $\gamma$- and $\beta$-structures in the tableau graph themselves contain tableau graphs.

The mapping given in figure 2 (left side) is not well-defined, since structure sets provide no fixed order on their elements. By grouping all tableau graphs which can be built from a structure set $S$ with a specific ordering, one can define an equivalence relation such that all graphs in that group form an equivalence class. All graphs of an equivalence class differ only in the order of their literals, their $\gamma$-structures and their $\beta$-structures. An arbitrary element of such an equivalence class corresponding to a structure set of a formula $\Phi$ is called a tableau graph for $\Phi$.

In a tableau graph representation, there are at most two columns resulting from the $\beta$-structures and arbitrarily many rows corresponding to the rows of the underlying structure set as shown in figure 2 (right side). Rows have to be interpreted as conjunctively connected and columns represent disjunctive connections. The truth signs $T$ and $F$ represent positive and negative literals, respectively.

![Figure 3: Examples of tableau graphs](image-url)

**Example 2.3 (Examples of tableau graphs)** Figure 3 shows possible tableau graphs for $F\Phi_1$ and $F\Phi_2$. The graph for $\Phi_1$ has 40 paths and the graph for $\Phi_2$ has 16 paths. The left graph is closed under id, thus it is an axiom and the formula $\Phi_1$ is proven. The graph
for $\Phi_2$ cannot be closed though $\Phi_2$ is valid, too. Here some rule applications of the only rule $\vdash^\varepsilon$ defined below are necessary.

**Definition 2.5 (Extension Rule)** The extension relation $\vdash^\varepsilon$ on the set of all structure sets $\mathcal{S}_S$ is defined by:

1. $\{(S_1)^\mu\} \cup S \vdash^\varepsilon \{(S_1)^\mu, ([S_1]_\mu)^{\overline{\mu}}\} \cup S$, where $\overline{\mu}$ is a new metavariable
2. $\{(S_1, S_2)_\beta\} \cup S \vdash^\varepsilon \{(\overline{S_1}, S_2)_\beta\} \cup S$, if $S_1 \vdash^\varepsilon \overline{S_1}$
3. $\{(S_1, S_2)_\beta\} \cup S \vdash^\varepsilon \{(S_1, \overline{S_2})_\beta\} \cup S$, if $S_2 \vdash^\varepsilon \overline{S_2}$

In other words, an extension of a structure set includes a variant of an arbitrary $\gamma$-structure of it, obtained by replacing the bound metavariable $\mu$ by $\overline{\mu}$. This $\gamma$-structure variant can be generated from a $\gamma$-structure which is not necessarily an element of the structure set, but can also be generated from a $\gamma$-structure within nested structure sets due to item 2. and 3. of the definition. One extension step in $\mathcal{C}_{TG}$ corresponds to the application of a $\gamma$-rule and all the subsequent $\alpha-$, $\beta-$ and $\delta$-rules applicable to the obtained formula in tableau calculus.

![Figure 4: Extension of a tableau graph](image)

**Definition 2.6 (Derivation in $\mathcal{C}_{TG}$)** A derivation of $S_1$ in $\mathcal{C}_{TG}$ is a finite list $S_1, \ldots, S_n$ of structure sets, such that the following conditions hold:

- There is a metasubstitution $\sigma$ such that $S_n$ is closed under $\sigma$
\[ S_i \vdash S_{i+1}, \text{ for all } i \in \{1, \ldots, n-1\} \]

\( n \) is called the length of the derivation.

It should be noted that for each formula of propositional logic, the structure set does not contain a \( \gamma \)-structure, thus no extension rule can be applied. Hence, proofs for propositional logic have just to check if the original path set itself is closed under \( id \).

In order to prove soundness and completeness results, we have to define the semantics of structure sets. Lemma 2.2 lists the essential properties of structure sets, which are needed to show the correctness.

