Connection-Based Proof Construction in Linear Logic

C. Kreitz[1] H. Mantel[2] J. Otten[3] S. Schmitt[3]

[1] Department of Computer Science, Cornell University
Ithaca, NY 14853, USA
kreitz@cs.cornell.edu

[2] Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
mantel@dfki.uni-sb.de

[3] Fachgebiet Intellektik, Fachbereich Informatik, Technische Hochschule Darmstadt
Alexanderstr. 10, 64283 Darmstadt, Germany
{jeotten,steph}@intellektik.informatik.th-darmstadt.de

Abstract. We present a matrix characterization of logical validity in the multiplicative fragment of linear logic. On this basis we develop a matrix-based proof search procedure for this fragment and a procedure which translates the machine-found proofs back into the usual sequent calculus for linear logic. Both procedures are straightforward extensions of methods which originally were developed for a uniform treatment of classical, intuitionistic and modal logics. They can be extended to further fragments of linear logic once a matrix characterization has been found.

1 Introduction

Linear logic [12] is often viewed as the most adequate formalism for reasoning about action and change in real world applications. Formulas can be considered as resources which disappear after their use unless they are explicitly marked as reusable. No *frame axioms* about the environment [18] need to be stated and one only has to deal with axioms about those objects which are involved in the action. Proof search in linear logic will therefore have many useful applications such as resource sensitive logic programming [14], modeling concurrent computation by petri nets [11], and planning [17].

Because of the expressivity of logic, however, reasoning in linear logic is difficult to automate. Propositional linear logic is already undecidable. In order to prove a linear logic formula syntactically one has to rely on either sequent calculi or proof nets [12,6], a kind of natural deduction system with multiple conclusions. The former cover all of linear logic but are not useful for efficient proof search because of the many redundancies contained in them. Attempts to remove permutabilities from sequent proofs [1,10] and to add proof strategies [26] have provided some improvements but the main difficulties still remain. Proof nets, on the other hand, are applicable only to a small fragment of the logic. In order to handle the other parts one has to introduce the concept of boxes [12] which again cause major problems for automated proof search. Although there has been progress in removing some boxes [13] efficient proof search for full linear logic appears to be beyond the scope of proof nets at this point of time.

William McCune (Ed.): 14th International Conference on Automated Deduction, CADE-14, LNAI 1249, pp. 207–221, 1997.

In classical and many non-classical logics *matrix characterizations* of logical validity have successfully been used as foundation for efficient proof search methods. They yield a very compact representation of the search space and thus avoid many kinds of redundancies which usually occur in the sequent calculus and tableaux proof search methods. Originally developed as foundation of Bibel's connection method for classical logic [2,4] they have later been extended to non-classical logics by Wallen [27]. Wallen's formulation serves as a basis of a uniform proof method for a rich variety of logics [19,21] and also allows to transform matrix proofs into sequent-style proofs by a uniform procedure [23,24].

By Wallen's conjecture [27] matrix methods can be developed for any logic which has the same primary properties as classical logic. The *linear connection method* [3] has demonstrated that matrix methods can be resource sensitive. A desire for a matrix characterization of linear logic has already been expressed in [9]. Because of a superficial similarity between matrix characterizations and proof nets it is very likely that this can be achieved at least for those fragments which can be handled by proof nets. On the other hand, as far as proof search is concerned, matrix methods have proven to be a more general approach which does not underly the limitations of proof nets and may therefore apply to a larger fragment of linear logic. Therefore it is reasonable to develop matrix characterizations for various fragments of linear logic and to extend both the uniform proof method and the transformation procedure accordingly. The resulting combined procedure will then be able to efficiently search for a matrix proof of a linear logic formula and to present it in the more convenient sequent calculus.

In this paper we begin this work by investigating the multiplicative fragment of linear logic ($\mathcal{MLL}$). We shall develop a matrix characterization of logical validity in $\mathcal{MLL}$ whose formulation is close to Wallen's characterization of validity in modal logics [27] and prove it to be correct and complete (Section 2). On this basis we shall extend our uniform proof method into one that generates matrix proofs for $\mathcal{MLL}$ (Section 3) and our uniform transformation procedure into one that converts the matrix proof back into a sequent proof (Section 4). Finally we shall discuss other recent approaches to reasoning within fragments of linear logic, current and future work, and evidence which makes us confident that extensions of our methods to larger parts of linear logic are possible.

2 A Matrix Characterization of Logical Validity in $\mathcal{MLL}$

Linear Logic [12] 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 $\mathbf{1}$ and $\top$ for the same reason and `false` splits up into $\perp$ and $\mathbf{0}$. 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. ${}^{\perp}$, $\otimes$, $\parr$, $\multimap$, $\mathbf{1}$, 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 are to 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 $F_1 \parr F_2$ is explained best by its equivalence to ${F_1}^{\perp} \multimap F_2$ and to ${F_2}^{\perp} \multimap F_1$. Having a resource $\mathbf{1}$ has no impact while nothing can be constructed when $\perp$ is used up.

Validity of a linear logic formula can be proved syntactically by using a sequent calculus. For multi-sets Γ and Δ of formulas $\Gamma \vdash \Delta$ is called a sequent. It can be understood as the specification of a transformation where Δ is constructed from Γ. All formulas in Γ are connected implicitly by $\otimes$ while all formulas in Δ are connected implicitly by $\parr$. There is a close relation between $\vdash$ and linear implication $\multimap$. Thus, sequents clearly lie in the multiplicative realm.

The sequent calculus $\mathcal{L}in_M$ for the multiplicative fragment without constants is depicted in Figure 1. In a rule the sequents above the line are called *premises*. The one below is the *conclusion*. A *principal formula* is a formula which occurs in the conclusion but not in any premise. Formulas which occur in a premise but not in the conclusion are called *active*. All other formulas are the *context*.

identity

$$\frac{}{A \vdash A}\ \mathit{axiom}$$

negation

$$\frac{\Gamma \vdash \Delta, G}{\Gamma, G^{\perp} \vdash \Delta}\ {}^{\perp}l \qquad \frac{\Gamma, F \vdash \Delta}{\Gamma \vdash \Delta, F^{\perp}}\ {}^{\perp}r$$

multiplicative fragment

$$\frac{\Gamma, F_1, F_2 \vdash \Delta}{\Gamma, F_1 \otimes F_2 \vdash \Delta}\ \otimes l \qquad \frac{\Gamma_1 \vdash \Delta_1, G_1 \quad \Gamma_2 \vdash \Delta_2, G_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, G_1 \otimes G_2}\ \otimes r$$

$$\frac{\Gamma_1, F_1 \vdash \Delta_1 \quad \Gamma_2, F_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2, F_1 \parr F_2 \vdash \Delta_1, \Delta_2}\ \parr l \qquad \frac{\Gamma \vdash \Delta, G_1, G_2}{\Gamma \vdash \Delta, G_1 \parr G_2}\ \parr r$$

$$\frac{\Gamma_1 \vdash \Delta_1, G_1 \quad \Gamma_2, F_1 \vdash \Delta_2}{\Gamma_1, \Gamma_2, G_1 \multimap F_1 \vdash \Delta_1, \Delta_2}\ \multimap l \qquad \frac{\Gamma, F_1 \vdash \Delta, G_1}{\Gamma \vdash \Delta, F_1 \multimap G_1}\ \multimap r$$

