Connection-based Theorem Proving in Classical and Non-classical Logics

Christoph Kreitz
Department of Computer Science, Cornell-University
Ithaca, NY 14853-7501, U.S.A.
kreitz@cs.cornell.edu

Jens Otten
FG Intellektik, FB Informatik, Technische Universität Darmstadt
Alexanderstr. 10, 64283 Darmstadt, Germany
jeotten@intellektik.informatik.tu-darmstadt.de

Abstract: We present a uniform procedure for proof search in classical logic, intuitionistic logic, various modal logics, and fragments of linear logic. It is based on matrix characterizations of validity in these logics and extends Bibel's connection method, originally developed for classical logic, accordingly. Besides combining a variety of different logics it can also be used to guide the development of proofs in interactive proof assistants and shows how to integrate automated and interactive theorem proving.

1 Introduction

Classical and non-classical logics are used extensively in various branches of Artificial Intelligence and Computer Science as logics of knowledge and belief, logics of programs, logics of action and change, and for the specification of distributed and concurrent systems. In many of these applications there is a need for deductive systems that can simulate mathematical reasoning. Proof assistants like NuPRL [Constable *et al.*, 1986], Coq [Dowek, 1991], Alf [Altenkirch *et al.*, 1994], Isabelle [Paulson, 1990] were developed to support interactive formal reasoning in a humanly comprehensible form and to automate it to a certain extent. But since they are are based on sequent or natural deduction calculi, the proof *search* strategies in these systems have to follow the connectives when analyzing a logical formula, which makes the degree of automation very low. On the other hand, the success of theorem provers for classical logic [Wos *et al.*, 1990, Letz *et al.*, 1992, Bibel *et al.*, 1994, Beckert and Posegga, 1994] has shown that formal reasoning can be automated sufficiently well.

Matrix-based proof search procedures like the *connection method* [Bibel, 1981, Bibel, 1987] can be understood as compact representations of tableaux, natural deduction, or sequent proof techniques. They avoid the usual redundancies contained in these calculi and are driven by *complementary connections*, i.e. possible leaves in a sequent proof, instead of the logical connectives of a proof goal. Although developed mainly for classical logic and formulas in clause-form, their theoretical foundations could be extended to intuitionistic and various modal logics [Wallen, 1990]. On this basis we have extended the connection method to non-clausal form, intuitionistic logic [Otten and Kreitz, 1995], modal logics [Otten and Kreitz, 1996(b)], and also to fragments of linear logic [Kreitz *et al.*, 1997, Mantel and Kreitz, 1998].

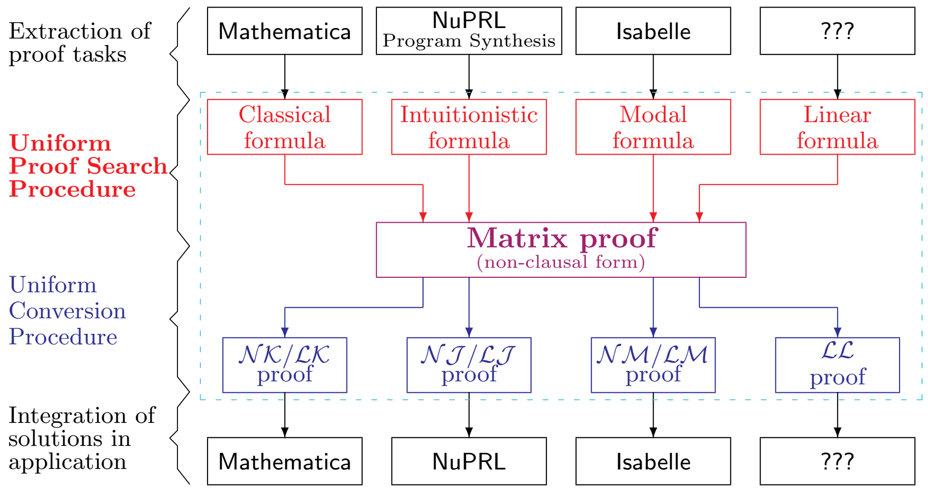

Extraction of proof tasks
Mathematica
NuPRL Program Synthesis
Isabelle
???
Uniform Proof Search Procedure
Classical formula
Intuitionistic formula
Modal formula
Linear formula
Matrix proof (non-clausal form)
Uniform Conversion Procedure
$\mathcal{NK}/\mathcal{LK}$ proof
$\mathcal{NJ}/\mathcal{LJ}$ proof
$\mathcal{NM}/\mathcal{LM}$ proof
$\mathcal{LL}$ proof
Integration of solutions in application
Mathematica
NuPRL
Isabelle
???

Figure 1: Combining automated and interactive proof systems for various logics

Since the actual proofs generated by automated proof search procedures tend to have a very technical look, we have developed a uniform algorithm for converting matrix proofs in these logics into sequent proofs [Schmitt and Kreitz, 1995, Schmitt and Kreitz, 1996, Kreitz and Schmitt, 1999]. This allows us to view matrix proofs as plans for predicate logic proofs to be executed within a proof assistant [Bibel *et al.*, 1996, Kreitz *et al.*, 1996]. Thus our integrated approach, illustrated in Figure 1, does not only combine proof methods for a variety of different logics but also automated and interactive theorem proving.

In this paper we will present a uniform, matrix-based proof search procedure for classical logic, intuitionistic logic, modal logics, and fragments of linear logic. In Section 2 we will present the matrix characterization of logical validity in classical logic in a unified representation. On this basis we develop a non-clausal extension of Bibel's original connection method in Section 3 and adapt it to constructive logic (Section 4) and the modal logics T, D, D4, S4, and S5 (Section 5). In Section 6 we extend both the matrix characterization and our proof search procedure to $\mathcal{MLL}$, the multiplicative fragment of linear logic. We conclude with brief discussion of possible further extensions and the integration of matrix methods into interactive proof systems.

2 The Matrix Characterization of Logical Validity

The connection method [Bibel, 1981, Bibel, 1987] was originally designed as proof search method for formulas in clause normal form. But as normalization of formulas is often costly and as many non-classical logics do not have normal forms, it is necessary to develop connection methods for formulas in non-clausal form. Bibel [Bibel, 1987] already describes a non-clausal version of his connection method. The version that we will present here is more general. It is based on Wallen's matrix characterizations of logical validity [Wallen, 1990] and can easily be adapted to a variety of logics.

Since matrix proofs can be viewed as compact representations of analytic tableaux, many notions from tableau calculi carry over to matrix methods. The main difference is that tableau proofs are based on rules that decompose a formula, generate subformulas, and eventually close off proof branches, while matrix methods operate directly on the formula tree and search for connections, i.e. pairs of identical literals with different polarities that could close a branch in a tableau proof. In this section we will first introduce the basic concepts used in matrix methods and then characterize logical validity in terms of these concepts.

A *formula tree* is the representation of a formula F as a syntax tree. Each node corresponds to exactly one subformula F_s of F and is marked with a unique name $a_0, a_1, \ldots$, its *position*. The *label* of a position u denotes the major connective of F_s or the formula F_s, if it is atomic. In the latter case, u is called an *atomic position* (or *atom*) and can also be identified by its label. The *tree ordering* $<$ of F is the partial ordering on the positions in the formula tree where the *root* is the smallest position with respect to this tree ordering.

Each position in a formula tree is associated with a polarity and a principal type. The *polarity* (0 or 1) of a position is determined by the label and polarity of its parent. The root position has polarity 0. The *principal type* of a position is determined by its polarity and its label. Atomic positions have no principal type. Polarities and types of positions are defined in the table below. For example, a position labelled with $\Rightarrow$ and polarity 1 has principal type β and its successor positions have polarity 0 and 1, respectively. For a given formula we denote the sets of positions of type γ and δ by Γ and Δ.

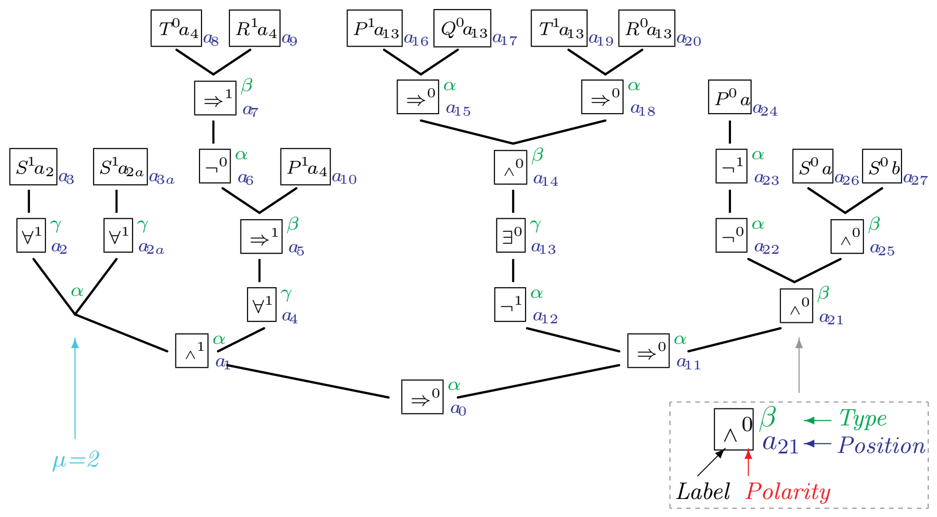
principal type α	$(A \wedge B)^1$	$(A \vee B)^0$	$(A \Rightarrow B)^0$	$(\neg A)^1$	$(\neg A)^0$
successor polarity	$A^1,\ B^1$	$A^0,\ B^0$	$A^1,\ B^0$	A^0	A^1
principal type β	$(A \wedge B)^0$	$(A \vee B)^1$	$(A \Rightarrow B)^1$		
successor polarity	$A^0,\ B^0$	$A^1,\ B^1$	$A^0,\ B^1$		
principal type γ	$(\forall x A)^1$	$(\exists x A)^0$	*principal type* δ	$(\forall x A)^0$	$(\exists x A)^1$
successor polarity	A^1	A^0	successor polarity	A^0	A^1

A *quantifier multiplicity* $\mu_Q{:}\Gamma{\rightarrow}\mathbb{N}$ (briefly μ) encodes the number of distinct instances of γ-subformulas that need to be considered during the proof search. By F^μ we denote an *indexed formula*, i.e. a formula and its multiplicity. We consider multiple instances of subformulas according to the multiplicity of its corresponding position in the formula tree and extend the tree ordering accordingly. For technical reasons we substitute variables in atomic formulas by the corresponding quantifier positions, i.e. γ- and δ-positions. The formula tree of $F_1 \equiv$

$$\forall x Sx \wedge \forall y(\neg(Ty \Rightarrow Ry) \Rightarrow Py) \ \Rightarrow\ \neg\exists z((Pz \Rightarrow Qz) \wedge (Tz \Rightarrow Rz)) \ \Rightarrow\ \neg\neg Pa \wedge Sa \wedge Sb,$$

where each position is marked with its label, polarity, and principal type, and the multiplicity of the subformula $\forall x Sx$ is 2, is shown in Figure 2.

The *matrix(-representation)* of a formula F is a two-dimensional representation of its atomic formulas without connectives and quantifiers, which is better suited for illustration purposes. In a matrix representation α-related positions appear side by side and β-related positions appear on top of each other, where two positions u and v are *α-related* (or *β-related*), denoted $u{\sim_\alpha}v$ ($u{\sim_\beta}v$), iff $u{\neq}v$ and the greatest common ancestor of u and v w.r.t. the tree ordering $<$ is of principal type α (or β). u is *α-related* (*β-related*) to a set S of positions if $u{\sim_\alpha}v$ ($u{\sim_\beta}v$) for all $v \in S$. The matrix representation of F_1^μ is given in Figure 3.



