Based-on dependency Calculi for Non-commutative Logic

D. Galmiche, J.M. Notin

LORIA–Université Henri Poincaré
Campus Scientifique, BP 239
Vandœuvre-lès-Nancy, France

Abstract

In this paper we propose new calculi for the multiplicative fragment of Non-commutative Logic (MNL) which is a linear logic that combines both commutative and non-commutative connectives. Both sequent and proof net calculi, that are new proof-theoretical formulations of MNL, are based on dependency relations. After proving their main properties and also adequacy and sequentialization theorems between both calculi, we provide an algorithm for MNL proof nets construction that is correct and complete. Moreover the related correctness criterion for MNL proof nets is shown quadratic in time. The use of dependency graphs for deciding provability appears as a powerful tool for efficient proof search in such mixed logics.

1 Introduction

Many works have been recently devoted to the study of linear logic (LL) [15] because its ability, as a logic of resources, to be easily and strongly related to different topics in computer science, like process calculi or concurrent constraint programming [4]. In many cases, it appears that commutative linear logic is not enough expressive for specifying processes with both asynchronous and synchronous behaviors and thus a possible solution is the use of non-commutative logic [2] or non-commutative connectives to deal with sequentiality. Recently, a mixed version of classical propositional linear logic (NL), which combines both commutative and non-commutative connectives, has been proposed [1], for defining logical characterizations of synchronization mechanisms in concurrent constraint programming [22]. Moreover, the intuitionistic fragment of NL has been recently used to specify and verify properties about the TCP/IP protocol [17]. In order to study the execution of programs (agents) or to prove systems properties specified in such a mixed logic, we aim to define appropriate proof systems dedicated to efficient proof-search. A first

step is to consider the multiplicative fragment of Non-commutative linear logic (MNL) [1] that extends the commutative multiplicative linear logic (MLL) and the cyclic multiplicative linear logic (McyLL). We have previously designed proof-search methods, based on proof nets [16], for these fragments [8,9], but they cannot be directly extended to MNL because of the difficulty to capture interactions between commutative and non-commutative connectives. A first solution for MNL consists in using labelled (with constraints) proof nets, the labels allowing to define constraints to be satisfied or to propagate some information useful to guide and control the search [11]. But the labels are not naturally related to the semantics of MNL and, for instance, the failure in proof search cannot lead to the generation of countermodels.

In this paper, we define, for MNL, a new sequent calculus (DepSC) and a new proof net calculus (DepPN), that are based on dependency relations and can be seen as a new formulation of this logic. The idea of dependency relations was previously used in [7], in order to define provability in Full Intuitionistic Linear Logic (FILL) from provability in Classical Linear logic (CLL). In this work, we use dependency relations in order to relate provability in MNL with provability in MLL with, in addition, the verification of a condition on a so-called dependency graph. These structures well capture the interactions between connectives and allow to translate proof-theoretical results of MNL into simple and tractable automated proof-search procedures, that are based on semantical structures like proof nets [8,16]. We also study the relationships between both proof systems through the adequacy and sequentialization theorems. The latter naturally provides sequentialization for proof nets with cuts, property that is difficult to obtain with the initial MNL proof nets theory [20,1]. Moreover, we prove the cut-elimination property for the dependency sequent calculus. From our based-on dependency formulations and also from previous works in MLL [8], we propose an algorithm for MNL proof nets construction, that builds in parallel a MNL proof structure and a set (or graph) of dependencies on which some validity conditions have to be verified. It is naturally proved correct and complete and we observe that the dependency graph is a semantic structure from which we could generate countermodels and analyze the proof search failure. To complete the study, we adapt this algorithm to the verification of MNL proof structures and show that the related correctness criterion for MNL proof nets is quadratic in time like the one based on contractibility proposed in [20]. We expect to obtain, in further work, a better complexity from a deeper analysis of the dependency sets. Finally, we will apply these proof-theoretical and algorithmic results to the verification of properties of programs or protocols expressed in MNL and also focus on the use of dependency graphs for countermodels generation.