Fig. 1. The sequent calculus $\mathcal{L}in_M$ for $\mathcal{MLL}$

In analytic proof search one starts out with the sequent to be proven and reduces sequents by application of rules until the *axiom*-rule can be applied. There are several choice points within this process. First, a principal formula in the sequent must be chosen. Due to the restriction to the multiplicative fragment this choice already implies the choice of the rule by which the formula shall be reduced. Second, when applying $\otimes r$, $\parr\, l$, or $\multimap l$ the context of the sequent must be partitioned onto the premises. This is called *context splitting*. Several solutions have been proposed in order to optimize these choices.

Resource management systems have been proposed in [5] as efficient techniques for splitting contexts. If subsequent applications of two rules have no impact on each other their order is unimportant. In proof search it suffices just to consider one possible order. This phenomenon is called permutability of rules and has been investigated for linear logic in [1,10,26]. As solutions to fix an order for such rules the focusing principle [1], normal proofs [10], and proof search strategies [26] have been proposed. Though being improvements compared to simple sequent calculus proof search all these proposals suffer from that they are still connective oriented. During proof search the state of a proof under construction needs to be stored at every choice point in order to make backtracking in case of a later failure possible. This causes major notational redundancies.

Matrix proof methods avoid these redundancies which results in an improved efficiency compared to sequent based proof search. They are built on notions of polarities, position trees, prefixes, paths, connections, and substitutions.

Polarities, Types, Position–trees, and Prefixes. A *signed formula* $\langle F, k\rangle$ relates a formula F to a *polarity* $k \in \{0,1\}$. The *components* of a signed formula consist of an immediate sub-formula of F and a polarity as depicted in Table 1. The polarity indicates whether the number of explicit and implicit negations during the above decomposition is even (polarity 0) or odd (polarity 1). In a derivation of a signed formula $\langle F, 0\rangle$ a sub-formula $\langle F', k'\rangle$ occurs in the succedent of sequents only for $k' = 0$, and for $k' = 1$ in the antecedent only. Signed formulas are classified by *(principal) types* (α, β, o). In the case of negation we decided to deviate from the usual tableaux scheme for reasons which will become clear later on. Since each component of a signed formula $\langle F, k\rangle$ contains an immediate sub-formula F' of F a relation $\succ$ such that $F' \succ F$ is induced. The transitive closure of $\succ$ shall be an ordering $\gg$. As usual, the *formula tree* of F is a graph with all sub-formulas of F as nodes and edges indicating $\succ$.

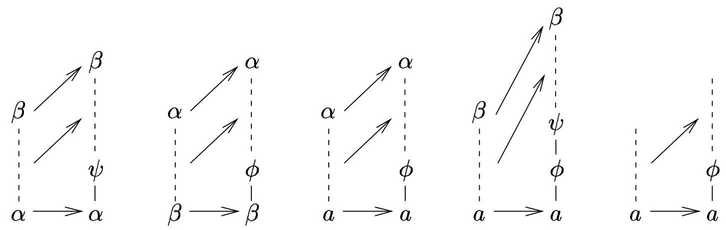
α	$\langle G_1 \parr G_2, 0\rangle$	$\langle F_1 \otimes F_2, 1\rangle$	$\langle F_1 \multimap G_1, 0\rangle$
b_1	$\langle G_1, 0\rangle$	$\langle F_1, 1\rangle$	$\langle F_1, 1\rangle$
b_2	$\langle G_2, 0\rangle$	$\langle F_2, 1\rangle$	$\langle G_1, 0\rangle$

β	$\langle F_1 \parr F_2, 1\rangle$	$\langle G_1 \otimes G_2, 0\rangle$	$\langle G_1 \multimap F_1, 1\rangle$
b_1	$\langle F_1, 1\rangle$	$\langle G_1, 0\rangle$	$\langle G_1, 0\rangle$
b_2	$\langle F_2, 1\rangle$	$\langle G_2, 0\rangle$	$\langle F_1, 1\rangle$

o	$\langle G_1^{\perp}, 1\rangle$	$\langle F_1^{\perp}, 0\rangle$
b_1	$\langle G_1, 0\rangle$	$\langle F_1, 1\rangle$

Table 1. Principal types and polarities of formulas

With respect to some signed formula $\langle F, 0\rangle$ occurrences of sub-formulas are abbreviated uniquely by positions from a set Pos. We use type symbols as meta-variables for positions abbreviating formulas of that type (e.g. α for a formula of type α), b for arbitrary types, and a for atomic formulas. The corresponding formula, its polarity, and its type can be retrieved from a position b by means of $lab(b)$, $pol(b)$, and $Ptype(b)$, respectively. $sform(b)$ shall denote the signed formula $\langle lab(b), pol(b)\rangle$. The relations $\succ$ and $\gg$ are defined like for formulas. $succ(b)$ denotes the set of all positions b' for which $b' \succ b$ holds. A *position tree* for a formula F is obtained from the formula tree of F by, first, marking all nodes of the tree with positions thereby removing the old markings and, second, applying the rewrite rules 1–5 in Figure 2 as long as possible. The dashed lines may be replaced by an arbitrary number of positions of type o. These rewrite rules insert *special positions* ϕ (variable) and ψ (constant) from sets Φ_L and Ψ_L (both sets disjoint with Pos) into the tree. The inserted positions separate layers of α–type positions from layers of β–type positions and atomic positions from all other ones. During the separation of layers we do not care about positions of type o. Constant special positions mark the *ports* of β-layers. For instance, when rule 1 is applied to positions α and β where $\alpha \gg \beta$ holds and only positions of type o occur between these positions a constant special position ψ is inserted as immediate predecessor of α. The motivation for the insertion of special positions will become clearer after the definition of σ_L-complementarity. Note, however, that special positions for $\mathcal{MLL}$ seperate layers of formulas instead of marking formulas with specific connectives which is done for intuitionistic logic [27].