**Definition 2.7 (Semantics of Structure Sets)** Given an interpretation \( (D, I) \) and an assignment \( \xi : V \rightarrow D, \Omega^{D, I, \xi} : SS_\Sigma \rightarrow \{\text{true, false}\} \) assigns each structure set a truth value according to the following rules:

1. \( \Omega^{D, I, \xi}(\{\}) := \text{true} \)

2. \( \Omega^{D, I, \xi}(\{T \varphi \} \cup S) := \begin{cases} \text{true} : & \omega^{D, I, \xi}(\varphi) = \text{true} \text{ and } \Omega^{D, I, \xi}(S) = \text{true} \\ \text{false} : & \text{otherwise} \end{cases} \)

3. \( \Omega^{D, I, \xi}(\{F \varphi \} \cup S) := \begin{cases} \text{true} : & \omega^{D, I, \xi}(\varphi) = \text{false} \text{ and } \Omega^{D, I, \xi}(S) = \text{true} \\ \text{false} : & \text{otherwise} \end{cases} \)

4. \( \Omega^{D, I, \xi}(\{(S_1, S_2)\} \cup S) := \begin{cases} \text{true} : & \Omega^{D, I, \xi}(S_1) = \text{true and } \Omega^{D, I, \xi}(S) = \text{true}\text{ or } \Omega^{D, I, \xi}(S_2) = \text{true and } \Omega^{D, I, \xi}(S) = \text{true} \\ \text{false} : & \text{otherwise} \end{cases} \)

5. \( \Omega^{D, I, \xi}(\{(S_i)\} \cup S) := \begin{cases} \text{true} : & \Omega^{D, I, \xi}(S_1) = \text{true for each } d \in D \\ \text{false} : & \text{otherwise} \end{cases} \)

**Lemma 2.2 (Properties of Structure Sets)** For each \( \varphi \in \text{For}_\Sigma, S \in SS_\Sigma \) and each interpretation \( D, I \) with assignment \( \xi \) the following holds:

1. \( \exists (T \varphi) \) is satisfiable iff \( \varphi \) is satisfiable.

2. \( \exists (F \varphi) \) is unsatisfiable iff \( \varphi \) is valid.

3. \( \Omega^{D, I, \xi}(S) = \text{true implies that for each } \sigma \ \omega^{D, I, \xi}(\bigvee_{i=1}^{n} \bigwedge_{j=1}^{i} \sigma(\text{sgn}(\varphi_{i,j}))) = \text{true where} \)

\[ \mathcal{P}S(S) = \{\{\varphi_{i,1}, \ldots, \varphi_{i,i}\} \mid i \in \{1, \ldots, n\}\} \text{ and } \text{sgn}(\varphi) := \begin{cases} \psi : & \text{if } \varphi = T \psi \\ \neg \psi : & \text{if } \varphi = F \neg \psi \end{cases} \]

4. \( \Omega^{D, I, \xi}(S) = \text{false holds if there is a substitution } \sigma \text{ such that for each path } p \in \mathcal{P}S(S) \) \( \Omega^{D, I, \xi}(\sigma(p)) = \text{false holds.} \)

5. \( S \) is unsatisfiable if there is a substitution \( \sigma \) such that \( S \) is closed under \( \sigma \).
6. $S_1 \vdash^c S_2$ implies $\Omega^{D, I, \xi}(S_1) = \Omega^{D, I, \xi}(S_2)$.

Proof sketch:
1 is obtained by the usual skolemization theorem and structural induction. 2 is equivalent to 1. 3 can be easily proven by structural induction with definition 2.7 and a substitution lemma. 4 follows from 3.
If a structure set $S$ is closed under a substitution $\sigma$ then each path $p$ of it is unsatisfiable\(^5\). Hence 5 follows from 4. The proof of 6 is based on bounded renaming and idempotency of conjunction.

\[\square\]

Theorem 2.1 (Soundness and Completeness) $\varphi$ is valid iff there is a derivation for $\mathcal{Z}(F\varphi)$ in $\mathcal{C}_{TG}$.