$T^0 a_4$ a_8
$R^1 a_4$ a_9
$P^1 a_{13}$ a_{16}
$Q^0 a_{13}$ a_{17}
$T^1 a_{13}$ a_{19}
$R^0 a_{13}$ a_{20}
$\Rightarrow^1$ β a_7
$\Rightarrow^0$ α a_{15}
$\Rightarrow^0$ α a_{18}
$P^0\, a$ a_{24}
$S^1 a_2$ a_3
$S^1 a_{2a}$ a_{3a}
$\neg^0$ α a_6
$P^1 a_4$ a_{10}
$\wedge^0$ β a_{14}
$\neg^1$ α a_{23}
$S^0\, a$ a_{26}
$S^0\, b$ a_{27}
$\forall^1$ γ a_2
$\forall^1$ γ a_{2a}
$\Rightarrow^1$ β a_5
$\exists^0$ γ a_{13}
$\neg^0$ α a_{22}
$\wedge^0$ β a_{25}
α
$\forall^1$ γ a_4
$\neg^1$ α a_{12}
$\wedge^0$ β a_{21}
$\wedge^1$ α a_1
$\Rightarrow^0$ α a_{11}
$\Rightarrow^0$ α a_0
$\mu{=}2$
$\wedge^0$ β a_{21}
Type
Position
Label
Polarity

Figure 2: Formula tree for F_1^μ with $\mu(a_2)$=2

The matrix characterizations of logical validity [Bibel, 1981, Bibel, 1987] [Wallen, 1990, Kreitz *et al.*, 1997] depend on the concepts of paths, connections, and complementarity. A *path* through a formula F is a maximal set of mutually α-related atomic positions of its formula tree. It can be visualized as a maximal horizontal line through the matrix representation of F. A *connection* is a pair of atomic positions labelled with the same predicate symbol but with different polarities. A connection is *complementary* if its atomic formulas are unifiable by an admissible substitution.

The precise definition of complementarity depends on the logic under consideration. For classical logic, we only need to consider quantifier- or first-order substitutions. A *(first-order) substitution* σ_Q (briefly σ) is a mapping from positions of type γ to terms over $\Delta\cup\Gamma$. It induces a relation $\sqsubset_Q \subseteq \Delta \times \Gamma$ in the following way: if $\sigma_Q(u) = t$, then $v \sqsubset_Q u$ for all $v \in \Delta$ occurring in t. The relation $\sqsubset_Q$ expresses the *eigenvariable condition* of the sequent calculus, where eigenvariables $v \in \Delta$ have to be introduced before they are assigned to variables $u \in \Gamma$. Together with the tree ordering $<$ the relation $\sqsubset_Q$ determines a reduction ordering $\lhd$. This reduction ordering has to be acyclic, i.e. it has to respect the eigenvariable condition.

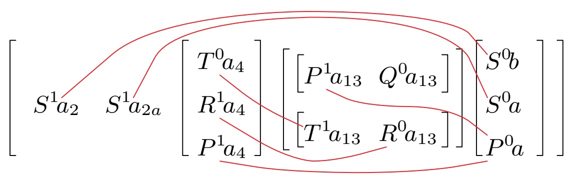


Figure 3: Matrix of the formula F_1

Definition 1 (Complementarity in Classical Logic).

- A first-oder substitution σ_Q is *admissible* with respect to F^μ iff the induced *reduction ordering* $\lhd := (< \cup \sqsubset_Q)^+$, i.e. the transitive closure of $<$ and $\sqsubset_Q$,is irreflexive.
- A connection $\{u, v\}$ is σ_Q-*complementary* iff $\sigma_Q(\mathit{label}(u)) {=} \sigma_Q(\mathit{label}(v))$.

As paths through matrices correspond to fully expanded branches of tableau proofs and complementary connections to closed branches, a formula F is valid if every path through some F^μ contains a complementary connection.

Theorem 2 (Matrix Characterization for Classical Logic). *A formula F is (classically) valid iff there is a multiplicity μ, an admissible substitution σ and a set of σ-complementary connections such that every path through F^μ contains a connection from this set.*

A proof of this theorem can be found in [Bibel, 1987].

Example 1. In the matrix[1] for F_1^μ in Figure 3 there are 18 paths (note that $P^1 a_{13}$ and $R^0 a_{13}$ or $T^1 a_{13}$ and $Q^0 a_{13}$ are *not* α-related). Each path contains one of the connections $\{S^1 a_2, S^0 b\}$, $\{S^1 a_{2a}, S^0 a\}$, $\{T^0 a_4, T^1 a_{13}\}$, $\{R^1 a_4, R^0 a_{13}\}$, $\{P^1 a_4, P^0 a\}$, and $\{P^1 a_{13}, P^0 a\}$. The six connections are complementary under the first-order substitution $\sigma_Q = \{a_2 \backslash b, a_{2a} \backslash a, a_4 \backslash a, a_{13} \backslash a\}$. As no δ-positions occur in σ_Q, the induced reduction ordering $\lhd$ is the tree ordering $<$ and irreflexive. Thus σ_Q is admissible and F_1 is valid.

3 A Uniform Proof Search Procedure

According to the above matrix characterization the validity of a formula F can be proven by showing that all paths through the matrix representation of F^μ are complementary, i.e. contain a complementary connection. Obviously it is not very efficient to check all possible paths for complementarity. Instead, a path checking algorithm should be driven by the connections: once a complementary connection has been identified all paths containing this connection can be eliminated from further consideration. This idea is similar to Bibel's connection method for classical logic [Bibel, 1987], but our algorithm is more general and because of that useful for proof search in various non-classical logics.

The key notions of our path checking algorithm are active paths, active subgoals, and open goals. During proof search the *active path* $\mathcal{P}$ specifies those paths that are currently investigated for complementarity. All paths that contain $\mathcal{P}$ and an element u of the *active subgoal* $\mathcal{C}$ have already been proven to contain a complementary connection. All paths that contain $\mathcal{P}$ and an element v of the *open goal* $\mathcal{E}$ must still be tested for complementarity. If the latter can be proven complementary as well then all paths containing the active path are complementary. The algorithm will recursively check whether all paths containing the empty active path are complementary.

Let $\mathcal{A}$ denote the set of all atomic positions in the formula F. A *subpath* $\mathcal{P} \subseteq \mathcal{A}$ is a (not necessarily maximal) set of mutually α-related atomic positions. A subpath $\mathcal{P} \subseteq \mathcal{A}$ is a path iff there is no $u \in \mathcal{A}$ with $u {\sim_\alpha} \mathcal{P}$. A *subgoal* $\mathcal{C} \subseteq \mathcal{A}$ is a set of mutually β-related atomic positions. During proof search certain tuples

[1] We denote atomic positions by their labels, i.e. by atomic formulas.

$(\mathcal{P},\mathcal{C})$ consisting of a non-complementary subpath $\mathcal{P}$ and a subgoal $\mathcal{C}$ with $u{\sim_\alpha}\mathcal{P}$ for all $u \in \mathcal{C}$ will be called *active goals*. $\mathcal{P}$ will be the *active path* and $\mathcal{C}$ the *active subgoal*. The *open goal* $\mathcal{E} \subseteq \mathcal{A}$ with respect to an active goal $(\mathcal{P},\mathcal{C})$ is the set of atomic positions v with $v{\sim_\alpha}\mathcal{P}$ and $v{\sim_\beta}\mathcal{C}$.

Example 2. In Figure 3 the sets $\mathcal{P}_1{=}\{S^1a_2, S^1a_{2a}, T^1a_{13}, P^0a\}$, $\mathcal{P}_2{=}\{S^1a_2, S^1a_{2a}, R^1a_4, S^0b\}$, and $\mathcal{P}_3{=}\{T^0a_4, T^1a_{13}, R^0a_{13}\}$ are subpaths for the formula F_1. $\mathcal{C}_1{=}\{T^0a_4\}$, $\mathcal{C}_2{=}\{Q^0a_{13}, T^1a_{13}\}$, and $\mathcal{C}_3{=}\{S^1a_2\}$ are subgoals. $(\mathcal{P}_1,\mathcal{C}_1)$, $(\mathcal{P}_2,\mathcal{C}_2)$, and $(\mathcal{P}_3,\mathcal{C}_3)$ are active goals. The set $\{R^1a_4, P^1a_4\}$ is the open goal w.r.t. the active goal $(\mathcal{P}_1,\mathcal{C}_1)$; the empty set $\emptyset$ is the open goal w.r.t. $(\mathcal{P}_2,\mathcal{C}_2)$ or $(\mathcal{P}_3,\mathcal{C}_3)$.

We call an active goal $(\mathcal{P},\mathcal{C})$ *provable* with respect to a formula F iff for the open goal $\mathcal{E}$ w.r.t. $(\mathcal{P},\mathcal{C})$ and for all $v \in \mathcal{E}$ all paths $\mathcal{P}'$ through F with $P{\cup}\{v\} \subseteq \mathcal{P}'$ are complementary. This definition leads to a more algorithmic characterization of validity.

Theorem 3.
A formula F is valid iff there is a multiplicity μ and an admissible substitution σ such that the active goal $(\emptyset,\emptyset)$ w.r.t. F^μ is provable.

Proof. Using the characterization of validity in theorem 2 it suffices to show that the active goal $(\emptyset,\emptyset)$ w.r.t. F^μ is provable iff every path through F^μ is σ-complementary.

Let $(\emptyset,\emptyset)$ be provable. Then the open goal $\mathcal{E} := \{v \in \mathcal{A} | v{\sim_\alpha}\emptyset \wedge v{\sim_\beta}\emptyset\}$ w.r.t. $(\emptyset,\emptyset)$ is the set of all atomic positions. Thus all paths through F^μ contain an element of $\mathcal{E}$ and are complementary according to the definition of provability. Conversely, if every path through F^μ is σ-complementary then every path that contains some $v \in \mathcal{E} := \{v \in \mathcal{A} | v{\sim_\alpha}\emptyset \wedge v{\sim_\beta}\emptyset\}$ is complementary. Since every paths through F^ν contains a $v \in \mathcal{E}$ the active goal $(\emptyset,\emptyset)$ is provable. □

Since the proof of theorem 3 depends only on a matrix characterization of logical validity, the algorithmic characterization applies to all logics for which a matrix characterization in the style of theorem 2 can be given. It leads to a uniform path checking algorithm which, coupled with an appropriate definition of complementarity, can be used as proof search procedure for a variety of different logics. The path checking method is described by the following theorem, which gives sufficient and necessary conditions for provability.

Theorem 4.
Let $(\mathcal{P},\mathcal{C})$ an active goal and $\mathcal{E} := \{v \in \mathcal{A} | v{\sim_\alpha}\mathcal{P} \wedge v{\sim_\beta}\mathcal{C}\}$ the open subgoal w.r.t. $(\mathcal{P},\mathcal{C})$. The active goal $(\mathcal{P},\mathcal{C})$ is provable iff

1. *the open goal $\mathcal{E}$ is empty, or*
2. *there is a complementary connection $\{A, \bar{A}\}$ with $A \in \mathcal{E}$, such that the active goal $(\mathcal{P}, \mathcal{C} \cup \{A\})$ is provable and*
   $\bar{A} \in \mathcal{P}$ or $\bar{A}{\sim_\alpha}(\mathcal{P} \cup \{A\})$ and the active goal $(\mathcal{P} \cup \{A\}, \{\bar{A}\})$ is provable.

Proof. Let $(\mathcal{P},\mathcal{C})$ a provable active goal and $\mathcal{E} := \{v \in \mathcal{A} | v{\sim_\alpha}\mathcal{P} \wedge v{\sim_\beta}\mathcal{C}\} {\neq} \emptyset$. Then there is a complementary connection $\{A, \bar{A}\}$ with $A \in \mathcal{E}$. $(\mathcal{P}, \mathcal{C} \cup \{A\})$ is provable because $(\mathcal{P},\mathcal{C})$ is. If $\bar{A} \notin \mathcal{P}$ then let $u \in \mathcal{E}' := \{u \in \mathcal{A} | u{\sim_\alpha}\mathcal{P} \cup \{A\} \wedge u{\sim_\beta}\{\bar{A}\}\}$ and $\mathcal{P}'$ be a path with $P{\cup}\{A\}{\cup}\{u\} \subseteq \mathcal{P}'$. Since $(\mathcal{P},\mathcal{C})$ is provable and $A \in \mathcal{E}$, $\mathcal{P}'$ must be complementary. Hence $(\mathcal{P} \cup \{A\}, \{\bar{A}\})$ is provable.

Conversely let $(\mathcal{P},\mathcal{C})$ an active goal. If $\mathcal{E} = \emptyset$ then all conditions for the provability of $(\mathcal{P},\mathcal{C})$ are trivially satisfied. In the other case let $\{A, \bar{A}\}$ be a complementary connection with $A \in \mathcal{E}$ such that $(\mathcal{P}, \mathcal{C}{\cup}\{A\})$ is provable, $v \in \mathcal{E}$, and $\mathcal{P}'$ be a path with $P{\cup}\{u\} \subseteq \mathcal{P}'$.