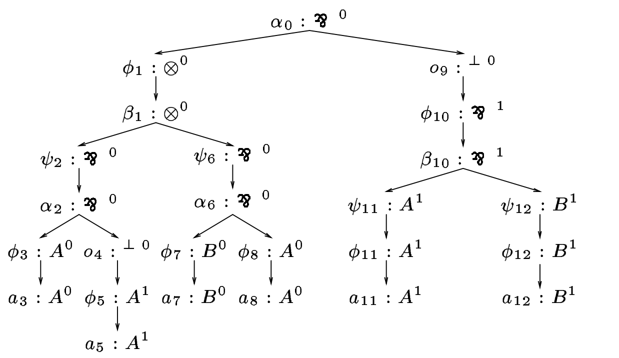


1 2 3 4 5

Fig. 2. Construction of a position tree by insertion of special positions

For each position b we define the *prefix* of that position as the string $pre(b)$ of special positions from the root of the tree to b.

Example 1. A position tree $\mathcal{T}$ for the formula $((A \parr A^{\perp}) \otimes (B \parr A)) \parr (A \parr B)^{\perp}$ and the prefix of every position in the tree is depicted below.



b	$pre(b)$
α_0, o_9	
ϕ_1, β_1	ϕ_1
ψ_2, α_2, o_4	$\phi_1.\psi_2$
ϕ_3, a_3	$\phi_1.\psi_2.\phi_3$
ϕ_5, a_5	$\phi_1.\psi_2.\phi_5$
ψ_6, α_6	$\phi_1.\psi_6$
ϕ_7, a_7	$\phi_1.\psi_6.\phi_7$
ϕ_8, a_8	$\phi_1.\psi_6.\phi_8$
ϕ_{10}, β_{10}	ϕ_{10}
ψ_{11}	$\phi_{10}.\psi_{11}$
ϕ_{11}, a_{11}	$\phi_{10}.\psi_{11}.\phi_{11}$
ψ_{12}	$\phi_{10}.\psi_{12}$
ϕ_{12}, a_{12}	$\phi_{10}.\psi_{12}.\phi_{12}$

Paths, Connections, Substitutions, and Complementarity. We define the *paths* (sets of positions) through a position tree $\mathcal{T}$ starting at the root b_0 of $\mathcal{T}$.

- $\{b_0\}$ is a path.
- If $p\cup\{b\}$ is a path where b is neither a leaf-position nor of type β then $p \cup succ(b)$ is a path.
- If $p\cup\{\beta\}$ is a path and $succ(\beta)=\{b_1, b_2\}$ then $p\cup\{b_1\}$ and $p\cup\{b_2\}$ are paths.

A path which contains atomic positions only is called *atomic path*. A *connection* c is a two-element set of positions $\{a_1, a_2\}$ for which a_1, a_2 are leaf-positions, and $lab(a_1) = lab(a_2)$ as well as $pol(a_1)\neq pol(a_2)$ holds. We say that a path p contains a connection c if $c \subseteq p$ holds.

A *prefix substitution* is an idempotent mapping $\sigma_L : \Phi_L \to (\Phi_L \cup \Psi_L)^*$ which deviates from the identity on Φ_L only for a finite subset of Φ_L. We extend σ_L to $\Phi_L \cup \Psi_L$ by the identity mapping on Ψ_L and denote the homomorphic extension to $(\Phi_L \cup \Psi_L)^*$ by σ_L as well. σ_L is *admissible* if $\sigma_L(pre(b))=s_1.b$ holds for the image $s= s_1.b.s_2$ of every prefix, i.e. substitutions shall be computed by unification.

We call a connection $\{a_1, a_2\}$ σ_L*-complementary* iff the images of the prefixes of a_1 and a_2 are identical under an admissible substitution σ_L (i.e. iff $\sigma_L(pre(a_1)) = \sigma_L(pre(a_2))$). A set C of connections is said to *span* a position tree $\mathcal{T}$ iff each path through $\mathcal{T}$ contains at least one connection from C. A spanning set C of connections is *minimal* for $\mathcal{T}$ iff removing any connection from C yields a set which is not spanning for $\mathcal{T}$. $\mathcal{T}$ is *relevant* for C iff each atomic position of $\mathcal{T}$ is contained in at least one connection from C.

Definition 1 (Complementarity of a Formula Tree). A position tree $\mathcal{T}$ is *complementary* iff there exists a set of connections C and a prefix substitution σ_L such that all connections in C are σ_L-complementary, $\mathcal{T}$ is relevant for C, and C is spanning and minimal for $\mathcal{T}$.

The σ_L-complementarity of a connection ensures that the members of a connection cannot be separated during context split and therefore occur in an initial sequent. Thus, unification guarantees the existence of an order of rule applications together with a context splitting which form a sequent proof rather than calculating one such order. Hereby avoiding redundancies due to permutabilities of rules, irrelevant reductions and notational redundancies.

Example 2. Consider the position tree in Example 1. The atomic paths through $\mathcal{T}$ are $p_1 = \{a_3, a_5, a_{11}\}$, $p_2 = \{a_3, a_5, a_{12}\}$, $p_3 = \{a_7, a_8, a_{11}\}$, and $p_4 = \{a_7, a_8, a_{12}\}$. p_1 contains the connection $c_1 = \{a_3, a_5\}$ and p_3 contains $c_2 = \{a_8, a_{11}\}$. The set $C = \{c_1, c_2, \{a_7, a_{12}\}\}$ spans $\mathcal{T}$. The admissible prefix substitution

$$\sigma_L = \{\phi_1\backslash\varepsilon,\ \phi_3\backslash\varepsilon,\ \phi_5\backslash\varepsilon,\ \phi_7\backslash\psi_{12},\ \phi_8\backslash\psi_{11},\ \phi_{10}\backslash\psi_6,\ \phi_{11}\backslash\varepsilon,\ \phi_{12}\backslash\varepsilon\}$$

makes all connections in C σ_L-complementary. $\mathcal{T}$ is complementary since C is spanning and minimal and since $\mathcal{T}$ is relevant for C.

From a prefix substitution σ_L a binary relation $\sqsubset_L \subseteq \Psi_L \times \Phi_L$ is constructed. If for some ϕ and some ψ there exist strings s_1 and s_2 such that $\sigma_L(\phi) = s_1.\psi.s_2$ then $\psi \sqsubset_L \phi$ holds. The *reduction ordering* $\lhd\ =\ (\ll \cup \sqsubset_L)^+$ is the transitive closure of this relation and the tree ordering.

Irreflexivity of a reduction ordering would become a separate requirement only if – as in first order modal logics [27] – a combined substitution is used. We expect that such a substitution would be required for linear logics when the fragment will be extended. An equivalent theorem for the propositional modal logics in [27] could be shown similarly.

Theorem 2. *If a set of connections C is σ_L-complementary for a position tree $\mathcal{T}$ then the reduction ordering $\lhd$ induced by σ_L is irreflexive.*
Proof. Assume that there exists a position b such that $b \lhd b$ holds. If any such position exists, there is one of type ϕ. Then for some index set $\{0, \ldots, n-1\}$ ($n > 1$) there are positions ϕ_i and ψ_i ($i \in \{0, \ldots, n-1\}$, $\phi = \phi_0$) such that $\phi_i \ll \psi_i$ and $\psi_i \sqsubset_L \phi_{(i+1) \bmod n}$ holds. This implies that σ_L is not admissible for the image of some prefix and for some ψ_i – a contradiction.