Proof sketch (The numbers used below refer to lemma 2.2)
Correctness: Assume there is a derivation $\mathcal{Z}(F\varphi) \vdash^c S_1 \vdash^c \ldots \vdash^c S_n$, then $S_n$ is closed under a substitution $\sigma$ and because of 5 it is unsatisfiable. By repeated use of 6 one can conclude, that $\mathcal{Z}(F\varphi)$ is also unsatisfiable, and by 2, that $\varphi$ has to be valid.
Completeness: The completeness follows from the theorem of G"odel-Herbrand-Skolem and the compactness theorem of first-order logic, but can also obtained from the completeness of tableau calculus in the following way: Without loosing completeness, we remove the signed formula on which a rule is applied from the tableau unless it is a $\gamma$-formula. Moreover, allowing the tableau only to be atomically closed also retains the completeness in tableau calculus. Now we define a homomorphism $\Theta$ which assigns each tableau a tableau graph by the following rules: The tableau graph $\Theta(T_1)$ for the initial tableau $T_1$ is an arbitrary tableau graph built from the structure set of the input formula. Applications of $\alpha$-, $\beta$-, $\delta$- and the first applications of $\gamma$-rules do not change the assigned tableau graph. Further applications of $\gamma$-rules in tableau calculus correspond to extension steps in $\mathcal{C}_{TG}$. Then we can show, that if the tableau is atomically closed under a substitution $\sigma$, the corresponding tableau graph is also closed under $\sigma$. As for each valid formula $\varphi$ a closed tableau described as above for $F\varphi$ exists (since tableaux calculus is complete), the completeness is proven.

\[\square\]

3 Algorithmic aspects

In order to prove the validity of a given formula $\varphi \in For_{\Sigma}$ in $\mathcal{C}_{TG}$, one has to compute the structure set $\mathcal{Z}(F\varphi)$ and to check if it can be closed under a certain substitution $\sigma$. If it can be closed, the proof is finished, if not, an extension rule has to be applied and again the closure has to be checked and so on. Theorem 2.1 guarantees, that if the extension

\(^5\)Assuming the satisfiability of $p$ means that there must be $D, I$ such that $\Omega^{D, I, \xi}(\varphi) = \text{true}$ for each $\varphi \in p$ under each assignment $\xi$. Using an arbitrary assignment $\xi$ with the property $\xi(\mu) := \xi(\sigma(\mu))$ leads then to a contradiction.
rule is applied in a fair manner, the process terminates yielding in a finite derivation of the valid formula.

While the computation of the structure set is deterministic and can be done in a time linear to the length of the input formula (if no equivalence or antivalence connective occurs) (cf. lemma 2.1), checking the closure of the structure set is much more complex (because of the \( \mathcal{NP} \)-completeness of the whole problem). Moreover, the extension rule introduces an indeterminism, since the \( \gamma \)-structure from which a variant should be included is not specified. In this section we describe, how checking the closure can be enhanced by precomputation of complementary pairs and reordering of rows in the tableau graph\(^6\). In the next section we give some improvements in order to remove some of the indeterminism and the complexity of extension steps.

Several proof procedures can be obtained by combining some of the following items which are explained later:

- depth-first or breadth-first path-traversal
- backtracking or collecting all possible unifiers in a set \( \mathcal{U} \)
- reordering of rows or using a fixed order

Let us first describe a non-backtracking depth-first-traversal without reordering of rows: The procedure starts with a tableau graph using an arbitrary, but fixed order and the unifier set \( \mathcal{U}_0 := \{ id \} \). The paths are traversed in a depth-first manner from left to right until the considered path \( p \), instantiated with a unifier \( \sigma \in \mathcal{U}_n \), contains a complementary pair under \( \varphi \). Then the path is left and the unifier set is updated by composing the unifiers found so far with the unifiers which cause the path to become complementary as follows:

\[
\mathcal{U}_{n+1} := \{ \varphi \circ \sigma \mid \sigma \in \mathcal{U}_n \land \exists \mathcal{F} \varphi, \mathcal{T} \psi \in p, \varphi = \text{mgu}\{ \sigma(\varphi), \sigma(\psi) \} \}.
\]

In other words, all possibilities of closing the path \( p \) proceeding from the unifier set \( \mathcal{U}_n \) are saved in \( \mathcal{U}_{n+1} \). If there is a unifier \( \sigma' \in \mathcal{U}_n \) which has no refinement \( \varphi' \) but other unifiers of \( \mathcal{U}_n \) can be refined, then \( \sigma' \) is omitted in \( \mathcal{U}_{n+1} \).