– If $A \notin \mathcal{P}'$ then there must be some $u \in \mathcal{P}'$ with $u{\sim_\beta}A$, $u{\sim_\beta}\mathcal{C}$, and $u{\sim_\alpha}\mathcal{P}$. Since $(\mathcal{P}, \mathcal{C} \cup \{A\})$ is provable, $\mathcal{P}'$ must be complementary.
– If $A \in \mathcal{P}'$ and $\bar{A} \in \mathcal{P}$ then $\mathcal{P}'$ is obviously complementary.
– If $A \in \mathcal{P}'$ and $\bar{A} \notin \mathcal{P}$ then there must be some $u \in \mathcal{P}'$ with $u{\sim_\beta}\bar{A}$. For this u we have $u{\sim_\alpha}A$ and $u{\sim_\alpha}\mathcal{P}$. Thus u is an element of the open goal for $(\mathcal{P} \cup \{A\}, \{\bar{A}\})$. Since by assumption $(\mathcal{P} \cup \{A\}, \{\bar{A}\})$ is provable, $\mathcal{P}'$ must be complementary.

Thus all the requirements for the provability of $(\mathcal{P}, \mathcal{C})$ are satisfied. □

Figure 4 presents a mathematical description of a uniform algorithm for proving the validity of a given formula F that is based on the above theorems and can easily be converted into a functional or logic program. According to Theorem 3 we have to show that the active goal $(\emptyset, \emptyset)$ is provable w.r.t. some F^μ and some admissible substitution σ. To check the provability of an active goal $(\mathcal{P}, \mathcal{C})$ the function `provable` recursively applies Theorem 4. If the open goal $\mathcal{E} = \{v \in \mathcal{A} \mid v{\sim_\alpha}\mathcal{P} \wedge v{\sim_\beta}\mathcal{C}\}$ is not empty, it must investigate the extensions of $\mathcal{E}$ by complementary connections $\{A, \bar{A}\}$ that originate in $\mathcal{E}$.

For this purpose, the function `check-extensions` recursively considers all positions $A \in \mathcal{E}$ and the sets $\mathcal{D}$ of all literals $\bar{A}$ that are connected to A. According to Theorem 4.2 the latter can be restricted to the positions $\bar{A}$ that are already on the active path $\mathcal{P}$ or α-related to $(\mathcal{P} \cup \{A\})$. As an optimization, the choice of alternative elements of $\mathcal{E}$ is limited to $\{v \in \mathcal{E} \mid v{\sim_\alpha}A\}$ if the complementarity of all paths through A could not be shown by investigating $\{A, \bar{A}\}$. In this case the complementarity must depend on a different connection on the path, i.e. on some $v \in \mathcal{E}$ that is α-related to A. If there is no more choice, the computation fails.

```
prove($F$,$n$)
  = { provable($\emptyset,\emptyset,\sigma$)                                  if this computation succeeds
        where $\sigma,\mu$ = initialize($F$,$n$)
        and $\mathcal{CON}$ = connections($F^\mu$)
      prove($F$,$n$+1)                                        otherwise

provable($\mathcal{P},\mathcal{C},\sigma$)
  = { check-extension($\mathcal{E},\sigma$)                                 if $\mathcal{E}\neq\emptyset$
        where $\mathcal{E} = \{v \in \mathcal{A} \mid v{\sim_\alpha}\mathcal{P} \wedge v{\sim_\beta}\mathcal{C}\}$
      $\sigma$                                                        otherwise

check-extension($\mathcal{E},\sigma$)
  = { check-connections($\mathcal{D},A,\sigma$)                             if this computation succeeds
        where $\mathcal{D} = \{\bar{A} \in \mathcal{A} \mid \{A,\bar{A}\} \in \mathcal{CON} \wedge (\bar{A} \in \mathcal{P} \vee \bar{A}{\sim_\alpha}(\mathcal{P}\cup\{A\}))\}$
        where $A \in \mathcal{E}$ arbitrary
      check-extension($\{v \in \mathcal{E} | v{\sim_\alpha}A\},\sigma$)             otherwise (and $\mathcal{E}\neq\emptyset$)

check-connections($\mathcal{D},A,\sigma$)
  = { provable($\mathcal{P},\mathcal{C}\cup\{A\},\sigma_2$)                          if this computation succeeds
        where $\sigma_2$ = { $\sigma_1$                                       if $\bar{A} \in \mathcal{P}$
                     provable($\mathcal{P}\cup\{A\},\{\bar{A}\},\sigma_1$)      otherwise
        where $\bar{A} \in \mathcal{D}$ arbitrary
        and $\sigma_1$ = unify-check($A,\bar{A},F^\mu,\sigma$)
      check-connections($\mathcal{D}-\{\bar{A}\},A,\sigma$)               otherwise (and $\mathcal{D}\neq\emptyset$)
```

Figure 4: Uniform path checking algorithm in mathematical notation

The function check-connections recursively checks whether the remaining conditions of Theorem 4.2 hold for some $\bar{A} \in \mathcal{D}$. It tests $\{A, \bar{A}\}$ for complementarity and whether $\bar{A} \in \mathcal{P}$ is true or the active goal $(\mathcal{P} \cup \{A\}, \{\bar{A}\})$ is provable. It succeeds if this is the case and the active goal $(\mathcal{P}, \mathcal{C} \cup \{A\})$ is provable. It fails otherwise or if the choices for $\bar{A} \in \mathcal{D}$ are exhausted. The chosen order of provability tests leads to a depth-first search where active paths are extended as long as possible before another path is explored.

As usual, the substitution σ is constructed *during* the extension process. It is the result of the main function prove and threaded through all the subfunctions. To determine the multiplicity, the function prove is initialized with $\mu \equiv 1$ and globally increases μ until the active goal $(\emptyset, \emptyset)$ w.r.t F^μ is shown to be provable.

The algorithm is parameterized with two functions, which express the specific properties of the logic under consideration. The function initialize determines the initial value for the substitution σ and the multiplicity μ while unify-check$(A, \bar{A}, F^\mu, \sigma)$ tries to compute a substitution that unifies the labels of A and $\bar{A}$, extends σ, and leads to an acyclic reduction ordering in F^μ. Both functions are provided separately in tables for each logic while the general path checking algorithm remains unchanged for a variety of logics.

initialize(F,n)	$(\emptyset,\ n)$
unify-check$(A, \bar{A}, F^\mu, \sigma)$	term_unify$(\sigma(label(A)), \sigma(label(\bar{A})))$ if $\lhd := (< \cup \sqsubset_Q)^+$ is irreflexive

Table 1: initialize(F,n) and unify-check$(A, \bar{A}, F^\mu, \sigma)$ for classical logic

For classical logic (table 1) initialize(F,n) computes a pair (σ, μ) with $\sigma{=}\emptyset$ and $\mu(u){=}n$ for all $u \in \Gamma$. unify-check$(A, \bar{A}, F^\mu, \sigma)$ computes a most general term-unifier σ_Q of $\sigma(label(A))$ and $\sigma(label(\bar{A}))$, as well as the induced reduction ordering $\lhd := (< \cup \sqsubset_Q)^+$. It returns σ_Q if $\lhd$ is irreflexive and fails otherwise or if the two atoms cannot be unified. To compute the first-order substitution σ_Q, we can use well-known term unification algorithms [Robinson, 1965, Martelli and Montanari, 1982], while the irreflexivity of the induced reduction ordering can be checked by standard algorithms for testing the acyclicity of directed graphs or by the well-known skolemization technique.

Theorem 5 (Correctness and Completeness for Classical Logic).
The function prove$(F$,1$)$ *together with initialization and unification from table 1 succeeds iff F is valid in classical logic. In this case it returns an admissible substitution σ_Q which makes every path through some F^μ complementary.*

Proof. By simultaneous induction we prove the following facts

1. check-connections$(\mathcal{D}, A, \sigma)$ succeeds if there is some $\bar{A} \in \mathcal{D}$ and some admissible substitution σ' that extends σ such that $\{A, \bar{A}\}$ is σ'-complementary, $(\mathcal{P}, \mathcal{C} \cup \{A\})$ is provable under σ', and $\bar{A} \in \mathcal{P}$ or $\bar{A} {\sim_\alpha} (\mathcal{P} \cup \{A\})$ and $(\mathcal{P} \cup \{A\}, \{\bar{A}\})$ is provable under σ'.
2. check-extension$(\mathcal{E}, \sigma)$ succeeds if there is some admissible substitution σ' that extends σ and some σ'-complementary connection $\{A, \bar{A}\}$ with $A \in \mathcal{E}$ such that $(\mathcal{P}, \mathcal{C} \cup \{A\})$ is provable under σ', and $\bar{A} \in \mathcal{P}$ or $\bar{A} {\sim_\alpha} (\mathcal{P} \cup \{A\})$ and $(\mathcal{P} \cup \{A\}, \{\bar{A}\})$ is provable under σ'.

3. `provable`$(\mathcal{P},\mathcal{C},\sigma)$ succeeds if there is some admissible substitution σ' that extends σ such that $(\mathcal{P},\mathcal{C})$ is provable under σ'.

In each of the respective cases the result is the substitution σ'. As a consequence, the function call `prove(`F`,`n`)` succeeds if there is a multiplicity μ with $\mu(u){\geq}n$ for all $u \in \Gamma$ and an admissible substitution σ such that $(\emptyset,\emptyset)$ w.r.t. F^μ is provable under σ'. In this case the result is σ and F is valid according to Theorem 3 and 4. $\square$

Our general path checking algorithm can be viewed as reference version for the implementation of theorem provers for a variety of different logics that is far more flexible than especially tailored proof search procedures. Obviously it is possible to improve the efficiency of the algorithm presented in Figure 4, since the stepwise increase of the multiplicity is very coarse and because it cannot decide that a given propositional formula is *invalid*. An efficient implementation of our algorithm will determine the multiplicity for each suitable position dynamically *during* the path checking process. Other techniques that were used in theorem provers based on the usual connection method can be integrated as well. For example a technique similar to the liberalized δ-rule applied in tableau calculi like ileanTAP [Otten, 1997] can be integrated.

4 Proving Theorems in Constructive Logic

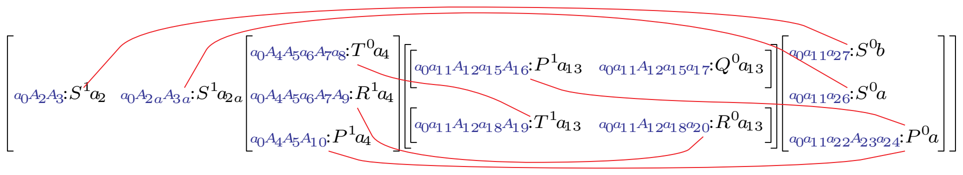
As program synthesis and verification often relies on constructive arguments, systems for automated program development must be supported by proof search procedures for intuitionistic logic. Independently from the philosophical differences between classical and intuitionistic logic the main distinction between these two logics can be expressed by a different treatment of $\forall$, $\Rightarrow$, and $\neg$. Whereas in the classical sequent calculus only the quantifier rules affected by the eigenvariable condition are not permutable, in the intuitionistic sequent calculus in addition the rules dealing with $\forall$, $\Rightarrow$, and $\neg$ in the succedent are not permutable.

A matrix method for intuitionistic logic must therefore not only check if two connected atomic formulas can be unified by a first-order substitution but also if they can both be reached by applying an appropriate sequence of sequent rules. Only then they form a leaf in a sequent proof. In the matrix characterization this is reflected by an additional *intuitionistic substitution* σ_J, which has to make the *prefixes* of the connected atomic positions identical, where a prefix of a position u is a string consisting of variables and constants that essentially describes the location of u in the formula tree.

For this purpose the positions labelled with atoms, $\forall$, $\Rightarrow$, or $\neg$ receive an additional *intuitionistic type* according to the following table.

intuitionistic type ϕ	$(\neg A)^1$	$(A \Rightarrow B)^1$	$(\forall x A)^1$	P^1 (P atomic)
successor polarity	A^0	A^0, B^1	A^1	—
intuitionistic type ψ	$(\neg A)^0$	$(A \Rightarrow B)^0$	$(\forall x A)^0$	P^0 (P atomic)
successor polarity	A^1	A^1, B^0	A^0	—

Positions of type ψ correspond to the application of non-invertible sequent rules and are viewed as constants in a prefix string while ϕ-positions are variables. This makes it possible to use unification to determine the ψ-positions that must



$a_0A_2A_3$:S^1a_2
$a_0A_{2a}A_{3a}$:S^1a_{2a}
$a_0A_4A_5a_6A_7a_8$:T^0a_4
$a_0A_4A_5a_6A_7A_9$:R^1a_4
$a_0A_4A_5A_{10}$:P^1a_4
$a_0a_{11}A_{12}a_{15}A_{16}$:$P^1a_{13}$
$a_0a_{11}A_{12}a_{15}a_{17}$:$Q^0a_{13}$
$a_0a_{11}A_{12}a_{18}A_{19}$:$T^1a_{13}$
$a_0a_{11}A_{12}a_{18}a_{20}$:$R^0a_{13}$
$a_0a_{11}a_{27}$:S^0b
$a_0a_{11}a_{26}$:S^0a
$a_0a_{11}a_{22}A_{23}a_{24}$:$P^0a$

Figure 5: Matrix of the formula F_1 with prefixes

be reduced before a ϕ position in a valid sequent proof[2] and to develop a matrix characterization for intuitionistic validity whose formulation is almost identical to the one for classical logic.