We state the correctness and completeness of our characterization and sketch the proofs only due to limitations of space. The complete proofs can be found in [15]. All proofs are based on a sequent calculus for linear logic and do not use any criterions from proof nets. Thus, they can serve as a basis for extensions to other fragments of linear logic.

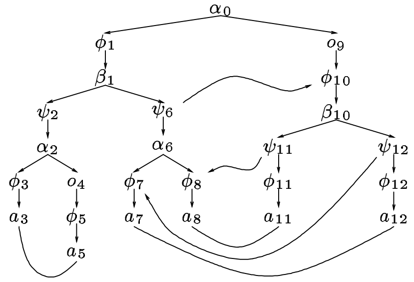
Lemma 3 (Correctness). *If the position tree $\mathcal{T}$ corresponding to a formula F is complementary then F is valid.*
Proof sketch. Since $\mathcal{T}$ is complementary there is a set of connections C and a prefix substitution σ_L such that all complementarity conditions (relevance, spanning, minimality, and σ_L-complementarity) are satisfied. Using a reduction ordering induced by σ_L a sequent proof can be constructed in an analytic fashion. Connections become the elements of initial sequents in the sequent proof. This construction uses the sequent calculus $\mathcal{K}^{\mathcal{MLL}}$ introduced in [15]. The correctness of the construction procedure is proven by induction on the weight of sequences. For the complete proof of the lemma we refer to [15].

Lemma 4 (Completeness). *If F is a valid formula then the corresponding position tree $\mathcal{T}$ is complementary.*
Proof sketch. Since F is valid a sequent proof $\mathcal{P}$ for $\cdot \vdash F$ exists. We construct a connection from every application of the *axiom*-rule in $\mathcal{P}$. A partial ordering $\sqsubset_{\mathcal{P}}$ is constructed from $\mathcal{P}$ such that if a special position b is reduced before a position b' then $b \sqsubset_{\mathcal{P}} b'$ holds. We substitute ϕ by an ordered string of constant special positions such that for every position ψ in the string the label of ψ is reduced before ϕ but after every special position $b \ll \phi$. The lengthy proof in [15] that $\mathcal{T}$ is complementary for C and σ_L uses induction on the structure of $\mathcal{P}$.

The following theorem is the foundation for matrix proof methods (based on our characterization) which prove the validity of linear logic formulas.

Theorem 5. *A formula F in the multiplicative fragment of linear logic is valid iff the corresponding position tree $\mathcal{T}$ is complementary.*
Proof. Follows directly from Lemma 3 and 4.

Example 3. The position tree $\mathcal{T}$ of the formula $((A ⅋ A^{\perp}) \otimes (B ⅋ A)) ⅋ (A ⅋ B)^{\perp}$ from Example 1 with the spanning set of connections C from Example 2 is depicted below. Connections are drawn as curved lines. The reduction ordering $\lhd$ induced by the substitution σ_L from Example 2 is depicted by arrows where straight arrows are induced by $\ll$ and curved arrows by $\sqsubset_L$. Since the position tree is complementary the formula is valid according to Theorem 5.



During automated proof search a useful reduction of the search space can be achieved by focusing on linearity. A set of connections C is *linear* iff each atomic position of $\mathcal{T}$ is contained in at most one connection from C.

Lemma 6. *If a position tree $\mathcal{T}$ is complementary for a set of connections C and an admissible substitution σ_L, then C is linear.*
Proof sketch. σ_L-complementarity guarantees that proper context splitting is possible. Minimality ensures that no unnecessary connections exist.

3 The Proof Procedure for $\mathcal{MLL}$

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, i.e. through the position tree, contain a complementary connection. In this section we will describe a general path checking algorithm as well as the corresponding complementarity test which involves an algorithm for T-string unification.

Path Checking. One possibility to perform proof search is to use an algorithm based on analytic tableaux as done in [22] for intuitionistic logic. The path checking algorithm presented in the following is driven by connections instead of the logical connectives. Once a complementary connection has been identified all paths containing this connection are deleted. This is similar to Bibel's connection method for classical logic and formulas in clausal form [4].

The theoretical basis of the following algorithm is described in detail in [21] where it is used for proof search in classical, intuitionistic and modal logics. Only a few modifications were necessary to adapt it to $\mathcal{MLL}$.

Definition 7 (α-related, β-related). Two positions u and v are *α-/β-related*, denoted $u{\sim_\alpha}v/u{\sim_\beta}v$, iff $u{\neq}v$ and the greatest common ancestor of u and v, wrt. the tree ordering $\ll$, is of principal type α/β. A position u and a set of positions $\mathcal{S}$ are *α-/β-related*, denoted $u{\sim_\alpha}\mathcal{S}/u{\sim_\beta}\mathcal{S}$, iff $u{\sim_\alpha}v/u{\sim_\beta}v$ for all $v \in \mathcal{S}$.

Remark. If two atoms are α-/β-related they appear side by side/one upon the other in the matrix representation (see Example 4).

The main function $\text{PROOF}_{MLL}(F)$ in Figure 3 returns *true* iff the formula F is valid in the multiplicative linear logic $\mathcal{MLL}$.

```
Function Proof_MLL(F)
    Input:     multiplicative formula F
    Output:    true, if, and only if, F is valid in MLL
begin Proof_MLL;
    Con := ∅;
    valid := Subproof_MLL(F, ∅, ∅);
    return valid;
end Proof_MLL.
```

Fig. 3. Function $\text{PROOF}_{MLL}(F)$

The function $\text{PROOF}_{MLL}(F)$ initializes the set of connections *Con*. After that the function SUBPROOF_{MLL} is invoked.