[1] This logic is called here "Non-commutative linear logic" like in [1], but was called before "Mixed Linear Logic".

2 Non-Commutative (Linear) Logic

In this section we consider the multiplicative fragment of non-commutative logic (MNL), that is a conservative extension of both commutative (MLL) and non-commutative or cyclic (McyLL) linear logic [22]. This logic is an alternative to commutative classical, intuitionistic or linear logics for applications in computer science and has been used for a fine logical characterization of operational aspects of concurrent constraint programming (CC) [22]. Informally, a program P written in CC can be translated to a formula F in a specific logic, so that properties of this program can be expressed through the validity of F. Thus, encoding a CC program P in a formula F of commutative linear logic allows to state whether or not the program can lead to a success (i.e., can be successfully executed). But, due to logical interaction between the linear implication ($\multimap$) and multiplicative conjunction ($\otimes$), this translation does not allow to state whether or not the program P can lead to a particular deadlock situation. A solution is to use a non-commutative implication ($\bullet\!\!-$) together with the commutative conjunction ($\otimes$). Recently, the intuitionistic fragment of NL has been used to specify and verify properties of the TCP/IP protocol [17]. Though the IP protocol can be specified and verified in commutative LL, the specification and verification of the TCP protocol request to have a linear implication that consumes its premise in a particular order, together with a commutative conjunction. In addition to this ability to represent mechanisms or properties dealing with sequentiality and concurrency, this logic can also offer new perspectives to the connection between concurrency and proof theory, with the paradigm of logic programming (proof-search as computation). Therefore, one could propose new definitions or extensions of process calculi [5,19] where sequentiality and concurrency can cohabit, following such a correspondence with Non-commutative Linear Logic. It suggests new investigations of the semantics of concurrency from proof theory.

2.1 A sequent calculus with explicit structural rules

The formulae (of MNL) are build from literals $p, q, \ldots, p^\perp, q^\perp, \ldots$ with non-commutative connectives (conjunction $\odot$ and disjunction $\lhd$), commutative connectives (conjunction $\otimes$ and disjunction $\parr$). The negation is defined as follows with the De Morgan laws: $(p)^\perp = p^\perp$, $(p^\perp)^\perp = p$, $(A \odot B)^\perp = B^\perp \lhd A^\perp$, $(A \lhd B)^\perp = B^\perp \odot A^\perp$, $(A \otimes B)^\perp = B^\perp \parr A^\perp$, $(A \parr B)^\perp = B^\perp \otimes A^\perp$. The formulae of MLL (resp. McyLL) are build from the atoms $p, q, \ldots, p^\perp, q^\perp, \ldots$ and the connectives $\otimes$, $\parr$, (resp. $\odot$ and $\lhd$).
There exists a sequent calculus with explicit structural rules for MNL [1]. Two separators of formulae are used in a sequent: a commutative one ',' and a non-commutative one ';'. In this MNL sequent calculus (see figure 1), some

$$\frac{}{\vdash A^{\perp}, A}\ ax \qquad \frac{\vdash \Gamma, A \quad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}\ cut \qquad \frac{\vdash \Gamma[\Delta; \Sigma]}{\Gamma[\Delta, \Sigma]}\ entr$$

$$\frac{\vdash \Gamma, \Delta}{\vdash \Gamma; \Delta}\ seesaw \qquad \frac{\vdash \Gamma; A \qquad \vdash B; \Delta}{\vdash \Gamma; A \odot B; \Delta}\ \odot \qquad \frac{\vdash \Gamma; A; B}{\vdash \Gamma; A \lhd B}\ \lhd$$

$$\frac{\vdash \Gamma, A \qquad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta}\ \otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ \parr$$

Fig. 1. A sequent calculus for MNL

explicit structural rules (*seesaw* and *entropy*) define the relationships between these separators. Moreover they are used in conjunction with the rule for the commutativity of ',' in order to recover the cyclicity of ';'. The notation $\Gamma[\ ]$ represents a context with a hole in its decomposition tree and $\Gamma[\Delta]$ is the context $\Gamma[\ ]$ where the hole is filled by Δ. The proof-theoretic and semantic results about MNL (cut-elimination, sub-formula property, phase semantics, completeness theorem) are given in [1,21]. If we aim to analyze the proof search with this calculus, some problems arise: some orders are fixed by non-commutative connectives and they interact with commutative connectives in the same framework. Let us note that an alternative to study the proof theory of MNL is to consider the notion of proof net [16].