Instead of collecting the most general unifiers which close the path set traversed so far, one can also choose a unifier among them. If this unifier can be refined to a substitution under which the whole path set is closed, a proof is obtained, otherwise one has to backtrack after a given number of extensions and choose another unifier (backtracking depth-first-traversal without reordering of rows).

Instead of a depth first traversal of the paths, one can also use a breadth-first traversal: Beginning with the root node of the tableau graph all paths are traversed simultaneously until all paths can be closed under a substitution \( \sigma \). Unlike the depth-first approaches presented above, this procedure is complete, but it turns out to be too slow for practical use since all paths are traversed further even if some of them could already be closed (breadth-first traversal without reordering of rows).

\(^6\) Precomputation of complementary pairs and reordering are concepts used in connection calculi [Bibe87], but can also be found in tableau calculi [Wrig84].
While the procedures described so far use a given ordering of the tableau graph, one can reorder the tableau graph during the traversal in order to close paths as soon as possible. Since a complementary pair indicates that each path containing it is complementary, a single complementary pair can close many paths. In general, the closer the complementary pair lies to the root, the larger is the number of paths it closes, thus the rows should be reordered\(^7\). As the complementary pairs now control the path traversal, reordering requires that all complementary pairs must be known before the paths are traversed. This precomputation of complementary pairs can be done in polynomial time\(^8\). In this step, the tableau graph is extended by links labeled with a most general unifier which makes the formulae connected by that link complementary (this is usually done in connection calculi, too).

Having computed all complementary pairs and the unifiers which make them complementary, we reorder the rows of the tableau graph in such a manner that complementary nodes clash together as soon as possible.

For example consider the depth-first traversal without backtracking using reordering: Again we start with the unifier set \(U_0 := \{ id \}\) and a leftmost path \(p\) in an arbitrary row \(R_1\) of the tableau graph which contains at least one signed formula \(\varphi\) which is part of a complementary pair \((\varphi, \psi)\). If \(\psi \in p\), too, then the path can be closed, otherwise there is another row \(R_2\) in the graph which contains \(\psi\). Thus, we exchange rows such that \(R_2\) immediately follows \(R_1\) in order to bring the complementary pair \(\varphi, \psi\) as near as possible to each other. As \(R_2\) contains again a structure set which has rows itself, the reordering has to proceed recursively in \(R_2\) such that the row which contains \(\psi\) will be the first one in \(R_2\), and so on.

A possible proof of the formula \(\Phi_1\) given in section 2 is shown in figure 5. First, the root node is taken into account. Two links are found, one connecting the root node with the node labeled with \(Th\) in the third row and another connecting the root node with the node labeled with \(Th\) in the fifth row. We decided to use the fifth row and therefore we have to exchange the second and the fifth row of the graph. This situation is given in the leftmost graph in figure 5. As one of the two paths leaving from the root is immediately closed, the number of paths to consider is halved after this step. Taking the remaining path \(\{Th, Tg\}\) into account, we find a link \((Tg, Fg)\) between the second and the fourth row in the leftmost graph in figure 5, thus we exchange the third and the fourth row to obtain the graph as shown in the middle of figure 5. The contradiction on the closed path \(\{Th, Tg, Fg, Tm\}\) halves again the remaining path set, thus the number of paths is reduced to a fourth by these two steps. In the graph in the middle of figure 5, we choose

\(^7\)The reordering of rows is correct since rows are conjunctively connected and the conjunction is an associative and commutative operation.

\(^8\)Assume the structure set of a formula of length \(n\) contains \(p\) atomic formulae (nested arbitrarily deep) and \(l_1, \ldots, l_p\) are the lengths of these atomic formulae, thus \(\sum_{i=1}^{p} l_i \leq n\) and \(p \leq n\). If all literals are unified with each other, and the unification of \(l_i\) and \(l_k\) requires a time bounded by \(c \cdot (l_i + l_k)\), the time required for all unifications is

\[
\sum_{i=1}^{p} \sum_{k=1}^{p} c \cdot (l_i + l_k) = \sum_{i=1}^{p} c \cdot p \cdot l_i \cdot \left( \sum_{k=1}^{p} l_k \right) = c \cdot p \cdot \left( \sum_{i=1}^{p} l_i \right) \cdot \left( \sum_{k=1}^{p} l_k \right) \leq c \cdot p \cdot n \cdot n \leq c \cdot n^3
\]
the fourth row as the next row to be visited, thus no exchanges have to be performed. The rightmost graph in figure 5 shows which paths are finally visited. It has to be noted that just 6 of the 40 paths of the structure set of the formula have been visited!