The function $\text{SUBPROOF}_{MLL}(F, \mathcal{P}, \mathcal{C})$ in Figure 4 implements the path checking algorithm where the set $\mathcal{P}$ is called the *active path* and the set $\mathcal{C}$ is called the *proven subgoals*. By $\mathcal{A}$ we denote the set of all atomic positions in the formula F. All variables except for $\mathcal{A}$ and *Con* are local.

```
Function Subproof_MLL(F, P, C)
    Input:     formula F, active path P⊆A, proven subgoals C⊆A
    Output:    true, if (P, C) wrt. F is provable (see [21]); false, otherwise
begin Subproof_MLL;
    if there is no A ∈ A where A∼_α P and A∼_β C then return true;
    E := ∅;   Con' := Con;
    repeat
        select A ∈ A where A∼_α (P∪E) and A∼_β C;
        if there is no such A then return false;
        E := E ∪ {A};   D := ∅;   valid := false;   noconnect := false;
        repeat
            select Ā ∈ A where Ā ∉ D  and  Ā∼_α (P∪{A})  and
                [Complementary_MLL(F, Con'∪{{A, Ā}})]  and  Line(F, Con'∪{{A, Ā}});
            if there is no such Ā
                then noconnect := true
                else  D := D ∪ {Ā};   Con := Con' ∪ {{A, Ā}};
                      valid := Subproof_MLL(F, P∪{A}, {Ā});
                      if valid=true then valid := Subproof_MLL(F, P, C∪{A});
                      if valid=true and P=∅ then valid := Mini_Rele(F, Con);
        until valid=true or noconnect=true;
    until valid=true;
    return true;
end Subproof_MLL.
```

Fig. 4. Function $\text{SUBPROOF}_{MLL}(F, \mathcal{P}, \mathcal{C})$

During the proof search the active path $\mathcal{P}$ will specify those paths which are just being investigated for complementarity. All paths which contain the active path $\mathcal{P}$ and additionally one element of the proven subgoals $\mathcal{C}$ will already have been proven complementary. The only modifications wrt. [21] are an additional set *Con* which contains the connections computed so far and the two additional functions $\text{LINE}(F, \mathit{Con})$ and $\text{MINI\_RELE}(F, \mathit{Con})$. LINE returns *true* iff *Con* is linear wrt. F. MINI_RELE returns *true* iff *Con* is minimal and relevant wrt. F and is invoked only after a spanning set *Con* has been found.[1]

[1] In this case linearity and relevance of *Con* implies minimality, if the following condition holds: $|\mathit{Con}| = \#_\beta + 1$ where $\#_\beta$ is the number of β-type positions in the formula tree of F. This *cardinality criterion* optimizes proof search in $\mathcal{MLL}$ but may not generalize to larger fragments of linear logic (see [15] for details).

T-String Unification. In our path checking algorithm we have to ensure that after adding a connection to the current set Con there still is a (multiplicative) substitution σ_L under which *all* connections are complementary. Therefore the function $\textsc{Complementary}_{MLL}(F, \mathit{Con})$ is used, which returns *true* iff there is a substitution σ_L that unifies the prefixes of the connected atoms Con, i.e. iff the set of prefix-equations $\{pre(u) = pre(v) \mid \{u, v\} \in \mathit{Con}\}$ is solvable.

To unify the set of prefixes $\Gamma = \{p_1{=}q_1, \ldots, p_n{=}q_n\}$ we use a specialized string unification which respects the restrictions on every two prefixes p and q: no character is repeated either in p nor in q and equal characters only occur within a common substring at the beginning of p and q. This restriction allows us to give an efficient algorithm computing a minimal set of most general unifiers. Similar to the ideas of Martelli and Montanari [16] rather than by giving a recursive procedure we consider the process of unification as a sequence of transformations.

We start with the given set of (prefix-) equations $\Gamma = \{p_1{=}q_1, \ldots, p_n{=}q_n\}$ and an empty substitution $\sigma_L{=}\emptyset$. Each transformation step replaces the tuple Γ, σ_L by a modified tuple $\sigma_L{}'(\Gamma'), \sigma_L{}'(\sigma_L)$ in which one equation $\{p_i{=}q_i\}$ in Γ is replaced by $\{p_i'{=}q_i'\}$ and the (modified) substitution $\sigma_L{}'$ is applied to it. The algorithm is described by transformation rules " $\{p_i{=}q_i\}, \sigma_L \rightarrow \{p_i'{=}q_i'\}, \sigma_L{}'$ " which can be applied nondeterministically to the selected equation $\{p_i{=}q_i\} \in \Gamma$.[2] The set Γ is solvable, iff there are some transformation steps transforming Γ into the empty set $\Gamma'{=}\emptyset$. In this case the (final) substitution $\sigma_L{}'$ represents an idempotent most general unifier for Γ. For technical reasons we divide the right part q_i of each equation into two parts $q_i^1|q_i^2$ where the left part contains the substring which is not yet assigned to a variable. Therefore we start with the set of prefixes $\Gamma = \{p_1 = \varepsilon|q_1, \ldots, p_n{=}\varepsilon|q_n\}$.

Definition 8 (Transformation Rules for $\mathcal{MLL}$).
Let $\mathcal{V}$ be a set of variables, $\mathcal{C}$ a set of constants, and $\mathcal{V}'$ a set of *auxiliary variables* with $\mathcal{V} \cap \mathcal{V}'{=}\emptyset$. The set of *transformation rules* for $\mathcal{MLL}$ is defined in Table 2.

R1.	$\{\varepsilon = \varepsilon\|\varepsilon\},\ \sigma_L$	$\rightarrow$	$\{\},\ \sigma_L$	
R3.	$\{Xs = \varepsilon\|Xt\},\ \sigma_L$	$\rightarrow$	$\{s = \varepsilon\|t\},\ \sigma_L$	
R5.	$\{Vs = z\|\varepsilon\},\ \sigma_L$	$\rightarrow$	$\{s = \varepsilon\|\varepsilon\},\ \{V\backslash z\}\cup\sigma_L$	
R8.	$\{Vs^+ = \varepsilon\|V_1t\},\ \sigma_L$	$\rightarrow$	$\{V_1t = V\|s^+\},\ \sigma_L$	
R9.	$\{Vs^+ = z^+\|V_1t\},\ \sigma_L$	$\rightarrow$	$\{V_1t = V'\|s^+\},\ \{V\backslash z^+V'\}\cup\sigma_L$	
R10.	$\{Vs = z\|Xt\},\ \sigma_L$	$\rightarrow$	$\{Vs = zX\|t\},\ \sigma_L$	($V{\neq}X$, and $s{=}\varepsilon$ or $t{\neq}\varepsilon$ or $X \in \mathcal{C}$)

s, t and z denote (arbitrary) strings and s^+, z^+ non-empty strings. X, V, and V_1 denote single characters with $X \in \mathcal{V} \cup \mathcal{C} \cup \mathcal{V}'$ and $V, V_1 \in \mathcal{V} \cup \mathcal{V}'$ (with $V{\neq}V_1$). $V' \in \mathcal{V}'$ is a new variable which does not occur in the substitution σ_L computed so far.

Table 2. Transformation Rules for Multiplicative Linear Logic ($\mathcal{MLL}$)

These rules are identical with the rules used in [21] and [20] which deal with intuitionistic logic. Since the prefixes to be unified in $\mathcal{MLL}$ have either the form $C_1V_1C_2V_2 \ldots C_nV_n$ or $V_1C_2V_2 \ldots C_nV_n$ (where $C_i \in \mathcal{C}$ and $V_i \in \mathcal{V}$ for $1{\leq}i{\leq}n$, $n{\geq}1$), we do not need the rules R2, R4, R6, and R7 anymore.