2.2 A sequent calculus with implicit structural rules

Ruet has introduced a sequent calculus which used a specific mathematical structure, namely order variety, to deal with the interactions between connectives [1]. Order varieties can be seen as refinements of cyclic orders and are structures which, provided a point of view (an element x in the base set), can be seen as partial orders on the complement of $\{x\}$ [1,21].

Definition 1 (order variety) *Let E be a set, an* order variety *on E is a ternary relation α which is :*
- cyclic : $\forall x, y, z \in E\ \alpha(x, y, z) \Rightarrow \alpha(y, z, x)$
- anti-reflexive : $\forall x, y \in E\ \neg\alpha(x, x, y)$
- transitive : $\forall x, y, z, t \in E\ \alpha(x, y, z) \wedge \alpha(z, t, x) \Rightarrow \alpha(y, z, t)$
- spreading : $\forall x, y, z, t \in E\ \alpha(x, y, z) \Rightarrow \alpha(t, y, z) \vee \alpha(x, t, z) \vee \alpha(x, y, t)$.
An order variety is said total *when :*
$\forall x, y, z \in E\ x \neq y \neq z \Rightarrow \alpha(x, y, z) \vee \alpha(z, y, x)$

Any order variety α on a E induces an oriented graph $\rightarrow_\alpha$ on E with an edge between x and $y \in E$ iff : $\forall z \in E\ z \neq x \wedge z \neq y \Rightarrow \alpha(x, y, z)$. If α is a total order variety, then $\rightarrow_\alpha$ is an oriented cycle. Let α a variety on E and $x \in E$, the binary relation α_x on $E \setminus \{x\}$ is defined by $\alpha_x(y, z)$ iff $\alpha(x, y, z)$.

$$\frac{}{\vdash A^{\perp}, A\ \langle\emptyset\rangle}\ (\mathrm{ax}) \qquad \frac{\vdash \Gamma, A\ \langle\alpha\rangle \quad \vdash A^{\perp}, \Delta\ \langle\beta\rangle}{\vdash \Gamma, \Delta\ \langle\alpha \asymp_{A,A^{\perp}} \beta\rangle}\ (\mathrm{cut})$$

$$\frac{\vdash \Gamma, A\ \langle\alpha\rangle \quad \vdash A^{\perp}, \Delta\ \langle\beta\rangle}{\vdash \Gamma, A \odot B, \Delta\ \langle\alpha \odot_{A,B} \beta\rangle}\ (\odot) \qquad \frac{\vdash \Gamma, A, B\ \langle\alpha\rangle}{\vdash \Gamma, A \lhd B\ \langle\alpha[A \lhd B/A, B]\rangle}\ (\lhd)\ \text{if } A \to_{\alpha} B$$

$$\frac{\vdash \Gamma, A\ \langle\alpha\rangle \quad \vdash A^{\perp}, \Delta\ \langle\beta\rangle}{\vdash \Gamma, A \otimes B, \Delta\ \langle\alpha \otimes_{A,B} \beta\rangle}\ (\otimes) \qquad \frac{\vdash \Gamma, A, B\ \langle\alpha\rangle}{\vdash \Gamma, A \parr B\ \langle\alpha[A \parr B/A, B]\rangle}\ (\parr)$$

Fig. 2. A sequent calculus with order varieties

Definition 2 *Let (E, ω) be a strict order and $z \in E$, the binary relation $\overset{z}{<}_{\omega}$ is defined by : $x \overset{z}{<}_{\omega} y$ iff $x <_{\omega} y$ and z is comparable with neither x nor y. The ternary relation $\overline{\omega}$ on E is defined by:*

$$\overline{\omega}(x, y, z)\ iff\ \begin{cases} x <_{\omega} y <_{\omega} z\ \vee\ z <_{\omega} x <_{\omega} y\ \vee\ y <_{\omega} z <_{\omega} x\ \vee \\ x \overset{z}{<}_{\omega} y\ \vee\ y \overset{x}{<}_{\omega} z\ \vee\ z \overset{y}{<}_{\omega} x \end{cases}$$