The *prefix* $pre_J(u)$ of an atomic position u is a string $u_1u_2\ldots u_n$ where $u_1{<}u_2{<}\ldots{<}u_n{=}u$ are the elements of $\Psi \cup \Phi$ (the positions of type ψ or type ϕ) that dominate u in the formula tree. An *intuitionistic substitution* σ_J is a mapping from positions of type ϕ to (possibly empty) strings over $\Psi \cup \Phi$. It induces a relation $\sqsubset_J \subseteq \Psi \times \Phi$ in the following way: if $\sigma_J(u){=}p$, then $v{\sqsubset_J}u$ for all characters $v \in \Psi$ occurring in p.

Definition 6 (Complementarity in Intuitionistic Logic).
Let $\sigma{:=}(\sigma_Q, \sigma_J)$ be a *combined substitution* consisting of a first-order substitution σ_Q and an intuitionistic substitution σ_J.

1. σ is *J-admissible* iff the induced *reduction ordering* $\lhd{:=}({<}{\cup}{\sqsubset_Q}{\cup}{\sqsubset_J})^+$ is irreflexive and $|\sigma_J(\mathrm{pre}_J(v))|{\leq}|\sigma_J(\mathrm{pre}_J(u))|$ holds for all $u \in \Gamma$ and all $v \in \Delta$ occuring in $\sigma_Q(u)$.
2. A connection $\{u, v\}$ is σ*-complementary* iff $\sigma_Q(label(u)){=}\sigma_Q(label(v))$ and $\sigma_J(\mathrm{pre}_J(u)){=}\sigma_J(\mathrm{pre}_J(v))$.

In the intuitionistic sequent calculus formulas of type ϕ can be copied. An *intuitionistic multiplicity* $\mu_J{:}\Phi{\to}\mathbb{N}$ encodes the number of distinct instances of ϕ-subformulas that need to be considered during the proof search. It can be combined with a quantifier multiplicity μ_Q and leads to an indexed formula F^μ.

Theorem 7 (Matrix Characterization for Intuitionistic Logic).
A formula F is intuitionistically valid iff there is a multiplicity $\mu := (\mu_Q, \mu_J)$, a J-admissible combined substitution $\sigma = (\sigma_Q, \sigma_J)$, and a set of σ-complementary connections such that every path through F^μ contains a connection from this set.

A proof of this theorem can be found in [Wallen, 1990].

Example 3. Consider F_1^μ from Figure 2 with $\mu_Q(a_2){=}2$, $\mu_Q(a_4){=}1$, $\mu_Q(a_{13}){=}1$, and $\mu_J \equiv 1$. The ψ-positions are $\{a_0, a_6, a_8, a_{11}, a_{15}, a_{17}, a_{18}, a_{20}, a_{22}, a_{24}, a_{26}, a_{27}\}$, while $\{a_2, a_{2a}, a_3, a_{3a}, a_4, a_5, a_7, a_9, a_{10}, a_{12}, a_{16}, a_{19}, a_{23}\}$ is the set of ϕ-positions, which in the following will be indicated by capital letters. The prefixes of all atomic positions are shown in Figure 5. The prefixes of the six connections used to show the classical validity of F_1 in Example 1 can be unified by the intuitionistic substitution $\sigma_J = \{A_2\backslash\varepsilon,\ A_{2a}\backslash\varepsilon,\ A_3\backslash a_{11}a_{27},\ A_{3a}\backslash a_{11}a_{26},\ A_4\backslash\varepsilon,\ A_5\backslash a_{11}a_{22},\ A_7\backslash a_{18}a_{20},\ A_9\backslash\varepsilon,$

[2] This methodology is inspired by the admissibility condition for first-order substitutions in Definition 1, where $\sqsubset_Q$ determines which δ-positions must be reduced before certain γ-positions in order to satisfy the eigenvariable-condition.

$A_{10}\backslash a_6a_{15}a_{24}$, $A_{12}\backslash a_{22}a_6$, $A_{16}\backslash a_{24}$, $A_{19}\backslash a_{20}a_8$, $A_{23}\backslash a_6a_{15}\}$, where ε is the empty string. The terms are unified by the first-order substitution $\sigma_Q = \{a_2\backslash b,\ a_{2a}\backslash a,\ a_4\backslash a,\ a_{13}\backslash a\}$.

The combined substitution $\sigma = (\sigma_Q, \sigma_J)$ is J-admissible, as the induced reduction ordering $\lhd{=}({<}\cup\sqsubset_Q\cup\sqsubset_J)^+$ is irreflexive. Thus F_1 is intuitionistically valid.

The algorithmic characterizations of logical validity in theorems 3 and 4 hold accordingly with the intuitionistic definitions of complementarity and multiplicity. Therefore our path checking algorithm presented in Figure 4 can be used for intuitionistic logic as well. We only have to provide the logic-specific functions `initialize` and `unify-check`.

`initialize`(F,n)	$((\emptyset,\emptyset),\ n)$
`unify-check`$(A, \bar{A}, F^\mu, (\sigma_Q, \sigma_J))$	(σ_Q', σ_J') where $\sigma_Q' = \texttt{term\_unify}(\sigma_Q(\mathit{label}(A)), \sigma_Q(\mathit{label}(\bar{A})))$ $\sigma_J' = \texttt{prefix\_unify}_J(\sigma_J(\texttt{pre}_J(A)), \sigma_J(\texttt{pre}_J(\bar{A})))$ if $\lhd := ({<} \cup \sqsubset_Q' \cup \sqsubset_J')^+$ is irreflexive and $\lvert\sigma_J'(\mathrm{pre}_J(v))\rvert \leq \lvert\sigma_J'(\mathrm{pre}_J(u))\rvert$ for all $u \in \Gamma$ and all $v \in \Delta$ occurring in $\sigma_Q'(u)$

Table 2: `initialize`(F,n) and `unify-check`$(A, \bar{A}, F^\mu, \sigma)$ for intuitionistic logic

For intuitionistic logic (table 2) `initialize`(F,n) computes a pair (σ, μ) where $\sigma{=}(\emptyset, \emptyset)$ is a combined substitution and $\mu(u){=}n$ for all $u \in \Gamma\cup\Phi$. The function `unify-check`$(A, \bar{A}, F^\mu, \sigma)$ with $\sigma{=}(\sigma_Q, \sigma_J)$ computes a most general term unifier σ_Q' of $\sigma_Q(\mathit{label}(A))$ and $\sigma_Q(\mathit{label}(\bar{A}))$ as well as a most general prefix unifier σ_J' of $\sigma_J(\mathrm{pre}_J(A))$ and $\sigma_J(\mathrm{pre}_J(\bar{A}))$. It returns (σ_Q', σ_J') if this combined substitution is *J-admissible* and fails otherwise or if either of the two unifications fails.[3]

Prefix Unification.

Computing the most general unifiers of a set of prefix equations $\mathcal{EQ}$ is by no means trivial. General *string unification* [Matiyasevič, 1968, Manakin, 1977, Abdulrab and Pecuchet, 1990], although decidable, is already complicated, because two arbitrary strings may have infinitely many most general unifiers. Fortunately, prefixes are a very restricted class of strings. They do not contain duplicates and the same character cannot occur in two prefixes p and q of atoms in the same formula, unless it belongs to a common substring at the beginning of p and q. These restrictions enabled us to develop a short *prefix unification* algorithm [Otten and Kreitz, 1996(a)] for computing a *minimal* set of most general unifiers. Both [Ohlbach, 1988] and [Schmidt, 1998] do not compute a minimal set of unifiers.

Similar to the ideas of Martelli and Montanari [Martelli and Montanari, 1982] for term unification, our prefix unification algorithm is based on a series of transformation rules that are repeatedly applied to the tuple $\mathcal{EQ}, \sigma_J$. We start the unification with the set of prefix equations $\mathcal{EQ} = \{p_1{=}q_1, ..., p_n{=}q_n\}$ and an

[3] Since there can be more than one most general prefix unifier, backtracking is needed in case the prefix unification fails later on.

empty substitution $\sigma_J{=}\emptyset$. Each transformation step replaces the tuple $\mathcal{EQ}, \sigma_J$ by a modified tuple $\sigma'(\mathcal{EQ}'), \sigma'(\sigma_J)$, where one equation $\{p_i{=}q_i\}$ in $\mathcal{EQ}$ is replaced by $\{p'_i{=}q'_i\}$ and modified by the extended substitution σ'. The algorithm is described by a set of transformation rules " $\{p_i{=}q_i\}, \sigma \rightarrow \{p'_i{=}q'_i\}, \sigma'$ " which can be applied nondeterministically to the equation $\{p_i{=}q_i\} \in \mathcal{EQ}$. The procedure stops if $\mathcal{EQ}$ is empty and returns the resulting substitution σ_J as most general unifier. As the transformation rules are applied nondeterministically, the set of most general unifiers consists of the results of all successfully finished transformations.

R1.	$\{\varepsilon = \varepsilon \vert \varepsilon\},\ \sigma$	$\rightarrow$	$\{\},\ \sigma$	
R2.	$\{\varepsilon = \varepsilon \vert t^+\},\ \sigma$	$\rightarrow$	$\{t^+ = \varepsilon \vert \varepsilon\},\ \sigma$	
R3.	$\{Xs = \varepsilon \vert Xt\},\ \sigma$	$\rightarrow$	$\{s = \varepsilon \vert t\},\ \sigma$	
R4.	$\{Cs = \varepsilon \vert Vt\},\ \sigma$	$\rightarrow$	$\{Vt = \varepsilon \vert Cs\},\ \sigma$	
R5.	$\{Vs = z \vert \varepsilon\},\ \sigma$	$\rightarrow$	$\{s = \varepsilon \vert \varepsilon\},\ \{V \backslash z\} \cup \sigma$	
R6.	$\{Vs = \varepsilon \vert C_1 t\},\ \sigma$	$\rightarrow$	$\{s = \varepsilon \vert C_1 t\},\ \{V \backslash \varepsilon\} \cup \sigma$	
R7.	$\{Vs = z \vert C_1 C_2 t\},\ \sigma$	$\rightarrow$	$\{s = \varepsilon \vert C_2 t\},\ \{V \backslash z C_1\} \cup \sigma$	
R8.	$\{Vs^+ = \varepsilon \vert V_1 t\},\ \sigma$	$\rightarrow$	$\{V_1 t = V \vert s^+\},\ \sigma$	
R9.	$\{Vs^+ = z^+ \vert V_1 t\},\ \sigma$	$\rightarrow$	$\{V_1 t = V' \vert s^+\},\ \{V \backslash z^+ V'\} \cup \sigma$	
R10.	$\{Vs = z \vert Xt\},\ \sigma$	$\rightarrow$	$\{Vs = zX \vert t\},\ \sigma$	($V{\neq}X$, and $s{=}\varepsilon$, $t{\neq}\varepsilon$, or X constant)

s, t, z, s^+, t^+, z^+ denote strings where s^+, t^+, z^+ are non-empty. X, V, V_1, C, C_1 and C_2 denote single characters where V, V_1 are variables and C, C_1, C_2 are constants. V' is a new variable.

Table 3: Transformation rules for prefix unification in intuitionistic logic

Like our path checking algorithm, the basic algorithm `prefix_unify` is uniform for all logics and uses the set of transformation rules as parameters that characterize the specific logic under consideration. Intuitionistic prefix unification (`prefix_unify`$_J$) requires the set of 10 transformation rules presented in table 3. The number of most general unifiers is finite but may grow exponentially with the length of the prefixes to be unified.

Theorem 8.
Let $\mathcal{EQ} = \{p_1{=}q_1, ..., p_n{=}q_n\}$ be a set of prefix equations. Then the prefix unification algorithm together with the transformation rules in table 3 terminates and computes a complete and minimal set of unifiers for $\mathcal{EQ}$.