[2] To obtain an efficient unification procedure the order of the selected equations $\{p_i{=}q_i\}$ is essential, i.e. must be selected according to the tree ordering $\ll$.

For a comprehensive treatment of the algorithm for T-string unification together with an intuitive graphical motivation we refer to [20].

Example 4. Let $F \equiv ((A \parr A^{\perp}) \otimes (B \parr A)) \parr (A \parr B)^{\perp}$ as in Example 1. The proof of F below is marked in the matrix representation[3] of F. Furthermore the set of connections *Con* as well as the (multiplicative) substitution σ_L is given. This substitution represents the *most general* unifier for the prefixes of the connected atoms. The substitution σ_L from Example 2 is a special instance which results from replacing $\phi_1, \phi_5, \phi_{11}, \phi_{12}$, and ϕ_{20} by the empty word ε.

$$\left[\begin{bmatrix} [\, A^0 \quad A^1 \,] \\ [\, B^0 \quad \tilde{A}^0 \,] \end{bmatrix} \begin{bmatrix} \tilde{A}^1 \\ B^1 \end{bmatrix} \right] \qquad \begin{aligned} \mathit{Con} &:= \{\{A^0, A^1\}, \{B^0, B^1\}, \{\tilde{A}^1, \tilde{A}^0\}\} \\ \sigma_L &:= \{\phi_3 \backslash \phi_5,\ \phi_8 \backslash \phi_{20}\psi_{11}\phi_{11},\ \phi_7 \backslash \phi_{20}\psi_{12}\phi_{12}, \\ &\qquad \phi_{10} \backslash \phi_1\psi_6\phi_{20}\} \text{ where } \phi_{20} \text{ is a new variable.} \end{aligned}$$

4 Transforming $\mathcal{MLL}$ Matrix Proofs into Sequent Proofs

In [24,25] we have developed a conversion procedure for transforming matrix proofs into conventional sequent proofs for classical and non-classical logics. When constructing this procedure our main emphasis was the *uniformity* of the approach according to the matrix characterizations for these logics [27]. To emphasize uniformity we have developed unified representations of matrix characterizations and sequent calculi which were divided into *variant* and *invariant* parts. The division resulted in an *invariant* transformation algorithm which consults a *variant* table system reflecting different properties of the logics.

We were able to adapt our uniform transformation procedure to $\mathcal{MLL}$ by extending its variant part while leaving its general structure unchanged. In order to convert $\mathcal{MLL}$-matrix proofs into sequent proofs the procedure has to obtain a linearization of the partial reduction ordering $\lhd$. Essentially this can be done by traversing $\lhd$ but certain non-permutabilities of sequent rules which are not yet represented in $\lhd$ have to be respected as well. Similar to [24], we have achieved a "completion" of $\lhd$ by dynamically adding *wait*-labels to certain nodes which prevent the corresponding sequent rules from being applied too early. This concept fills the gap between the target calculus $\mathcal{L}in_M$ (Figure 1) of our conversion and a sequent calculus $\mathcal{K}^{\mathcal{MLL}}$ [15] on which the matrix characterization is based.

Proof Reconstruction in $\mathcal{MLL}$. Our algorithm takes as input a reduction ordering $\propto^{\star}$ which for technical reasons is generated from $\lhd$ by adding a new root w. While traversing $\propto^{\star}$ it will mark all the visited positions x as *solved* (*solved*$[x] = \top$). At the beginning, w is considered solved and its successor x is *open* for being solved next. Then the following process proceeds: an open position x will be selected and marked as solved if it is *solvable*. Afterwards the corresponding sequent rule will be constructed and the successor nodes of x will be added to the set of open positions. This means that the corresponding sub-formulas are now isolated in the actual sequent and may be reduced. This process is repeated until two solved positions form a connection which allows us to close the corresponding branch of the sequent proof with an axiom rule.

[3] We use labels instead of positions and distinguish the two atoms A^0/A^1 by a tilde.

In addition to the above traversal process there are a few subtle details that need to be taken care of. Before we explain these let us illustrate the reconstruction process by our running example.

Example 5. We take the formula $((A \parr A^{\perp}) \otimes (A \parr B)) \parr (A \parr B)^{\perp}$ and the reduction ordering $\lhd$ from Example 3. We start traversal by selecting the only open position α_0 in $\propto^{\star}$ and mark it as *solved*. From $Ptype(\alpha_0) = \alpha$ and $sform(\alpha_0) = \langle lab(\alpha_0), pol(\alpha_0) \rangle$, with $pol(\alpha_0) = 0$, we construct a sequent rule of type α reducing the operator $\parr$ of $lab(\alpha_0)$ in the succedent, i.e. $\alpha(sform(\alpha_0)) = \parr r$. For computing the sub-formulas in the rule's premises we consult Table 1.

After this step ϕ_1 and o_9, the successors of α_0, become *open* in $\propto^{\star}$. ϕ_1 is not yet solvable since the principle of *layer reductions* has to be respected, which means that a series of open α- (or β-) and o-positions has to be reduced as long as possible. To express this additional non-permutability of $\mathcal{L}in_M$–rules we *dynamically* assign a *wait*-label to ϕ_1 (a $wait_2$-label, to be precise). Hence, we will visit o_9 next, construct ${}^{\perp}r$, and mark the successor ϕ_{10} of o_9 as open.

ϕ_{10} is not yet solvable since the unsolved node ψ_6, a successor of ϕ_1, has a higher priority wrt. $\sqsubset_L$. To express $(\psi_6, \phi_{10}) \in \sqsubset_L$, a $wait_1$-label has been *statically* assigned to ϕ_{10} before the traversal process. Upon reaching this $wait_1$-label the algorithm will compute the $\ll$-greatest open predecessor of ψ_6, that is ϕ_1. This node can now be marked as solved since $wait_2[\phi_1]$ not longer holds. No rule will be constructed since ϕ_1 is a special position which does not encode a sequent rule in $\mathcal{L}in_M$.

In the next step we reach the β-position β_1 and construct the rule $\otimes r$ which will cause the sequent proof to branch into two independent sub-proofs. In $\mathcal{MLL}$ the remaining resources (i.e. sequent formulas) will now have to be distributed over the new sub-branches. This additional process, called *context splitting*, yields $\emptyset$ for the left sub-branch and $\langle A \parr B, 1 \rangle$ for right one. In the algorithm, context splitting is realized by an operation $split(\propto^{\star}, \beta_1)$ which will divide the reduction ordering $\propto^{\star}$ into two sub-orderings $[\propto^{\star}_{1'}, \propto^{\star}_{2'}]$. In our example $\propto^{\star}_{1'}$ has one open position ψ_2 whereas $\propto^{\star}_{2'}$ contains two open position ψ_6 and ϕ_{10}.