Definition 3 *Let ω_1 and ω_2 be orders on disjoints sets X and Y, the parallel $(\omega_1 \parallel \omega_2)$ and serial $(\omega_1 \ll \omega_2)$ compositions, that are both orders on $X \cup Y$, are defined by :*
- $(\omega_1 \parallel \omega_2)(x, y)$ iff $x <_{\omega_1} y$ or $x <_{\omega_2} y$,
- $(\omega_1 \ll \omega_2)(x, y)$ iff $x <_{\omega_1} y$ or $x <_{\omega_2} y$ or $(x \in X$ and $y \in Y)$.

Definition 4 (interior) *Let α be an order variety on E, its interior is defined by: $\natural\alpha = \bigcap_{x \in E} \overline{\alpha_x \parallel x}$*

Definition 5 (Sequents) *A sequent of MNL $\vdash \Gamma\ \langle\alpha\rangle$ is a multiset Γ of occurrences of formulae, together with an order variety α on Γ.*

The rules of the sequent calculus are given in figure 2. Informally, the inference rules are those of the sequent calculus for MLL, seeing $\lhd$ as a $\parr$, and $\odot$ as a $\otimes$, but, in order to recover the non-commutative behavior of $\lhd$ and $\odot$, order varieties are associated to sequents. The operations on order varieties are defined as follows: let α and β be two order varieties respectively defined on $\Gamma \cup \{A\}$ and $\Delta \cup \{B\}$, $\alpha \odot_{A,B} \beta$ is the order variety on $\Gamma \cup \Delta \cup \{A \odot B\}$ defined by $\alpha \odot_{A,B} \beta = \overline{\alpha_A \ll A \odot B \ll \beta_B}$. Moreover $\alpha \otimes_{A,B} \beta$ is the order variety on $\Gamma \cup \Delta \cup \{A \otimes B\}$ defined by $\alpha \otimes_{A,B} \beta = \overline{\alpha_A \parallel A \otimes B \parallel \beta_B}$. $\alpha \asymp_{A,B} \beta$ is the order variety on $\Gamma \cup \Delta$ defined by $\alpha \asymp_{A,B} \beta = \overline{\alpha_A \parallel \beta_B}$. In addition, let α be an order variety on a set $E \cup \{x, y\}$ with $x, y \notin E$, and let $z \in E$, we define the pasting $\alpha[z/x, y]$ of x and y along z in α by : $\alpha[z/x, y] = \natural\left(\alpha \restriction_{E \cup \{x\}} [z/x] \bigcap \alpha \restriction_{E \cup \{y\}} [z/y]\right)$. It is important to notice that the $\lhd$ rule can only be applied under the condition $A \to_{\alpha} B$. This point is essential to understand the next results. Let us note that cut-elimination has been proved in this sequent calculus [1]. We give a derivation of the sequent $\vdash (A \odot B) \lhd C, A^{\perp}, C^{\perp} \odot B^{\perp}$.

$$\dfrac{\dfrac{\dfrac{}{\vdash C^{\perp}, C\ \langle\emptyset\rangle}(ax) \quad \dfrac{\dfrac{}{\vdash A^{\perp}, A\ \langle\emptyset\rangle}(ax) \quad \dfrac{}{\vdash B^{\perp}, B\ \langle\emptyset\rangle}(ax)}{\vdash A^{\perp}, A \odot B, B^{\perp}\ \langle \overline{A^{\perp} < A \odot B < B^{\perp}} \rangle}(\odot)}{\vdash C, C^{\perp} \odot B^{\perp}, A^{\perp}, A \odot B\ \langle \overline{C < C^{\perp} \odot B^{\perp} < A^{\perp} < A \odot B} \rangle}(\odot)}{\vdash (A \odot B) \lhd C, A^{\perp}, C^{\perp} \odot B^{\perp}\ \langle \overline{C^{\perp} \odot B^{\perp} < A^{\perp} < (A \odot B) \lhd C} \rangle}(\lhd)$$