Example 4. Consider F_1 from Figure 2 with the prefixes shown in Table 5. To make the connection $\{a_{16}, a_{24}\}$, i.e. $\{P^1 a_{13}, P^0 a\}$, complementary we have to unify its prefixes $\text{pre}_J(a_{16}){=}a_0 a_{11} A_{12} a_{15} A_{16}$ and $\text{pre}_J(a_{24}){=}a_0 a_{11} a_{22} A_{23} a_{24}$, i.e. we have to consider the equation $\mathcal{EQ}{=}\{a_0 a_{11} A_{12} a_{15} A_{16}{=}a_0 a_{11} a_{22} A_{23} a_{24}\}$. To solve this equation we start the unification process with the tuple $\{a_0 a_{11} A_{12} a_{15} A_{16}{=}\varepsilon | a_0 a_{11} a_{22} A_{23} a_{24}\}, \{\}$ and apply the transformation rules according to Table 3:

$$\{a_0 a_{11} A_{12} a_{15} A_{16}{=}\varepsilon | a_0 a_{11} a_{22} A_{23} a_{24}\}, \{\}$$
$$\xrightarrow{\text{R3}} \{a_{11} A_{12} a_{15} A_{16}{=}\varepsilon | a_{11} a_{22} A_{23} a_{24}\}, \{\}$$
$$\xrightarrow{\text{R3}} \{A_{12} a_{15} A_{16}{=}\varepsilon | a_{22} A_{23} a_{24}\}, \{\}$$

1. $\xrightarrow{\text{R6}} \{a_{15} A_{16}{=}\varepsilon | a_{22} A_{23} a_{24}\}, \{A_{12} \backslash \varepsilon\} \quad \diamondsuit$
2. $\xrightarrow{\text{R10}} \{A_{12} a_{15} A_{16}{=}a_{22} | A_{23} a_{24}\}, \{\}$

2.1 $\xrightarrow{\text{R9}} \{A_{23}a_{24}{=}A'|a_{15}A_{16}\}, \{A_{12}\backslash a_{22}A'\} \xrightarrow{\text{R10}} \{A_{23}a_{24}{=}A'a_{15}|A_{16}\}, \{A_{12}\backslash a_{22}A'\}$
$\xrightarrow{\text{R9}} \{A_{16}{=}A''|a_{24}\}, \{A_{12}\backslash a_{22}A', A_{23}\backslash A'a_{15}A''\}$
$\xrightarrow{\text{R10}} \{A_{16}{=}A''a_{24}|\varepsilon\}, \{A_{12}\backslash a_{22}A', A_{23}\backslash A'a_{15}A''\}$
$\xrightarrow{\text{R5}} \{\varepsilon{=}\varepsilon|\varepsilon\}, \{A_{12}\backslash a_{22}A', A_{23}\backslash A'a_{15}A'', A_{16}\backslash A''a_{24}\}$
$\xrightarrow{\text{R1}} \{\}, \{A_{12}\backslash a_{22}A', A_{23}\backslash A'a_{15}A'', A_{16}\backslash A''a_{24}\} \quad \Diamond$

2.2 $\xrightarrow{\text{R10}} \{A_{12}a_{15}A_{16}{=}a_{22}A_{23}|a_{24}\}, \{\} \xrightarrow{\text{R10}} \{A_{12}a_{15}A_{16}{=}a_{22}A_{23}a_{24}|\varepsilon\}, \{\}$
$\xrightarrow{\text{R5}} \{a_{15}A_{16}{=}\varepsilon|\varepsilon\}, \{A_{12}\backslash a_{22}A_{23}a_{24}\} \quad \Diamond$

The only successful transformation sequence, leading to a tuple $\{\}, \sigma_J$, yields the only most general unifier $\sigma_J{=}\{A_{12}\backslash a_{22}A', A_{23}\backslash A'a_{15}A'', A_{16}\backslash A''a_{24}\}$ where A' and A'' are new introduced variables. See [Otten and Kreitz, 1996(a)] for an extensive example.

Theorem 8, whose proof can be found in [Otten and Kreitz, 1996(a)], together with the proof of theorem 5 shows that our uniform path checking algorithm is correct and complete for intuitionistic logic as well.

Corollary 9 (Correctness and Completeness for Intuitionistic Logic). *The function* `prove(`F`,1)` *from figure 4 together with initialization and unification from table 2 succeeds iff F is valid in intuitionistic logic. In this case it returns an admissible combined substitution $\sigma = (\sigma_Q, \sigma_J)$ which makes every path through some F^μ complementary.*

The first published theorem prover using the technique of prefix unification is ileanTAP [Otten, 1997]. The basic path checking algorithm it based on an analytic tableau calculus. An additional string unification is used to ensure the particular restrictions in intuitionistic logic. Although not connection-driven, this very compact prover already yields an impressive performance.

5 Modal Logics

Modal logics allow the formalization of knowledge and belief. For this purpose, two modal operators $\Box$ and $\Diamond$ are added to classical logic, where $\Box P$ means that P is necessarily true and $\Diamond P$ means that P is possibly true. As the precise meaning of *necessity* and *possibility* depends on the assumptions about the consequences of knowledge and belief, a variety of modal logics has been developed, each with different logical laws for $\Box$ and $\Diamond$. In the following we shall focus on the modal logics T, D, D4, S4, and S5 in their constant, cumulative, and varying domain variants.

Proof theoretically, the modal operators $\Box$ and $\Diamond$ behave similarly to the quantifiers $\forall$ and $\exists$. This requires us to introduce additional modal types ν and π, a modal multiplicity μ_M, modal prefixes similar to the prefixes in intuitionistic logic, a modal substitution σ_M, and an admissibility condition that depends on the modal logics and the selected domain variant. After adapting the notion of complementarity (i.e. the function `unify-check`) accordingly, our general proof search procedure can be applied to modal logics as well.

The *modal type* of a position labelled with $\Box$ or $\Diamond$ is defined according to the following table.

principal type ν	$(\Box A)^1$	$(\Diamond A)^0$	*principal type* π	$(\Box A)^0$	$(\Diamond A)^1$
successor polarity	A^1	A^0	successor polarity	A^0	A^1

Positions of type π will be viewed as constants in a prefix string while ν-positions are considered to be variables (denoted by capital letters in examples). This makes it possible to use unification to determine the π-positions that must be reduced before a ν position in a valid sequent proof.

Let $u_1 < u_2 < \ldots < u_n < u$ be the elements of $\Pi \cup \nu$ (the positions of type π or type ν) that dominate the atomic position u in the formula tree. The *prefix* $\mathit{pre}_M(u)$ of u is defined as $\mathrm{pre}_M(u) := u_1 u_2 \ldots u_n$ for the modal logics D, D4, S4 and T, and as $\mathrm{pre}_M(u) := u_n$ for S5.[4] A *modal substitution* σ_M is a mapping from the positions of type ν to (possibly empty) strings over $\Pi \cup \nu$. It induces a relation $\sqsubset_M \subseteq (\nu \cup \Pi) \times \nu$ in the following way: if $\sigma_M(u)=p$, then $v \sqsubset_M u$ for all v occurring in p.

domain $\setminus \mathcal{L}$	T	D	D4	S4	S5
·constant	–	–	–	–	–
·cumulative	$\sigma_M(\mathrm{pre}_M(u)) = \sigma_M(\mathrm{pre}_M(v))q$		$\sigma_M(\mathrm{pre}_M(u)) = \sigma_M(\mathrm{pre}_M(v))q^*$		–
·varying	$\sigma_M(\mathrm{pre}_M(u)) = \sigma_M(\mathrm{pre}_M(v))$				
accessibility	$\|\sigma_M(w)\| \leq 1$	$\|\sigma_M(w)\| = 1$	$\|\sigma_M(w)\| \geq 1$	–	–

Table 4: Domain/accessibility conditions for modal logics

Definition 10 (Complementarity in Modal Logics).
Let $\mathcal{L}$ be a modal logic and $\sigma{:=}(\sigma_Q, \sigma_M)$ be a *combined substitution* consisting of a first-order substitution σ_Q and a modal substitution σ_M.

1. σ is *$\mathcal{L}$-admissible* iff the *reduction ordering* $\lhd := (< \cup \sqsubset_Q \cup \sqsubset_M)^+$ is irreflexive, the *domain condition* in table 4 holds for all $u \in \Gamma$ and all $v \in \Delta \cup \Gamma$ occurring in $\sigma_Q(u)$ (for some $q \in \nu \cup \Pi \cup \{\varepsilon\}$, $q^* \in (\nu \cup \Pi)^*$), and the *accessibility condition* holds for all $w \in \nu$.
2. A connection $\{u, v\}$ is *σ-complementary*, iff $\sigma_Q(\mathit{label}(u)) = \sigma_Q(\mathit{label}(v))$ and $\sigma_M(\mathrm{pre}_M(u)) = \sigma_M(\mathrm{pre}_M(v))$.

Whereas in intuitionistic logic we have to consider copies of formulas of type ϕ the sequent calculi for modal logics allow copies of formulas of type ν. A *modal multiplicity* $\mu_M{:}\nu \rightarrow \mathbb{N}$ encodes the number of distinct instances of ν-subformulas that need to be considered during the proof search. It can be combined with a quantifier multiplicity μ_Q and leads to an indexed formula F^μ. With these definitions, the characterization of validity in modal logics can be formulated as for intuitionistic logic.

Theorem 11 (Matrix Characterization for Modal Logics).
Let $\mathcal{L}$ be one of the modal logics D, D4, S4, S5 or T. A formula F is valid in $\mathcal{L}$ iff there is a multiplicity $\mu := (\mu_Q, \mu_M)$, an $\mathcal{L}$-admissible combined substitution $\sigma = (\sigma_Q, \sigma_M)$, and a set of σ-complementary connections such that every path through F^μ contains a connection from this set.

A proof of this theorem can be found in [Wallen, 1990].

[4] The modal prefix in S5 is a string consisting of a single position, because the accessibility relation for S5 is an equivalence relation (see [Wallen, 1990]). If u has no predecessor of type π or type ν (i.e. $n{=}0$) then $\mathrm{pre}_M(u) := \varepsilon$ for D, D4, S4, S5, and T.

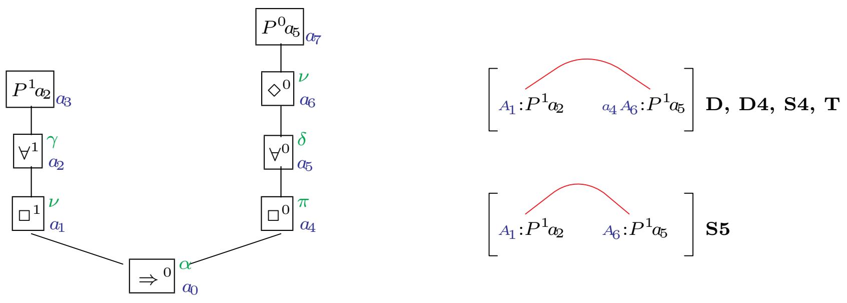

$P^0 a_5$ a_7
$\Diamond^0$ ν a_6
$\forall^0$ δ a_5
$\Box^0$ π a_4
$P^1 a_2$ a_3
$\forall^1$ γ a_2
$\Box^1$ ν a_1
$\Rightarrow^0$ α a_0
$A_1 : P^1 a_2$ $a_4 A_6 : P^1 a_5$
D, D4, S4, T
$A_1 : P^1 a_2$ $A_6 : P^1 a_5$
S5

Figure 6: Formula tree and matrix representations (with prefixes) for F_2

Example 5. Consider $F_2 \equiv \Box\forall x Px \Rightarrow \Box\forall y \Diamond Py$ and its formula tree and prefixed matrix representations in Figure 6. Let $\mu = (\mu_Q, \mu_M)$, where $\mu_Q(a_2) = \mu_M(a_1) = \mu_M(a_6) = 1$. The only path through F_2 consists of the connection $\{a_3, a_7\}$. There are three possible unifiers $\sigma_i = (\sigma_Q, \sigma_{M\,i})$ for this connection: $\sigma_Q = \{a_2 \backslash a_5\}$, $\sigma_{M\,1} = \{A_1 \backslash a_4 A_6\}$, $\sigma_{M\,2} = \{A_1 \backslash a_4, A_6 \backslash \varepsilon\}$, and $\sigma_{M\,3} = \{A_1 \backslash a_4, A_6 \backslash a_4\}$. The connection is σ_i-complementary in D, D4, S4, T (for i=1, 2) and S5 (for i=3). σ_1 is D4- and S4-admissible for constant and cumulative domains, σ_2 is S4- and T-admissible for constant, cumulative and varying domains, σ_3 is S5-admissible for constant, cumulative and varying domains. Thus F_2 is valid in D4, S4, S5 and T for the constant and cumulative domain and valid in S4, S5 and T for the varying domain.