Moreover, it has to be pointed out, that the last row which stems from the formula \( \{g \land m\} \lor \{\neg h \land \neg g\} \) is not needed for the proof. The tableau graph calculus makes use of this fact since the unnecessary branches are not considered.

To summarize, the proof process is split into three processes:

1. Construction of the tableau graph, i.e. the structure set
2. Computation of all complementary pairs and the corresponding unifiers
3. Checking the closure using the reordering technique, eventually apply the extension rule and check again.

As rule application in tableau calculus corresponds with path traversal in tableau graph calculus, this reordering of rows can be interpreted as a postponed rule application of those \( \beta \)-rules in tableau calculus which could have been applied at the same situation, in order to optimize the tableau length. Of course, this strategy will not always find a shortest tableau, but the proof length is reduced drastically.

Paths of analytic tableau can be closed during their construction, so the complementary pair need not necessarily be atomic. Therefore, some big formulae of simple structure (for example \( \Theta \lor \neg \Theta \)) can be proved in constant time regardless of the length of the formula. Since non-atomic closure is not available in tableau graph calculus, the tableau graph
calculus can not $p$-simulate ([Eder92]) the tableau calculus. However, definitional forms ([Eder92]) can simulate the non-atomic closure in $C_{TG}$, as each subformula is replaced by a new atomic formula which can then become part of a complementary pair. On the other hand, it can be observed, that the path set of the definitional form can be exponentially larger than the path set of the original formula.

4 Implementational details concerning extensions

In tableau calculus $\gamma$-rules are treated in a special manner. Unlike the other rules, $\gamma$-rules can be applied arbitrarily often. Therefore, some special treatment of these rules is necessary in order to compute short proofs. As the $\gamma$-rule of tableau calculus corresponds with the extension rule of $C_{TG}$, the same holds for the extension rule. We have implemented a prover based on tableau graph calculus, using various improvements mainly concerning the extension-rule, which are given in this section.

**Fairness.** After a finite number of extensions of a $\gamma$-structure, other $\gamma$-structures have to be extended. In other words, no $\gamma$-structure can be preferred infinitely often.

**Unification.** The $\gamma$-rule in tableau calculus allows the instantiation of an arbitrary term, but usually only specific terms allow the closure of the tableau path. Therefore tableau calculi with unification have been defined (see [Fitt90, Reev87, ScKK92]). Mainly there are two approaches: in Fitting’s approach, the $\delta$-quantifiers are eliminated by the introduction of skolem-functions, while in the other two approaches, the $\delta$-rules introduce new eigenvariables\(^9\). Since the $\delta$-rule of tableau calculus requires that the introduced eigenvariable has to be new, each metavariable $m$ has to be attributed with a restriction list $fs_m$ for calculating the terms which are allowed for substitution. In other words, a term $\tau$ can be instantiated for a metavariable $m$ if and only if no variable of $fs_m$ occurs in $\tau$. The substitution is then called allowed.

The two approaches are dual in sense, but the second one can be implemented more efficiently. If we do not use liberalized $\delta$-rules, then it can be seen, that each restriction list $fs_m$ contains all eigenvariables which are introduced after the metavariable $m$. If the eigenvariables are encountered in the order of their introduction, then each restriction list has the form $\{y_{r_m}, y_{r_m+1}, \ldots\}$, where $r_m$ is the index of the eigenvariable which are introduced immediately after $m$. Thus the only information which is relevant for restrictions is the restriction index $r_m$ which can be seen as an attribute of $m$. Using restriction indices, a term is called allowed iff the maximum index of all eigenvariables occurring in it is less than the restriction index of the metavariable.