It is a proof of the given sequent because the condition to apply to the $\lhd$ rule, namely $A \otimes B \rightarrow_{\alpha} C$, is satisfied. If we replace $(A \odot B) \lhd C$ by $(A \otimes B) \lhd C$ the corresponding condition is not satisfied and thus the corresponding derivation is not a proof. Moreover, the verification of this condition cannot be trivially included in the process of proof search within this calculus. One main interest of a new calculus with dependencies is to express it inside the calculus.

2.3 Proof search in MNL

The calculus with explicit structural rules is clearly not appropriate for automated proof search because of the difficult management of the structural rules during the proof search process. In the opposite, the calculus with implicit structural rules is strongly related to the MLL sequent calculus, for which we have results on proof normalization [13] and proof-search [8]. But the order varieties, associated to sequents, are mathematical structures which are difficult to deal with during proof-search. A first proposal to study proof-search in MNL has been based on labelled proof nets [11]. The labels allow to fix some constraints to satisfy during the proof net search and also to propagate some informations useful to guide and control the search. Moreover, such based-on proof nets methods are useful in order to efficiently detect the non-provability.
With new formulations of MNL based on dependency relations, we aim to have a more natural way to deal, from so-called dependency graphs, with the interactions between commutative and non-commutative connectives in the same logic. Thus, we propose proof-search methods, based-on proof nets [8,9], that could generate countermodels in case of non validity in MNL. Comparisons with our previous proposals [11] and implementation techniques will be studied in further work, from both algorithmic and complexity points of view.

3 A dependency sequent calculus

We give a new sequent calculus, which is a reformulation of MNL that appears appropriate for characterizing provability in MNL. This calculus is based on the notion of *dependency* in order to capture the interactions between com-

mutative and non-commutative connectives.

Definition 6 *A* dependency relation *between two formulae A and B is represented by $A \to B$. A* dependency sequent $\vdash \Gamma\ \mathcal{D}$ *is a multiset Γ of formulae, associated with a set of dependencies $\mathcal{D}$.*

Let $\vdash \Gamma\ \mathcal{D}$ be a dependency sequent, the set $\mathcal{D}$ is a graph whose nodes are the Γ formulae and whose edges are the dependency relations between these formulae. We observe that $\mathcal{D}$ is cyclic by construction.

Definition 7 *Let $\vdash \Gamma\ \mathcal{D}$ a dependency sequent, a cycle in $\mathcal{D}$ is said* maximal *if it contains exactly once all the formulae of Γ. The set of maximal cycles in $\mathcal{D}$ is denoted $\mathcal{D}_C$.*

Let γ be a subset of Γ, $\mathcal{D}\restriction_\gamma$ denotes the restriction of $\mathcal{D}$ to the subset γ of formulae ($\mathcal{D}\restriction_\gamma = \{E \to F | E, F \in \gamma, E \to F \in \mathcal{D}\}$). Moreover, $\mathcal{D}[Y/X]$ represents the set $\mathcal{D}$ where the formula (occurrences) X is replaced by Y.

Definition 8 *The dependency sequent calculus for MNL (denoted DepSC) is defined as follows with dependency sequents:*

$$\frac{}{\vdash A^\perp, A\ \mathcal{D}}\ (ax) \qquad \textit{with}\ \mathcal{D} = \left\{ \begin{array}{c} A \to A^\perp \\ A^\perp \to A \end{array} \right\}$$

$$\frac{\vdash \Gamma, A\ \mathcal{D}_1 \quad \vdash A^\perp, \Delta\ \mathcal{D}_2}{\vdash \Gamma, \Delta\ \mathcal{D}}\ (cut)$$