Proof reconstruction will now continue separately on each sub-ordering. For $\propto^{\star}_{2'}$ we continue by solving ψ_6 and deleting the $wait_1$-label which blocks ϕ_{10} (called *update*). But now $wait_2[\phi_{10}]$ must be set dynamically since α_6 is open (layer reduction). Solving α_6 yields the rule $\parr r$ which prepares correct context splitting at the next β-position β_{10}. The reconstruction process will now continue as before and eventually yield the following $\mathcal{L}in_M$–proof.

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}\ axiom\ (a_3, a_5)}{\vdash A \parr A^{\perp}}\ \parr r,\ {}^{\perp}r\ (\alpha_2, o_4) \qquad \dfrac{\dfrac{\dfrac{}{A \vdash A}\ axiom\ (a_{11}, a_8) \quad \dfrac{}{B \vdash B}\ axiom\ (a_{12}, a_7)}{A \parr B \vdash B, A}\ \parr l\ (\beta_{10})}{A \parr B \vdash B \parr A}\ \parr r\ (\alpha_6)}{\dfrac{A \parr B \vdash (A \parr A^{\perp}) \otimes (B \parr A)}{\vdash ((A \parr A^{\perp}) \otimes (B \parr A)) \parr (A \parr B)^{\perp}}\ \parr r,\ {}^{\perp}r\ (\alpha_0, o_9)}\ \otimes r\ (\beta_1)$$

Adapting the Conversion Algorithm to $\mathcal{MLL}$. From the above example we develop the concepts for the conversion procedure. We omit formal details (see [24,25]) in order to emphasize properties which are specific to $\mathcal{MLL}$.

```
function TOTAL(∝*, MLL) : S-list =
 for all x ∈ positions(∝*) do solved[x] := ⊥;
 solved[root(∝*)] := ⊤;
 return TOT(∝*, MLL)

function TOT(∝*, MLL) : S-list =
 proven_∝* := ⊥; S_∝* := []; compute P_o;
 for all x ∈ positions(∝*) do compute wait1[x];
 while not proven_∝* do
  x := select_fair P_o;
  S_∝* := append(S_∝*, SOLVE(x, ∝*, MLL))
 return S_∝*

function SOLVE(x, ∝*, MLL) : S-list =
    ┌─────────────────────────────────────────────┐
    │ if wait1[x] then                            │
    │   select (y, x) ∈ ⊏_L; (k, r) := free(y, ∝*); │
    │   return SOLVE(succ_r(k), ∝*, MLL)          │
(1) │ else if wait2[x] then                       │
    │   y := select_fair {z ∈ P_o | x≠z};         │
    │   return SOLVE(y, ∝*, MLL)                  │
    └─────────────────────────────────────────────┘
 else
  solved[x] := ⊤;
  P_o := (P_o \ {x}) ∪ succ(x);
  if Ptype(x) = ψ then update(x, ∝*);
  case Ptype(x) of
   {φ, ψ} :  return []
   {atom} :  select {x, y} ∈ C
             if solved[y] then
              proven_∝* := ⊤;
              return [axiom(sform(x), sform(y))]
             else
              (k, r) = free(y, ∝*);
              return SOLVE(succ_r(k), ∝*, MLL)
   {o, α} :  return [Ptype(x)(sform(x))]
   {β} :    ┌──────────────────────────────────────┐
            │ [∝*_1', ∝*_2'] = split(∝*, x);       │
(2)         │ p0 = [β(sform(x), (∝*_1', ∝*_2'))];  │
            │ p1 = TOT(∝*_1', MLL);                │
            │ p2 = TOT(∝*_2', MLL);                │
            └──────────────────────────────────────┘
             proven_∝* := ⊤;
             return append(p0, append(p1, p2))

function update(x, ∝*) =
 for all {y | (x, y) ∈ ⊏_L} do
  ⊏_L := ⊏_L \ {(x, y)}; wait1[y] := ⊥
```

Definitions:

$open(x) \Leftrightarrow solved[pred(x)] \wedge \neg solved[x]$

$P_o = \{x \in positions(\propto^\star) \mid open(x)\}$

$wait_1[x] \Leftrightarrow \exists y.(y,x) \in \sqsubset_L$

$wait_2[x] \Leftrightarrow Ptype(x) \in \{\psi, \phi\} \wedge \exists y \in P_o . Ptype(y) \notin \{\psi, \phi\}$

$succ_j^+(x) := \{succ_j(x)\} \cup succ^+(succ_j(x))$

$free(x, \propto^\star)$ computes an ancestor/rank pair (k, r) with the $\ll$-greatest ancestor k such that $x \in succ_r^+(k)$ and $open(succ_r^+(k))$

select_fair denotes a fair selection strategy of open positions to guarantee termination wrt. the different *wait*-labels.

Fig. 5. The uniform transformation procedure adapted to $\mathcal{MLL}$.

The set of immediate successors/predecessors of a position x wrt. $\ll$ is denoted by $succ(x)$/$pred(x)$. $succ^+(x)$/$pred^+(x)$ denotes *all* successors/predecessors of x. If $succ(x)=\{x_1, x_2\}$ we assume a unique selection function $succ_j(x) = x_j$, $j \in \{1,2\}$. The definitions on the right hand side of Figure 5 summarize the necessary concepts which we introduced informally in Example 5.

For *context splitting* at β-positions x we have adopted the operation $split(\propto^\star, x)$ from [25]. It first divides $\propto^\star$ into two subrelations $\propto_1^\star, \propto_2^\star$ where each $\propto_i^\star$ contains the successor tree of x having root $x_i \in succ(x)$. The connections C_i and the relation $\sqsubset_{Li}$ are divided accordingly. Second, two non-normal form reductions will be applied to each $\propto_i^\star$ for correct context splitting and result in $\propto_{i'}^\star$. For $\mathcal{MLL}$ we can simplify these reductions to the following $\mathcal{MLL}$*-reduction*: Iteratively delete all subtrees with root y from $\propto_i^\star$ if $y \in P_o$ and there exists some $b \in succ^+(y)$ with $b \notin c$ for all $c \in C_i$. Using Lemma 3 and $wait_2$-labels we obtain.

Lemma 9. *$split(\propto^\star, x)$ is correct & complete for context splitting in $\mathcal{L}in_M$.*

The resulting algorithm $TOTAL$ adapted to $\mathcal{MLL}$ is depicted in Figure 5. The boxed area (1) focuses on integrating rule non-permutabilities into the conversion process. Box (2) summarizes the splitting process and recursive calls at β-positions. The procedure terminates with $proven_{\propto^*}$ and results a list of sequent rules $\mathcal{S}_{\propto^*}$ forming the sequent proof in $\mathcal{L}in_M$. No search is needed for conversion, i.e. the $\mathcal{L}in_M$ proof can be reconstructed in polynomial time in the size of the $\mathcal{MLL}$ matrix proof.