The algorithmic characterizations of logical validity in theorems 3 and 4 hold accordingly with the modal definitions of multiplicity and complementarity for T, D, D4, S4, S5 in all domain variants. Thus we can again use our path checking algorithm presented in Figure 4 and only have to provide the logic-specific functions `initialize` and `unify-check` (see table 5).

`initialize`(F,n)	$((\emptyset,\emptyset),\ n)$
`unify-check` $(A, \bar{A}, F^\mu, (\sigma_Q, \sigma_M))$	$(\sigma_Q{}', \sigma_M{}')$ where $\sigma_Q{}' =$ `term_unify`$(\sigma_Q(\mathit{label}(A)), \sigma_Q(\mathit{label}(\bar{A})))$ $\sigma_M{}' =$ `prefix_unify`$_{\mathcal{L}}(\sigma_M(\mathtt{pre}_M(A)), \sigma_M(\mathtt{pre}_M(\bar{A})))$ if $\lhd := (< \cup \sqsubset_Q{}' \cup \sqsubset_M{}')^+$ is irreflexive and the *domain condition* holds for $u \in \Gamma$, v in $\sigma_Q(u)$

domain condition \ $\mathcal{L}$	T	D	D4	S4	S5
· constant	–	–	$\times$ [5]	–	–
· cumulative*	$\lvert V\rvert \leq \lvert U\rvert \leq \lvert V\rvert + 1$		$\lvert V\rvert \leq \lvert U\rvert$	$\lvert V\rvert \leq \lvert U\rvert$	–
· varying*	$\lvert\sigma_\varepsilon^M(U)\rvert = \lvert\sigma_\varepsilon^M(V)\rvert$		$\lvert U\rvert = \lvert V\rvert$	$\lvert\sigma_\varepsilon^M(U)\rvert = \lvert\sigma_\varepsilon^M(V)\rvert$	uni_{S5}

$\sqsubset_M{}'$ is the relation induced by $\sigma_M^* := \sigma_\varepsilon^M \circ \sigma_M{}'$ for D,S4,S5,T / $\sigma_M{}'$ for D4.
$U := \sigma_M{}'(\mathrm{pre}_M(u))$, $V := \sigma_M{}'(\mathrm{pre}_M(v))$, $\sigma_\varepsilon^M(w) \equiv \varepsilon$;
$uni_{S5} \cong$ `prefix_unify`$_{S5}(\sigma_M{}'(\mathtt{pre}_M(u)), \sigma_M{}'(\mathtt{pre}_M(v)))$

Table 5: `initialize`(F,n) and `unify-check`$(A, \bar{A}, F^\mu, \sigma)$ for modal logics

[5] We do *not* deal with the constant domain of D4, because it requires additional search when computing $\sigma_M{}'$.

`initialize`(F,n) computes a pair (σ,μ) where $\sigma{=}(\emptyset,\emptyset)$ is a combined substitution and $\mu(u){=}n$ for all $u\in\Gamma\cup\nu$. The function `unify-check`$(A,\bar{A},F^{\mu},\sigma)$ with $\sigma{=}(\sigma_Q,\sigma_M)$ computes a most general term unifier $\sigma_Q{}'$ of $\sigma_Q(\mathit{label}(A))$ and $\sigma_Q(\mathit{label}(\bar{A}))$ as well as a most general prefix unifier $\sigma_M{}'$ of $\sigma_M(\mathrm{pre}_M(A))$ and $\sigma_M(\mathrm{pre}_M(\bar{A}))$. It checks the irreflexivity of the induced reduction relation as well as a combination of the domain conditions and the accessibility conditions of the particular modal logic. It returns $(\sigma_Q{}',\sigma_M{}')$ if this combined substitution is $\mathcal{L}$-*admissible* and fails otherwise or if either of the two unifications fails.

To compute the modal substitutions we apply the same specialized string unification algorithm that we used for intuitionistic logic. We only had to modify the set of transformation rules in table 3 according to the peculiarities of the modal logics D, D4, S4, S5, and T (see [Otten and Kreitz, 1996(a)] for details).[6] The function `prefix_unify`$_{\mathcal{L}}(p,q)$ computes a set of most general unifiers for p and q with respect to the accessibility condition for the logic $\mathcal{L}$. Thus our path checking algorithm is correct and complete for the modal logics D, D4, S4, S5, and T.

Corollary 12 (Correctness and Completeness for Modal Logics).
For each of the modal logics D, D4, S4, S5, *and* T *the function* `prove(`F`,1)` *from figure 4 together with initialization and the respective unification from table 5 succeeds iff F is valid. In this case it returns an admissible combined substitution $\sigma=(\sigma_Q,\sigma_M)$ which makes every path through some F^{μ} complementary.*

6 Linear Logic Fragments

Linear Logic [Girard, 1987] is a resource sensitive logic. From a proof theoretical point of view it can be seen as the outcome of removing the rules for contraction and weakening from classical sequent calculus and re-introducing them in a controlled manner. Linear negation $^{\perp}$ is involutive like classical negation. The two different traditions for writing the sequent rule for classical conjunction result in two different conjunctions $\otimes$ and **&** and, due to the involutive negation, in two different disjunctions $⅋$ and $\oplus$. The constant `true` splits up into **1** and $\top$ for the same reason and `false` splits up into $\perp$ and **0**. The unary connectives **?** and **!** allow a controlled application of weakening and contraction. Quantifiers $\forall$ and $\exists$ can be added like in classical logic.

Linear logic connectives can be divided into the multiplicative, additive, and exponential fragment. While in the multiplicative fragment resources (i.e. formulas) are used exactly once, resource sharing is enforced in the additive fragment. By means of the exponentials formulas are marked as being reusable. All fragments can be combined freely and exist on their own right. However, the full power of linear logic comes from combining all of them.

The multiplicative fragment $\mathcal{MLL}$ can be seen as the core of linear logic. $^{\perp}$, $\otimes$, $⅋$, $\multimap$, **1**, and $\perp$ are the connectives of this fragment. Linear negation $^{\perp}$ expresses the difference between resources which are to be used up and resources which must be produced. Having a resource $G^{\perp}$ means that a resource G must be produced. Having a resource $F_1{\otimes}F_2$ is having F_1 as well as F_2. A resource $F_1\multimap F_2$ allows the construction of F_2 from F_1. The meaning of a resource $F_1 ⅋ F_2$

[6] Whereas the number of most general unifiers for the modal logics D4, S4 and T may grow exponentially, there is only one unifier for the modal logics D and S5.

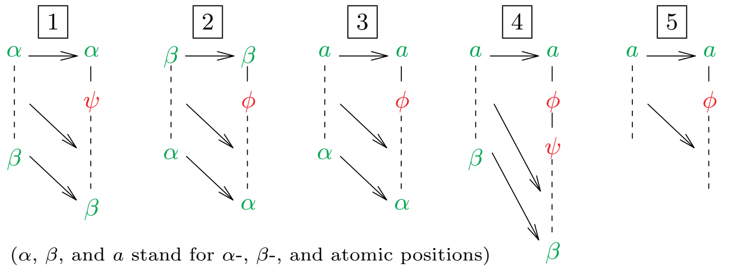

1
2
3
4
5
α
β
a
ϕ
ψ
(α, β, and a stand for α-, β-, and atomic positions)

Figure 7: Construction of a position tree by inserting special positions

is explained best by its equivalence to ${F_1}^\perp \multimap F_2$ and to ${F_2}^\perp \multimap F_1$. Having a resource **1** has no impact while nothing can be constructed when $\perp$ is used up.

As validity in linear logic can be described in terms of a sequent calculus, the matrix characterization for $\mathcal{MLL}$ [Mantel, 1996, Kreitz *et al.*, 1997] could be formulated in a way similar to Wallen's characterizations for modal and intuitionistic logics [Wallen, 1990], although semantically there are major differences. We define linear types, special positions, prefixes, substitutions, and admissibility conditions. Obviously, we do not need the notion of multiplicity for $\mathcal{MLL}$. But we need a minimality condition on the set of connections to express the linearity of the calculus.

principal type α	$(A\otimes B)^1$	$(A \parr B)^0$	$(A \multimap B)^0$	*principal type* o	$(A^\perp)^1$
successor polarity	A^1, B^1	A^0, B^0	A^1, B^0	successor polarity	A^0
principal type β	$(A\otimes B)^0$	$(A \parr B)^1$	$(A \multimap B)^1$	*principal type* o	$(A^\perp)^0$
successor polarity	A^0, B^0	A^1, B^1	A^0, B^1	successor polarity	A^1

The *linear type* of a position in a formula tree is defined by the above table. A *position tree* for a formula F, denoted by F as well, is obtained by inserting *special positions* of type ϕ (variable) and ψ (constant) into the formula tree of F. For this purpose we apply the rewrite rules 1–5 in Figure 7 as long as possible. The dashed lines may be replaced by an arbitrary number of o-positions (i.e. negations). For instance, when rule 1 is applied to positions u and v of type α and β with $v < u$, where only o-positions occur between v and u, a special position of type ψ is inserted as immediate predecessor of u. Despite the semantical differences, the special positions play a similar role as the ϕ- and ψ-positions in intuitionistic logic (see [Wallen, 1990]) and also have a similar proof theoretic motivation, which is discussed extensively in [Mantel, 1996, Mantel, 1998]. But instead of marking formulas with specific connectives, they separate layers of α-positions from layers of β-positions and atomic positions from all other ones.

As usual the *prefix* $pre_L(u)$ of an atomic position u is a string $u_1 u_2 \dots u_n$ where $u_1 < u_2 < \dots < u_n < u$ are special positions that dominate u in the position tree. A *linear substitution* σ_L is a mapping from Φ_L to strings over $\Phi_L \cup \Psi_L$ (the sets of ϕ- and ψ-positions). σ_L induces a relation $\sqsubset_L \subseteq \Psi_L \times \Phi_L$ in the following way: if $\sigma_L(u){=}p$, then $v \sqsubset_L u$ for all $v \in \Psi_L$ occurring in p.

The notion of complementarity in linear logic is a bit more complex than before, as we also have to put restrictions on the set of connections that is used to make each path complementary.

Definition 13 (Complementarity in $\mathcal{MLL}$).
Let σ_L be a linear substitution, F a position tree, and $\mathcal{C}on$ a set of connections.

1. σ_L is *admissible* if $\sigma_L(pre_L(v))=s_1 v$ holds whenever $s_1 v s_2 = \sigma_L(pre_L(u))$ for some position u.
2. A connection $\{u, v\}$ is *σ_L-complementary* iff $\sigma_L(pre_L(u))=\sigma_L(pre_L(v))$.
3. $\mathcal{C}on$ *spans* F iff each path through F contains at least one connection from $\mathcal{C}on$. A spanning set $\mathcal{C}on$ of connections is *minimal* for F iff no proper subset of $\mathcal{C}on$ is spanning for F.
4. $\mathcal{C}on$ is *linear* iff no atomic position of F occurs in more than one connection.
5. F is *relevant* for $\mathcal{C}on$ iff each atomic position of F is contained in at least one connection from $\mathcal{C}on$.

Note that the admissibility of σ_L does not explicitly require the irreflexivity of the induced reduction ordering, because we do not deal with quantifiers and thus do not have to use a *combined* substitution.

Lemma 14. *If a set of connections $\mathcal{C}on$ is σ_L-complementary for a position tree F then the induced reduction ordering $\lhd = (< \cup \sqsubset_L)^+$ is irreflexive.*

Proof. If there is some position u with $u \lhd u$ then there must be also one of type ϕ. Thus there are positions ϕ_i and ψ_i ($i \in \{0, \ldots, n-1\}$, $\phi = \phi_0$) such that $\phi_i < \psi_i$ and $\psi_i \sqsubset_L \phi_{(i+1) \bmod n}$ holds. This violates the admissibility condition for some $\sigma_L(pre_L(u))$ and some $v = \psi_i$. □

The σ_L-complementarity of a connection ensures that the elements of a connection cannot be separated during context split and therefore occur in an axiom in the sequent calculus. Thus unification guarantees the existence of an order of rule applications together with a context splitting from which a sequent proof can be constructed. Redundancies due to permutabilities of rules within α- and β-layers are avoided.