$$\begin{array}{rcl} \mathcal{D} & = & \mathcal{D}_1\restriction_\Gamma \cup \mathcal{D}_2\restriction_\Delta \cup \\ & & \{E \to F | E \in \Gamma, F \in \Delta, E \to A \in \mathcal{D}_1, \\ \textit{with} & & A^\perp \to F \in \mathcal{D}_2\} \cup \\ & & \{E \to F | E \in \Delta, F \in \Gamma, E \to A^\perp \in \mathcal{D}_2, \\ & & A \to F \in \mathcal{D}_1\} \end{array}$$

$$\frac{\vdash \Gamma, A\ \mathcal{D}_1 \quad \vdash B, \Delta\ \mathcal{D}_2}{\vdash \Gamma, A \odot B, \Delta\ \mathcal{D}}\ (\odot)$$

$$\begin{array}{rcl} \mathcal{D} & = & \mathcal{D}_1\restriction_\Gamma \cup \mathcal{D}_2\restriction_\Delta \cup \\ & & \{E \to F | E \in \Gamma, F \in \Delta, E \to A \in \mathcal{D}_1, \\ & & B \to F \in \mathcal{D}_2\} \cup \\ \textit{with} & & \{A \odot B \to E | E \in \Gamma, A \to E \in \mathcal{D}_1\} \cup \\ & & \{F \to A \odot B | F \in \Delta, F \to B \in \mathcal{D}_2\} \cup \\ & & \{E \to A \odot B | E \in \Gamma, E \to A \in \mathcal{D}_1, \Delta = \emptyset\} \cup \\ & & \{A \odot B \to F | F \in \Delta, B \to F \in \mathcal{D}_2, \Gamma = \emptyset\} \end{array}$$

$$\frac{\vdash \Gamma, A, B\ \mathcal{D}_1}{\vdash \Gamma, A \lhd B\ \mathcal{D}}\ (\lhd) \qquad \textit{with } B \to A \textit{ is in all the maximal cycles in } \mathcal{D}_1\ (\lhd^*)$$

$$\textit{and}\ \mathcal{D} = \mathcal{D}_1[A \lhd B/A, A \lhd B/B]$$

$$\frac{\vdash \Gamma, A\ \mathcal{D}_1 \quad \vdash B, \Delta\ \mathcal{D}_2}{\vdash \Gamma, A \otimes B, \Delta\ \mathcal{D}}\ (\otimes)$$

$$\begin{array}{rcl} \mathcal{D} & = & \mathcal{D}_1\restriction_\Gamma \cup \mathcal{D}_2\restriction_\Delta \cup \\ & & \{E \to F | E \in \Gamma, F \in \Delta, E \to A \in \mathcal{D}_1, \\ & & B \to F \in \mathcal{D}_2\} \cup \\ & & \{A \otimes B \to E | E \in \Gamma, A \to E \in \mathcal{D}_1\} \cup \\ \textit{with} & & \{F \to A \otimes B | F \in \Delta, F \to B \in \mathcal{D}_2\} \cup \\ & & \{F \to E | E \in \Gamma, F \in \Delta, A \to E \in \mathcal{D}_1, \\ & & F \to B \in \mathcal{D}_2\} \cup \\ & & \{A \otimes B \to F | F \in \Delta, B \to F \in \mathcal{D}_2\} \cup \\ & & \{E \to A \otimes B | E \in \Gamma, E \to A \in \mathcal{D}_1\} \end{array}$$

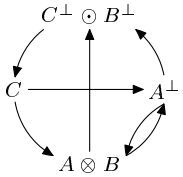
$$\frac{\vdash \Gamma, A, B \ \mathcal{D}_1}{\vdash \Gamma, A \parr B \ \mathcal{D}} \ (\parr) \qquad \text{with} \quad \begin{array}{l} \mathcal{D} = \mathcal{D}_1[A\parr B/A, A\parr B/B] \cup \\ \quad \{E \to F \,|\, E, F \in \Gamma,\ E \to A, A \to F \in \mathcal{D}_1\} \cup \\ \quad \{E \to F \,|\, E, F \in \Gamma,\ E \to B, B \to F \in \mathcal{D}_1\} \end{array}$$

In fact, we observe that the sequent rules are the same for $\otimes$ and $\odot$ and also for $\parr$ and $\lhd$ and that the distinction between commutative and non-commutative connectives is given in the definition of dependency sets. Considering the rule for $\otimes$, we can see the dependencies are extended as for $A \odot B$, and also as for $B \odot A$. This difference in the construction of the dependencies from those associated to the premise allows, in some way, to express the commutativity of $\otimes$.