In Fitting’s approach the dependency of the metavariables and eigenvariables is expressed by using a skolem function. The allowedness of a term is detected by the occur-check of the used unification algorithm. Since the terms instantiated by $\delta$-rules can become arbitrary large and the occur check is known to be a frequently used operation, we suppose that the approach using restrictions is more efficient.

\(^9\)The variables introduced by $\delta$-rules are called eigenvariables.
Both approaches carry over to tableau graph calculus. In section 2 we gave the definitions of the calculus with Fitting's approach, the definition of the calculus with the other approach is easily obtained from [Reev87], [ScKK92].

\textbf{γ-templates.} In general several extensions of a γ-structure are necessary to close the structure set.

If all γ-structures have been extended once, but the tableau graph is not closed under a substitution, another extension of a γ-structure is required. Since the subgraph of this extension has exactly the same structure as the original one, only a template is stored for all extensions which is parametrized with the metavariable to be introduced (cf. figure 4). For this reason, a new sort of node is introduced, namely the γ-templates, which represent all subgraphs which stem from extensions of the same γ-structure. These nodes are parametrized with some offset informations needed for making extensions. Thus, applying further extensions this way reduces storage requirements, but of course, γ-templates do not decrease the number of paths.

\textbf{Multiple γ-templates.} If two or more γ-quantifiers immediately follow each other (for example $\forall x.\forall y.\varphi(x, y)$), then only one γ-template will be created for them. Of course, this does not affect the completeness of our calculus, since applying an extension rule on the outermost γ-structure implies applying an extension on the inner γ-structure as well. Hence this is in principle not one extension-rule, but a multiple one. The advantage is, that the number of applicable extension rules decreases and this way the search space is pruned. Since applying extensions on outermost γ-structures does not require more effort than applying them on inner ones as their subgraph is exactly the same, neither the number of nodes nor the number of paths is increased by using these multiple γ-templates.

\textbf{Antiprenexing.} Each time an extension is made, the number of paths to close is decreased. Thus, the scopes of all quantifiers should be as small as possible in order to obtain γ-templates with a minimal number of paths. For this reason, we use in our prover a technique called antiprenexing which is used in other provers as well.

\textbf{Management of extension-rules in a stack.} Since the repeated use of an extension-rule on a γ-structure is simulated by using offsets of the corresponding template graph, all extensions of a γ-structure are, at a first glance, forced to immediately follow each other. It is however possible that the extension is required only for the closure of one path, but now the extension is ‘visible’ for each path running through the γ-template. Therefore the number of paths running through the γ-template is multiplied by the number of paths of the subgraph inside the γ-template, each time an extension is made.

To get rid of this unsatisfying situation, the following strategy is used: After a path $p$ which requires a further extension is detected (this can only be the case at the end of $p$), the extension-rule application is simulated by adding an appropriate offset in a γ-template on $p$. The traverse control now dives into the γ-template as if this template would be at the end of $p$. Assume that the considered paths (the γ-template can branch $p$ into more paths which have to be closed now) can now be closed, the extension will be removed
by simply deleting the offset information in the template. This way the extension-rule application is actually done at the end of \( p \) although the template is somewhere else. As several extensions can be required, and the \( \gamma \)-templates added to \( p \) are managed in a FIFO-manner, it turns out, that the \( \gamma \)-template traversals are similar to subprogram calls and can be handled similarly by a stack.

**Local metavariables and their scopes.** As a metavariable occurs only in the \( \gamma \)-template which introduced it, the \( \gamma \)-template is the scope of the metavariable. After leaving an instance of a \( \gamma \)-template, the instances of the metavariables introduced there can be removed of each unifier of the unifier set \( \mathcal{U}_n \) found so far, since they are not relevant for the other paths to be searched through. This has the effect, that different unifiers in the set \( \mathcal{U}_n \) may become equal and therefore the number of unifiers shrinks down. Thus, treating metavariables locally within their scopes has drastic effects on the time needed for the proof process.