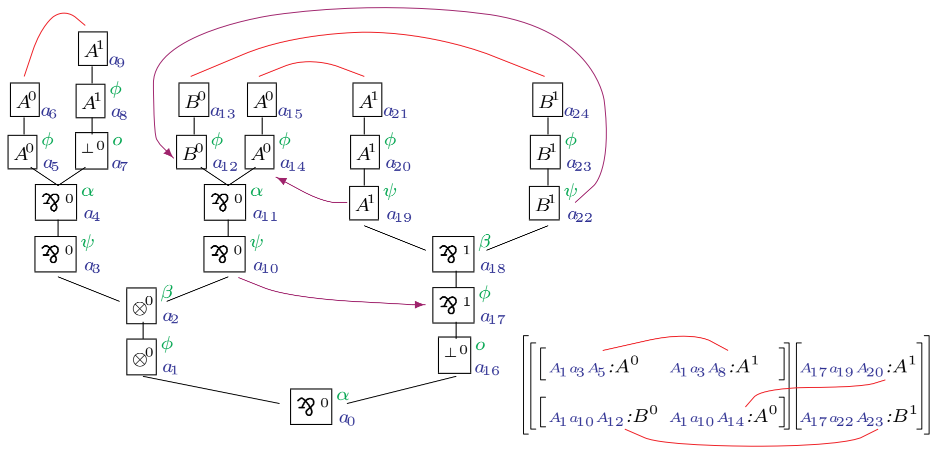
Theorem 15 (Matrix Characterization for $\mathcal{MLL}$).
A formula F in $\mathcal{MLL}$ is valid iff there exists a set of connections $\mathcal{C}on$ and an admissible linear substitution σ_L such that all connections in $\mathcal{C}on$ are σ_L-complementary, F is relevant for $\mathcal{C}on$, and $\mathcal{C}on$ is spanning and minimal for F.

A proof of this theorem can be found in [Kreitz *et al.*, 1997, Section 2].

Note that the matrix characterization does not include the linearity condition, because it is subsumed by σ_L-complementarity and minimality of $\mathcal{C}on$. Nevertheless we will include a linearity test in our proof procedure as it can be performed *while* the set of connections is being constructed. Furthermore, this allows us to avoid the expensive test for minimality completely, since we can replace it by a simple *cardinality* test.

Lemma 16. *Let $\mathcal{C}on$ be a linear spanning set of σ_L-complementary connections for F, and F be relevant for $\mathcal{C}on$. Then $\mathcal{C}on$ is minimal iff $|\mathcal{C}on| = \#_\beta + 1$ where $\#_\beta$ is the number of β-type positions in F.*

A proof of this *cardinality criterion* can be found in [Mantel, 1996] (see also [Mantel, 1998]).



$A_1 a_3 A_5 : A^0$
$A_1 a_3 A_8 : A^1$
$A_{17} a_{19} A_{20} : A^1$
$A_1 a_{10} A_{12} : B^0$
$A_1 a_{10} A_{14} : A^0$
$A_{17} a_{22} A_{23} : B^1$

Figure 8: Position tree and matrix representation (with prefixes) for F_3

Example 6. The position tree for $F_3 = ((A \parr A^{\perp}) \otimes (B \parr A)) \parr (A \parr B)^{\perp}$ and the prefixes of atomic positions is depicted in Figure 8. There are 4 atomic paths, each containing exactly one of the three connections $c_1 = \{a_6, a_9\}$, $c_2 = \{a_{15}, a_{21}\}$, and $c_3 = \{a_{13}, a_{24}\}$. $\mathcal{C}on = \{c_1, c_2, c_3\}$ is a minimal spanning set of connections for F_3 and F_3 is relevant for $\mathcal{C}on$. The admissible linear substitution

$$\sigma_L = \{A_1 \backslash \varepsilon,\ A_5 \backslash \varepsilon,\ A_8 \backslash \varepsilon,\ A_{12} \backslash a_{22},\ A_{14} \backslash a_{19},\ A_{17} \backslash a_{10},\ A_{20} \backslash \varepsilon,\ A_{23} \backslash \varepsilon\}$$

makes all connections in $\mathcal{C}on$ σ_L-complementary. Thus F_3 is valid according to Theorem 15. The ordering $\sqsubset_L$ induced by σ_L, depicted by curved arrows shows that the reduction ordering $\lhd$ is in fact irreflexive.

Since the matrix characterization for $\mathcal{MLL}$ is formally very similar to the characterizations for intuitionistic and modal logics, only a few modifications are necessary to adapt our general path checking to linear logic. In addition to the active path and the partial substitution we also have to consider the set $\mathcal{C}on$ of connections computed so far. Besides checking unifiability of a new connection $\{A, \bar{A}\}$ the function `unify-check` will also have to test linearity before adding $\{A, \bar{A}\}$ to $\mathcal{C}on$. Finally, the test for minimality and relevance can only be invoked after the complete spanning set of connections has been found. Because of lemma 16 we can replace the minimality test by the cheaper cardinality test.

Figure 9 presents the generalized path checking algorithm for linear logic. Since we do not deal with quantifiers, there is no need for iterating a multiplicity and thus our algorithm is able to decide the validity of a $\mathcal{MLL}$-formula.[7] The algorithm is now parameterized by three functions. `initialize` initializes the substitution σ_L. `unify-check`$(A, \bar{A}, F, \sigma_L, \mathcal{C}on)$ tries to compute an admissible substitution that unifies A and $\bar{A}$, and extends σ_L, provided that $\mathcal{C}on \cup \{A, \bar{A}\}$ is linear. Finally `mini_rele`$(F, \mathcal{C}on)$ tests minimality and relevance. Table 6 provides the corresponding functions for $\mathcal{MLL}$.

[7] Because the linearity of $\mathcal{C}on \cup \{A, \bar{A}\}$ implies $\mathcal{D} \cap \mathcal{P} = \emptyset$ we have removed the test $\bar{A} \in \mathcal{P}$, which would always return `false`.

```
prove($F$,$n$)
      ⎧ $\sigma_{L1}$                                                        if this computation succeeds
      ⎪   where $(\sigma_{L1},Con_1)$ = provable($\emptyset,\emptyset,\sigma_L,\emptyset$)   and mini_rele($F$,$Con_1$)
  =   ⎨     where $\sigma_L$ = initialize($F$)
      ⎪     and $\mathcal{CON}$ = connections($F$)
      ⎩ fail                                                                otherwise

provable($\mathcal{P},\mathcal{C},\sigma_L,Con$)
      ⎧ check-extension($\mathcal{E},\sigma_L,Con$)                          if $\mathcal{E}\neq\emptyset$
  =   ⎨   where $\mathcal{E} = \{v\in\mathcal{A} \mid v\sim_\alpha\mathcal{P} \wedge v\sim_\beta\mathcal{C}\}$
      ⎩ $\sigma_L$                                                          otherwise

check-extension($\mathcal{E},\sigma_L,Con$)
      ⎧ check-connections($\mathcal{D},A,\sigma_L,Con$)                      if this computation succeeds
  =   ⎪   where $\mathcal{D} = \{\bar{A}\in\mathcal{A} \mid \{A,\bar{A}\}\in\mathcal{CON} \wedge \bar{A}\sim_\alpha(\mathcal{P}\cup\{A\})\}$
      ⎨   where $A\in\mathcal{E}$ arbitrary
      ⎩ check-extension($\{v\in\mathcal{E}|v\sim_\alpha A\},\sigma_L,Con$)      otherwise (and $\mathcal{E}\neq\emptyset$)

check-connections($\mathcal{D},A,\sigma_L,Con$)
      ⎧ provable($\mathcal{P},\mathcal{C}\cup\{A\},\sigma_{L2},Con_2$)              if this computation succeeds
      ⎪   where $(\sigma_{L2},Con_2)$ = provable($\mathcal{P}\cup\{A\},\{\bar{A}\},\sigma_{L1},Con_1$)
  =   ⎨   where $\bar{A}\in\mathcal{D}$ arbitrary
      ⎪   and $Con_1 = Con\cup\{A,\bar{A}\}$
      ⎪   and $\sigma_{L1}$ = unify-check($A,\bar{A},F,\sigma_L,Con_1$)
      ⎩ check-connections($\mathcal{D}-\{\bar{A}\},A,\sigma_L,Con$)            otherwise (and $\mathcal{D}\neq\emptyset$)
```

Figure 9: Path checking algorithm for $\mathcal{MLL}$

The minimality test can only be done after the complete set of connections that were used to show the provability of $(\emptyset,\emptyset)$ has been determined. Therefore the function `provable`, `check-extensions`, and `check-connections` do not only return the substitution σ_L but also the set Con.

`initialize`(F)	$\emptyset$
`unify-check`($A,\bar{A},F,\sigma_L,Con$)	`prefix_unify`$_L(\sigma_L(pre_L(A)),\sigma_L(pre_L(\bar{A})))$ if $\{A,\bar{A}\}\cap(\bigcup_{c\,\in\,Con} c)=\emptyset$ or $\{A,\bar{A}\}\in Con$
`mini_rele`(F,Con)	$\lvert Con\rvert = \#_\beta+1$ and $\mathcal{A}\subseteq(\bigcup_{c\,\in\,Con} c)$

Table 6: The functions `initialize`, and `unify-check`, and `mini_rele` for $\mathcal{MLL}$

To compute the linear substitution we apply the same specialized string unification algorithm that we used for intuitionistic logic but restrict ourselves to the rules R1, R3, R5, R8, R9, and R10 from table 3. Since all the prefixes to be unified in $\mathcal{MLL}$ have either the form $\psi_1\phi_1\psi_2\phi_2\ldots\psi_n\phi_n$ or all of them have the form $\phi_1\psi_2\phi_2\ldots\psi_n\phi_n$ (where $\phi_i\in\Phi_L$ and $\psi_i\in\Psi_L$), we do not need the rules R2, R4, R6, and R7 anymore.[8] The function `prefix_unify`$_L(p,q)$ computes a complete and minimal set of most general unifiers for p and q. Since the algorithmic characterizations of logical validity in theorems 3 and 4 hold accordingly for linear logic our uniform path checking algorithm is correct and complete for $\mathcal{MLL}$.

[8] The number of most general unifiers is finite but may grow exponentially with the length of the prefixes.

Corollary 17 (Correctness and Completeness for $\mathcal{MLL}$).
The function `prove(`F`)` *from figure 9 together with the functions from table 6 succeeds if F is valid in $\mathcal{MLL}$ and returns an admissible linear substitution σ_L which makes every path through some F complementary. It terminates with failure if F is not valid.*

Attempts for obtaining matrix characterizations in fragments of linear logic have been made on the basis of acyclic connection graphs [Fronhöfer, 1996, Galmiche, 1996, Galmiche, 1999]. This acyclicity condition is very close to proof nets and therefore these attempts will very likely have similar limitations. In contrast to that our approach is based on prefixes and unifies the advantages of several approaches to proof search in linear logic without sharing their problems. There is also no need for transformations in negational normal form or for following the connectives during proof search (an advantage also over Tammet's proof search strategies [Tammet, 1994]). Prefix unification appears to be as efficient as the acyclicity test implicitly contained in [Fronhöfer, 1996] but yields informations which make the conversion into sequent proofs easier. Checking the *cardinality criterion* instead of an exponential minimality test is another improvement.

The tableau prover linTAP [Mantel and Otten, 1999] follows a similar approach to the one presented here. It proves the validity of a given formula by constructing an analytic tableau and uses an additional prefix unification to deal with the problem of context splitting. linTAP is not only a very compact prover but compares favourable with other (larger) implementations.

7 Conclusion

We have presented a uniform proof search procedure for classical and non-classical logics that generalizes our previously developed proof procedures for intuitionistic [Otten and Kreitz, 1995], modal [Otten and Kreitz, 1996(b)], and multiplicative linear logic [Kreitz *et al.*, 1997]. It is based on a unified representation of matrix characterizations for logical validity, which enables us to abstract from the semantical differences between various logics and to focus on structural similarities during proof search. Our procedure consists of a connection-driven general *path checking algorithm* and a component for checking the *complementarity* of two atomic formulas according to the peculiarities of a given logic. By presenting appropriate components for classical logic, intuitionistic logic, the modal logics D, D4, S4, S5, and T in their constant, cumulative, and varying domain variants, and the multiplicative fragment of linear logic, we have demonstrated that our procedure is suited to deal with a rich variety of logics in a simple and efficient way.

We believe that our proof search procedure can be further extended to other non-classical logics for which a matrix characterization can be developed. There is a variety of logics, for which this seems possible. The matrix characterization for $\mathcal{MELL}$ [Mantel, 1998, Mantel and Kreitz, 1998], the multiplicative fragment of linear logic with the exponentials **?** and **!** and the multiplicative constants **1** and $\perp$, for instance, provides a non-trivial extension of the characterization for $\mathcal{MLL}$. Nevertheless it could be formulated in a similar fashion, which makes an extension of our proof procedure to $\mathcal{MELL}$ feasible. The addition of quantifiers to $\mathcal{MELL}$ can very likely be handled by combined substitutions as in intuitionistic or modal logics.[9]

We are confident that matrix characterizations for other resource sensitive logics, such as affine or relevant logics, can be developed using the methodology of [Mantel, 1998] for introducing tableau classifications for connectives, special positions, prefixes, and criteria for complementarity and resource management. Experience has shown that our path checking and prefix unification algorithms can easily be adapted once a matrix characterization has been elaborated.