A central point is that the $\lhd$ rule of DepSC can only be applied if a specific condition $(\lhd^*)$ is satisfied. The application of the $\lhd$ rule, and then the construction of $\mathcal{D}$ from $\mathcal{D}_1$, is possible only if the condition $(\lhd^*)$ $B \to A$ *is in all the maximal cycles in* $\mathcal{D}_1$, is satisfied. Before to present the properties of this calculus, we give an example of derivation with the sequent $\vdash (A \otimes B) \lhd C, A^\perp, C^\perp \odot B^\perp$. A proof is the classical sequent tree where a set of dependencies is associated to each sequent, following the rules of definition 8. In case of a $\lhd$ rule application, we have to verify the $(\lhd^*)$ condition.

$$\dfrac{\dfrac{\dfrac{}{\vdash C^\perp, C \ \mathcal{D}_3}\,(Ax) \qquad \dfrac{\dfrac{}{\vdash A^\perp, A \ \mathcal{D}_1}\,(Ax) \quad \dfrac{}{\vdash B^\perp, B \ \mathcal{D}_2}\,(Ax)}{\vdash A^\perp, A \otimes B, B^\perp \ \mathcal{D}_4}\,(\otimes)}{\vdash C, C^\perp \odot B^\perp, A^\perp, A \otimes B \ \mathcal{D}_5}\,(\odot)}{\vdash (A \otimes B) \lhd C, A^\perp, C^\perp \odot B^\perp \ \mathcal{D}_6}\,(\lhd)$$

where $\mathcal{D}_5 = \left\{ \begin{array}{ll} C \to A \otimes B & C \to A^\perp \\ A \otimes B \to A^\perp & A^\perp \to A \otimes B \\ A^\perp \to C^\perp \odot B^\perp & A \otimes B \to C^\perp \odot B^\perp \\ C^\perp \odot B^\perp \to C & \end{array} \right\}$ or



There are two maximal cycles in $\mathcal{D}_5$, namely $C \to A \otimes B \to A^\perp \to C^\perp \odot B^\perp \to C$ and $C \to A^\perp \to A \otimes B \to C^\perp \odot B^\perp \to C$. In order to satisfy the condition $(\lhd^*)$ we have to verify if $C \to A \otimes B$ is in all maximal cycles in $\mathcal{D}_5$. It is not the case for the second maximal cycle and then we cannot deduce $(A \otimes B) \lhd C$ and therefore this derivation is not a proof. If we replace $(A \otimes B)$ by $(A \odot B)$ in the previous example, we have a similar proof tree but with another set $\mathcal{D}_5$. In this case, the only maximal cycle in $\mathcal{D}_5$ is $C \to A \odot B \to A^\perp \to C^\perp \odot B^\perp \to C$ and the condition $(\lhd^*)$ is satisfied. Thus, the given sequent $\vdash (A \odot B) \lhd C, A^\perp, C^\perp \odot B^\perp$ is provable.

The main point for the proof construction within this calculus is the construction of the dependency sets and the verification of $(\lhd^*)$ condition for the $\lhd$ rule.