**Choice of next extension rule.** If a path cannot be closed, the question arises, which one of the \( \gamma \)-templates on the path might be a good candidate for a further instantiation. Fairness has to be guaranteed, but it is inefficient to use all \( \gamma \)-templates in a queue, since the number of extensions of each \( \gamma \)-template is then exactly the same whether an extension of it is useful or not. However, this is not required by fairness.

If a path cannot be closed, then the information found so far may give some hints which can be used to choose the next \( \gamma \)-template for an extension. For example, if a unifier was deleted due to restrictions or an occur check, the \( \gamma \)-template which introduced the metavariable whose restriction was violated is a good candidate for a further instantiation, because an extension introduces a new metavariable \( \overline{\mu} \) which has no restriction, thus the deleted unifier may become allowed with \( \overline{\mu} \) instead of \( \mu \). Other good candidates are \( \gamma \)-templates, for which different unifiers suggest different instantiations of the introduced metavariable, though one might fail with this suggestion also, e.g. \((\exists x.Px) \land (\exists y.Qy) \rightarrow (\exists z.Pz \land Qz)\).

Some of the information needed for the decisions described above can be precomputed in the second step: For example, unifiers which are not allowed are not thrown away, but stored with a flag, so that they are available for extensions in which they may become allowed.

**Stopping criteria.** Predicate logic is only semidecidable, but special formula fragments are nevertheless decidable. In general, some specific stopping criteria can be formulated. For example, if no unifier has been deleted since it was not allowed, it will be of no use to work further at all. Another stopping criteria is, that unifiers suggest only terms which are already instantiated in the considered \( \gamma \)-rule.

**Merging/Lemma generation.** The enhancements usually called *merging* and *lemma generation* [Brod91] will also be integrated in our calculus. Lemma generation is based on the asymmetric \( \beta \)-rules given below:
Both rules are correct due to the absorption theorems (\(\varphi \lor \psi \leftrightarrow (\varphi \lor (\neg \varphi \land \psi))\) and (\(\varphi \lor \psi \leftrightarrow ((\varphi \land \neg \psi) \lor \psi)\)).

Merging is done by removing each path \(p_2\) which contains another path \(p_1\). This is correct since closing \(p_1\) implies closing of the larger path \(p_2\). Both techniques can influence the length of shortest proofs drastically: There are examples which can be proven in a time of \(O(2^k)\) without merging or lemma generation, while proofs of length \(O(k)\) exist when merging or lemma generation is used.

## 5 Embedding \(C_{TG}\) in HOL

The procedures corresponding to the various steps in the proof process (cf. section 3) are being implemented in SML within HOL90. Similar to the previous implementation of FAUST, there is a fast modus in which the theorems are smuggled into HOL using ‘mk_hm’ and a slower modus in which the proofs are automatically translated into a HOL-tactic or a HOL-rule. The tactic generation procedure was earlier implemented using a backward proof technique and a new implementation present in the public-domain version of FAUST uses a forward proof technique, which is much faster than the former. Additionally, we are currently implementing another version which converts the original higher-order goal into a propositional goal and then solves it using a propositional proof tactic. The propositional goal can be obtained by substituting the appropriate instantiations (this can be more than one for a quantifier) for the quantified variables, thereby removing the quantifications.