Our proof procedure also allows a treatment of combinations of modal, intuitionistic, linear, and resource sensitive logics. Because of the interrelations between several (prefix and first-order) substitutions, this will make the initialization and complementarity tests more complex but we can still rely on the same general unification and path checking algorithms.

A recently developed "*constructively adequate*" matrix characterization for intuitionistic logic [Korn, 1998] describes a slightly different treatment of intuitionistic logic. It shows how to avoid expensive constructive reasoning whenever the intuitionistic validity of a formula is already implied by its classical validity. In particular, it allows to decide the intuitionistic validity of propositional formulas, which is not the case for the intuitionistic sequent calculus and Wallen's original characterization [Wallen, 1990] (see theorem 7). An extension of our proof procedure based on this characterization could possibly improve the efficiency of theorem proving in intuitionistic logic.

The first uniform tableau calculi for various modal logics and intuitionistic logic are presented in [Fitting, 1983]. Like in our approach prefixes are used to describe the peculiarities of each logic. Uniform tableau methods for all 15 classes of modal logics are described in [Beckert and Goré, 1997]. These methods are limited to propositional logic and do not use a specialized string unification which is necessary for a more efficient proof search. There are also approaches for systematizing calculi for finite valued logics [Carnielli, 1987, Hähnle, 1993, Surma, 1984]. These approaches however, do not aim at efficient proof *search* procedures and are less compact than uniform matrix-representations.

Besides proving the validity of a given formulas our proof procedure can also be used to guide the development of proofs in interactively controlled proof assistants and thus to combine interactive and automated theorem proving. Our key concept is to view matrix-based proof methods as proof planners that do not underlie the typical limitations of sequent or natural deduction calculi when searching for a solution to a given problem. Once a matrix proof has been found, we only have to convert it into a sequent proof that can be executed (and checked) by the interactive proof system. As matrix proofs are nothing but compact representations of sequent proofs, converting them into sequent proofs means reintroducing the redundancies that had been avoided during proof search.

The algorithm presented in [Kreitz and Schmitt, 1999, Kreitz *et al.*, 1997] converts matrix proofs for all the logics discussed in this paper into the respective sequent proofs. For this purpose, it essentially traverses a formula tree

[9] For this purpose we have to re-introduce the iteration of multiplicities into the path checking algorithm presented in Figure 9. All the other conditions can be integrated into `initialize`, `unify-check`, and `mini_rele`.

F^μ in an order that respects the induced reduction ordering $\lhd = (< \cup \sqsubset_Q \cup \sqsubset_J)^+$ generated during proof search. It selects the appropriate sequent rule for each visited node and instantiates quantifiers according to the substitution σ_Q. The technical details of the conversion procedure are quite subtle, as it tries to avoid additional search during the conversion process. It is, however, equally uniform as our path checking algorithm and can easily be combined with our procedure in order to guide the proof search in interactive proof systems.

Viewing matrix proofs as proof plans also suggests the integration of additional proof planning techniques into our proof procedure. Rewrite techniques such as rippling [Bundy *et al.*, 1993], for instance, have successfully been used as proof planners for inductive theorem proving but are relatively weak as far as predicate logic reasoning is concerned. A recent extension of rippling [Pientka and Kreitz, 1998] has demonstrated that rippling techniques and proof search methods can be combined and used successfully for constructive theorem proving and the synthesis of inductive programs. Since the only weakness of this approach lies in a sequent-based proof search, we are currently exploring its integration into our matrix-based proof method [Kreitz *et al.*, 1998, Section 5]. Essentially this will lead to an integration of rippling techniques into the unification process and to further reductions of the search space in inductive theorem proving.

In order to allow practical experiments with our approach we have provided a reference implementation of our proof search procedure in Prolog. We intend to elaborate optimizations like integrating a decision procedure for the propositional cases of classical, intuitionistic and modal logics. Furthermore we study appropriate versions of efficiency improvements used in theorem provers based on the original connection method [Letz *et al.*, 1992, Bibel *et al.*, 1994], such as increasing the multiplicities dynamically *during* the path checking process. We also intend to provide implementations of our algorithm in ML or C in order to allow realistic comparisons and an integration of our procedure into existing interactive proof systems.

Acknowledgements

We thank Heiko Mantel for providing the matrix characterization for $\mathcal{MLL}$ that helped us to extend our proof procedure beyond intuitionistic and modal logics.

References

[Abdulrab and Pecuchet, 1990] H. Abdulrab and J.-P. Pecuchet. Solving word equations. In C. Kirchner, ed., *Unification*, pages 353–375. Academic Press, 1990.

[Altenkirch *et al.*, 1994] T. Altenkirch, V. Gaspes, B. Nordström, B. von Sydow. *A user's guide to ALF.* University of Göteborg, 1994.

[Beckert and Goré, 1997] B. Beckert and R. Goré. Free variable tableaux for propositional modal logics. *6th International Conference on Analytic Tableaux and Related Methods*, LNAI 1227, pp. 91–106, 1997.

[Beckert and Posegga, 1994] B. Beckert and J. Posegga. leanTAP: lean, tableau-based theorem proving. *12th Conference on Automated Deduction*, LNAI 814, pp. 793–797, 1994.

[Bibel, 1981] W. Bibel. On matrices with connections. *Journal of the ACM*, 28:633–645, 1981.

[Bibel, 1987] W. Bibel. *Automated Theorem Proving.* Vieweg, 1987.

[Bibel *et al.*, 1994] W. Bibel, S. Brüning, U. Egly, T. Rath. Komet. *12th Conference on Automated Deduction*, LNAI 814, pp. 783–787, 1994.

[Bibel *et al.*, 1996] W. Bibel, D. Korn, C. Kreitz, S. Schmitt. Problem-oriented applications of automated theorem proving. *Design and Implementation of Symbolic Computation Systems*, LNCS 1126, pp. 1–21, 1996.

[Bundy *et al.*, 1993] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, A. Smaill. Rippling: a heuristic for guiding inductive proofs. *Artificial Intelligence*, 62(2):185–253, 1993.

[Carnielli, 1987] W. A. Carnielli. Systematization of finite many-valued logics through the method of tableaux. *Journal of Symbolic Logic*, 52(2):473–493, 1987.

[Constable *et al.*, 1986] R. Constable, S. Allen, M. Bromley, et al. *Implementing Mathematics with the NuPRL proof development system.* Prentice Hall, 1986.

[Dowek, 1991] G. Dowek et al. *The Coq proof assistant user's guide.* Institut National de Recherche en Informatique et en Automatique, Report RR 134, 1991.

[Fitting, 1983] M. Fitting. *Proof Methods for Modal and Intuitionistic Logic.* D. Reidel, 1983.

[Fronhöfer, 1996] B. Fronhöfer. *The action-as-implication paradigm.* CS Press, 1996.

[Galmiche, 1996] D. Galmiche. Connection methods and proof nets construction in linear logic fragments (abstract). *CADE–13 workshop on proof search in type-theoretic languages*, 1996.

[Galmiche, 1999] D. Galmiche. Connection methods in linear logic and proof nets construction. *Theoretical Computer Science*, 1999 (to appear).

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

[Hähnle, 1993] R. Hähnle. *Automated deduction in multiple-valued logics.* Clarendon Press, 1993.

[Korn, 1998] D. Korn. Konstruktiv adäquate Beweisautomatisierung für intuitionistische Logik. *PhD Thesis*, TU Darmstadt, Germany, 1998.

[Kreitz *et al.*, 1996] C. Kreitz, J. Otten, S. Schmitt. Guiding Program Development Systems by a Connection Based Proof Strategy. *5th International Workshop on Logic Program Synthesis and Transformation*, LNCS 1048, pp. 137–151, 1996.

[Kreitz *et al.*, 1997] C. Kreitz, H. Mantel, J. Otten, S. Schmitt. Connection-based proof construction in linear logic. *14th Conference on Automated Deduction*, LNAI 1249, pp. 207–221, 1997.

[Kreitz *et al.*, 1998] C. Kreitz, J. Otten, S. Schmitt, B. Pientka. Matrix-based constructive theorem proving. *Intellectics and Computation. Essays in honor of Wolfgang Bibel*, Kluwer, 1999 (to appear).

[Kreitz and Schmitt, 1999] C. Kreitz and S. Schmitt. A uniform procedure for converting matrix proofs into sequent-style systems. *Journal of Information and Computation*, 1999 (to appear).

[Letz *et al.*, 1992] R. Letz, J. Schumann, S. Bayerl, W. Bibel. SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning*, 8:183–212, 1992.

[Manakin, 1977] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sb.*, 103(145):147–236, 1977. English translation: American Mathematical Soc. Translations (2), vol. 117, 1981.

[Mantel, 1996] H. Mantel. Eine Matrixcharakterisierung für ein Fragment der linearen Logik. *Diplomarbeit*, TU Darmstadt, Germany, 1996.

[Mantel, 1998] H. Mantel. Developing a matrix characterization for $\mathcal{MELL}$. Research Report RR-98-03, DFKI Saarbrücken, Germany, 1998.

[Mantel and Kreitz, 1998] H. Mantel and C. Kreitz. A matrix characterization for $\mathcal{MELL}$. *6th European Workshop on Logics in Artificial Intelligence*, LNAI 1489, pp. 169-183, 1998.

[Mantel and Otten, 1999] H. Mantel and J. Otten. linTAP: A tableau prover for linear logic. *Proc. 8th TABLEAUX Conference*, LNAI, 1999 (to appear).

[Martelli and Montanari, 1982] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM TOPLAS*, 4:258–282, 1982.

[Matiyasevič, 1968] Yu. V. Matiyasevič. A connection between systems of word and length equations and Hilbert's tenth problem. English Translation in: Sem. Math. V.A. Steklov Math. Inst. Leningrad 8 (1968).

[Ohlbach, 1988] H. J. Ohlbach. A resolution calculus for modal logics. Ph.D. Thesis (SEKI Report SR-88-08), Universität Kaiserslautern, 1988.

[Otten and Kreitz, 1995] J. Otten and C. Kreitz. A connection based proof method for intuitionistic logic. *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, pp. 122–137, 1995.

[Otten and Kreitz, 1996(a)] J. Otten and C. Kreitz. T-String-unification: unifying prefixes in non-classical proof methods. *5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 1071, pp. 244–260, 1996.

[Otten and Kreitz, 1996(b)] J. Otten and C. Kreitz. A uniform proof procedure for classical and non-classical logics. *KI-96: Advances in Artificial Intelligence*, LNAI 1137, pp. 307–319, 1996.

[Otten, 1997] J. Otten. ileanTAP: An Intuitionistic Theorem Prover. *Proc. 6th TABLEAUX Conference*, LNAI 1227, pp. 307–312, 1997.

[Paulson, 1990] L. Paulson. Isabelle: The next 700 theorem provers. *Logic and Computer Science*. pp. 361–386. Academic Press, 1990.

[Pientka and Kreitz, 1998] B. Pientka and C. Kreitz. Instantiation of existentially quantified variables in inductive specification proofs. In *4th International Conference on Artificial Intelligence and Symbolic Computation*, LNAI 1476, pp. 247–258, 1998.

[Robinson, 1965] A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Schmidt, 1998] R. Schmidt. E-unification for subsystems of S4. *RTA-98*, LNCS 1379, pp. 106–120, 1998.

[Schmitt and Kreitz, 1995] S. Schmitt and C. Kreitz. On transforming intuitionistic matrix proofs into standard-sequent proofs. *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, pp. 106–121, 1995.

[Schmitt and Kreitz, 1996] S. Schmitt and C. Kreitz. Converting non-classical matrix proofs into sequent-style systems. *13th Conference on Automated Deduction*, LNAI 1104, pp. 418–432, 1996.

[Surma, 1984] S. Surma. An algorithm for axiomatizing every finite logic. *Computer Science and Multiple-Valued Logic*, pp. 143–149. North-Holland, 1984.

[Tammet, 1994] T. Tammet. Proof strategies in linear logic. *Journal of Automated Reasoning*, 12:273–304, 1994.

[Wallen, 1990] L. Wallen. *Automated deduction in nonclassical logic.* MIT Press, 1990.

[Wos *et al.*, 1990] L. Wos et al. Automated reasoning contributes to mathematics and logic. *10th Conference on Automated Deduction*, LNCS 449, pp. 485–499, 1990.