Theorem 10 (Completeness/Correctness). *The procedure TOTAL for converting $\mathcal{MLL}$ matrix proofs into $\mathcal{L}in_M$ sequent proofs is correct & complete.*

Proof sketch. Correctness follows from the correct construction of sequent rules at each position and correct context splitting (Lemma 9). For completeness the non-permutabilities of rule applications are captured by $wait_1$- and $wait_2$-labels where the latter fill the gap between $\mathcal{L}in_M$ and $\mathcal{K}^{\mathcal{MLL}}$ (Lemma 3), i.e.*layer* reductions. Finally, $wait_2$-labels do not cause deadlocks during traversal since by definition there always exists an open non-special position. $TOTAL$ terminates because the set of unsolved positions is decreased by each step.

5 Conclusion

We have presented a matrix characterization of logical validity in the multiplicative fragment of linear logic $\mathcal{MLL}$. On this basis we have extended our uniform proof search method [21] into a matrix-based proof procedure for $\mathcal{MLL}$ and our uniform transformation method [24] into a procedure for translating the resulting matrix proofs back into a sequent proof. Both methods could be adapted without modifications of the algorithmic structure. 'Only' the entries of logic-dependent tables which are consulted by the algorithms had to be elaborated.

Preliminary attempts for obtaining matrix characterizations in fragments of linear logic have been made on the basis of acyclic connection graphs [7,8]. This acyclicity condition is very close to proof nets and these attempts will therefore very like 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. Like Andreoli's focusing principle [1] and normal proofs [10] it avoids the permutabilities of sequent rules. Context splitting can be performed as efficiently as in resource management systems [5]. There is, however, 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 [26]). Prefix-Unification appears to be as efficient as the acyclicity test implicitly contained in [7] but yields informations which make the conversion into sequent proofs more efficient. Checking the *cardinality criterion* instead of an exponential minimality test is another improvement.

The most striking feature ouf our approach, however, is its generality and uniformity. It is emphasized by the fact that both the proof search procedure and the algorithm for conversion into sequent proofs, which originally had been developed for dealing with classical, intuitionistic, and modal logics, could so easily be adapted to $\mathcal{MLL}$, which semantically is entirely different. This makes us very confident that our method can be extended to further fragments of linear logic once a matrix characterization has been found. The similarities between sequent calculi for linear logics and those for the logics already characterized gives us additional evidence that extensions of our approach to larger fragments of linear logic will be possible. We believe that Wallen's conjecture (see introduction) will eventually turn out to be true for linear logic. Currently we are developing a matrix characterization for $\mathcal{MELL}$, the combination of $\mathcal{MLL}$ and exponentials and will investigate other fragments afterwards.

In the logics investigated so far the matrix characterization was always strongly related to the Kripke semantics of the logic. It may therefore become possible to follow this relation in the opposite direction and to construct a Kripke semantics for linear logic out of the matrix characterizations. We shall explore this possibility once a larger fragment of linear logic has been characterized.

References

1. J.-M. Andreoli. Logic programming with focussing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1993.
2. W. Bibel. On matrices with connections. *Journal of the ACM*, 28:633–645, 1981.
3. W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.
4. W. Bibel. *Automated theorem proving*. Vieweg, 1987.
5. I. Cervesato, J.S. Hodas, F. Pfenning Efficient resource management for linear logic proof search. *Extensions of Logic Programming*, LNAI 1050, pp. 67–81, 1996.
6. V. Danos, L. Regnier The structure of the multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.
7. B. Fronhöfer *The action-as-implication paradigm*, CS Press, 1996.
8. D. Galmiche Connection methods in linear logic fragments and proof nets construction. *CADE–13 workshop on proof search in type-theoretic languages*, 1996.
9. D. Galmiche, G. Perrier. A procedure for automatic proof nets construction. *LPAR'92*, LNAI 624, pp. 42–53, Springer Verlag, 1992.
10. D. Galmiche, G. Perrier. On proof normalization in linear logic. *TCS*, 135:67–110, 1994.
11. V. Gehlot, C. Gunter. Normal process representatives. In *Proc. 5-th Annual IEEE Symposium on Logic in Computer Science*, pp. 200–207, 1991.
12. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
13. J.-Y. Girard. Proof-nets: the parallel syntax for proof-theory. In *Logic and Algebra*, LNPAM 150, pp. 97–124, 1996.
14. J.S. Hodas, D. Miller. Logic programming in a fragment of linear logic. *Journal of Information and Computation*, 110(2):327–365, 1994.
15. H. Mantel. Eine Matrixcharakterisierung für ein Fragment der linearen Logik. *Diplomarbeit*, TH-Darmstadt, Germany, 1996.
16. A. Martelli, U. Montanari. An efficient unification algorithm. *ACM TOPLAS*, 4:258–282, 1982.
17. M. Masseron, C. Tollu, J. Vauzielles Generating plans in linar logic. In *Foundations of Software Technology and Theoretical Computer Science*, LNCS 472, pp. 63–75, Springer, 1990.
18. J. McCarthy, P.H. Hayes Some philosophical problems from the standpoint of Artificial Intelligence *Machine Intelligence*,4:463–502, 1969.
19. J. Otten, C. Kreitz. A connection based proof method for intuitionistic logic. *4^{th} TABLEAUX Workshop*, LNAI 918, pp. 122–137, Springer Verlag, 1995.
20. J. Otten, C. Kreitz. T-string-unification: unifying prefixes in non-classical proof methods. *5^{th} TABLEAUX Workshop*, LNAI 1071, pp. 244–260, Springer Verlag, 1996.
21. J. Otten, C. Kreitz. A uniform proof procedure for classical and non-classical logics. *KI-96: Advances in Artificial Intelligence*, LNAI 1137, pp. 307–319, Springer Verlag, 1996.
22. J. Otten. ileanTAP: An intuitionistic theorem prover. *International Conference TABLEAUX'97*, LNAI, Springer Verlag, 1997.
23. S. Schmitt, C. Kreitz. On transforming intuitionistic matrix proofs into standard-sequent proofs. *4^{th} TABLEAUX Workshop*, LNAI 918, pp. 106–121, Springer Verlag, 1995.
24. S. Schmitt, C. Kreitz. Converting non-classical matrix proofs into sequent-style systems. *CADE–13*, LNAI 1104, pp. 418–432, Springer Verlag, 1996.
25. S. Schmitt, C. Kreitz. A uniform procedure for converting non-classical matrix proofs into sequent-style systems. *Journal of Information and Computation*, submitted.
26. T. Tammet. Proof strategies in linear logic. *Jour. of Automated Reasoning*, 12:273–304, 1994.
27. L. Wallen. *Automated deduction in non-classical logics*. MIT Press, 1990.