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\beta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\beta_1)</td>
<td>(\beta_2)</td>
</tr>
<tr>
<td>(-\beta_2)</td>
<td>(\beta_1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(p_1)</th>
<th>(\mathcal{F}_{new})</th>
<th>(\mathcal{F}_{old})</th>
<th>(p_{13})</th>
<th>(\mathcal{F}_{new})</th>
<th>(\mathcal{F}_{old})</th>
<th>(p_{29})</th>
<th>(\mathcal{F}_{new})</th>
<th>(\mathcal{F}_{old})</th>
<th>(p_{36})</th>
<th>(\mathcal{F}_{new})</th>
<th>(\mathcal{F}_{old})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_1)</td>
<td>3</td>
<td>20</td>
<td>6</td>
<td>40</td>
<td>(P_{29})</td>
<td>22</td>
<td>120</td>
<td>(P_{36})</td>
<td>21</td>
<td>800</td>
<td></td>
</tr>
<tr>
<td>(P_2)</td>
<td>2</td>
<td>(&lt;10)</td>
<td>8</td>
<td>50</td>
<td>(P_{26})</td>
<td>29</td>
<td>39</td>
<td>(P_{37})</td>
<td>43</td>
<td>720</td>
<td></td>
</tr>
<tr>
<td>(P_3)</td>
<td>1</td>
<td>(&lt;10)</td>
<td>4</td>
<td>10</td>
<td>(P_{27})</td>
<td>15</td>
<td>220</td>
<td>(P_{38})</td>
<td>387</td>
<td>8850</td>
<td></td>
</tr>
<tr>
<td>(P_4)</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>(&lt;10)</td>
<td>(P_{28})</td>
<td>18</td>
<td>210</td>
<td>(P_{39})</td>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>(P_5)</td>
<td>2</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>(P_{29})</td>
<td>27</td>
<td>370</td>
<td>(P_{40})</td>
<td>17</td>
<td>6410</td>
<td></td>
</tr>
<tr>
<td>(P_6)</td>
<td>1</td>
<td>(&lt;10)</td>
<td>4</td>
<td>40</td>
<td>(P_{30})</td>
<td>10</td>
<td>90</td>
<td>(P_{41})</td>
<td>22</td>
<td>1670</td>
<td></td>
</tr>
<tr>
<td>(P_7)</td>
<td>1</td>
<td>(&lt;10)</td>
<td>10</td>
<td>140</td>
<td>(P_{31})</td>
<td>9</td>
<td>100</td>
<td>(P_{42})</td>
<td>18</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>(P_8)</td>
<td>1</td>
<td>10</td>
<td>12</td>
<td>110</td>
<td>(P_{32})</td>
<td>18</td>
<td>160</td>
<td>(P_{43})</td>
<td>(*)</td>
<td>(*)</td>
<td></td>
</tr>
<tr>
<td>(P_9)</td>
<td>5</td>
<td>60</td>
<td>11</td>
<td>160</td>
<td>(P_{33})</td>
<td>21</td>
<td>220</td>
<td>(P_{44})</td>
<td>16</td>
<td>450</td>
<td></td>
</tr>
<tr>
<td>(P_{10})</td>
<td>7</td>
<td>80</td>
<td>8</td>
<td>70</td>
<td>(P_{34})</td>
<td>305</td>
<td>11710</td>
<td>(P_{45})</td>
<td>153</td>
<td>3590</td>
<td></td>
</tr>
<tr>
<td>(P_{11})</td>
<td>1</td>
<td>(&lt;10)</td>
<td>7</td>
<td>50</td>
<td>(P_{35})</td>
<td>3</td>
<td>70</td>
<td>(P_{46})</td>
<td>113</td>
<td>106850</td>
<td></td>
</tr>
<tr>
<td>(P_{12})</td>
<td>19</td>
<td>150</td>
<td>27</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Runtimes of the Pelletier benchmarks

Table 1 compares the proof times of the theorems of [Pell86] required by the old FAUST (\(\mathcal{F}_{old}\)) and the new version (\(\mathcal{F}_{new}\)). All times are given in milliseconds and are an average value of ten proofs.
6 Summary

A variant of the tableau calculus has been presented in this paper, which is based on an efficient representation obtained by a preprocessing step. Since most of the rules of traditional calculus are implicitly applied during this step, only a single rule is necessary for obtaining proofs. Efficient strategies for applying this rule have also been described.

The underlying concepts retain the advantages of tableau calculus — retention of the input formula structure and the clarity of proofs — and additionally yield more efficient implementations. Since the size of the representation used for proofs is linear in terms of the length of the formula, more complex problems can be handled efficiently.

The implementational aspects of the calculus, especially the handling of extension-rules have been elaborated. New improvements, made possible due to the underlying graph representation, have also been embedded in our calculus.

References


[Eder92] Eder E., Relative Complexities of First Order Calculi, Vieweg Verlag, 1992

[Fitt90] Fitting M.C., First-Order Logic and Automated Theorem Proving, Springer Verlag, New York, 1990


[Reev87] Reeves S.V., *Semantic tableau as a framework for automated theorem proving*, Department of Computer Science and Statistics, Queen Mary College, Univ. of London, 1987