Theorem 9 *Let Π be a proof in the DepSC dependency calculus, with conclusion $\vdash \Gamma\ \mathcal{D}$. Then, there exists a maximal cycle in $\mathcal{D}$.*

PROOF. By induction on the structure of Π. Informally, the rules for the construction of dependencies are such that, when a $\mathcal{D}$ is computed from $\mathcal{D}_1$ and possibly $\mathcal{D}_2$, the dependencies concerning A or B are copied into $\mathcal{D}$ and those concerning A or B are properly prolongated. A detailed proof is given in appendix. $\square$

Theorem 10 *The DepSC dependency calculus is equivalent to the MNL sequent calculus with order varieties.*

PROOF. Let us first point out that we erase the semantical structure associated to sequents (respectively order varieties or dependency sets), we are in the MLL calculus, by considering the connectives $\odot$ and $\lhd$ as, respectively, $\otimes$ and $\parr$. Thus, in order to prove the theorem, we only have to give an equivalence relation between order varieties and dependency sets. Let $\mathcal{D}$ be a dependency set, and let us consider the relation $\alpha_{\mathcal{D}}$ defined on Γ by: $\forall x, y, z \in \Gamma\ \alpha_{\mathcal{D}}(x, y, z)$ *iff* $x \neq y \neq z$ *and* z, y, x *appears in this cyclic order in all the maximal cycles in* $\mathcal{D}$. Then, given a proof in the sequent calculus with order variety with conclusion $\vdash \Gamma\ \langle\alpha\rangle$, then we can build a proof in DepSC with conclusion $\vdash \Gamma\ \mathcal{D}$ such that $\alpha = \natural\alpha_{\mathcal{D}}$. Conversely, given a proof in DepSC with conclusion $\vdash \Gamma\ \mathcal{D}$, we can build a proof in the sequent calculus with order varieties with conclusion $\vdash \Gamma\ \langle\alpha_{\mathcal{D}}\rangle$. This can be directly shown by induction, using the results on order varieties [1]. The proof is not detailed here, but it will follow from the results on the use of dependencies for proof nets construction. Let us just point out that the condition $B \rightarrow A$ *is in all the maximal cycles in* $\mathcal{D}$ is equivalent to $A \rightarrow_{\alpha_{\mathcal{D}}} B$. $\square$

Let us analyze the $(\lhd^*)$ condition of the $\lhd$ rule of the DepSC calculus. The complexity of the proof-search in DepSC depends on the complexity of the satisfiability of this condition and thus we propose an equivalent condition but more appropriate for efficient verification.

Theorem 11 *Let Π be a proof in the DepSC dependency calculus, with conclusion $\vdash \Gamma\ \mathcal{D}$, if $E \rightarrow F \in \mathcal{D}$ then there exists a maximal cycle $\mathcal{C}$ in $\mathcal{D}$ such that $E \rightarrow F \in \mathcal{C}$*

PROOF. From the construction of dependency sets, by induction on the structure of Π. When we build a new dependency set $\mathcal{D}$ from $\mathcal{D}_1$ and $\mathcal{D}_2$, we keep all the dependencies that does not concern A or B. The ones concerning A or B are prolongated in such a way that we always can build a maximal cycle containing them from maximal cycles in $\mathcal{D}_1$ and $\mathcal{D}_2$. $\square$

From this result we can deduce a new equivalent condition for the $\lhd$ rule.

Theorem 12 *Let $\vdash \Gamma, A, B\ \mathcal{D}_1$ a dependency sequent, the two following conditions are equivalent*
($\lhd^*$) $B \to A$ is in all the maximal cycles in $\mathcal{D}_1$.
($\lhd^{**}$) $\{F \in \Gamma\ s.t.\ B \to F \in \mathcal{D}_1\} = \{A\}$.

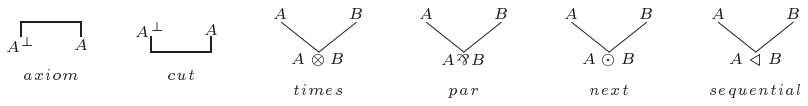
PROOF. If condition ($\lhd^{**}$) is satisfied, A is the only formula for which we have $B \to A \in \mathcal{D}_1$ and by definition of a maximal cycle, every maximal cycle in $\mathcal{D}_1$ contains $B \to A$ and then condition ($\lhd^*$) is satisfied. If condition ($\lhd^*$) is satisfied, $B \to A$ is in all maximal cycles in $\mathcal{D}_1$. Let us suppose that there exists X (not equal to A) such that $B \to X \in \mathcal{D}_1$. By theorem 11 there exists a maximal cycle $\mathcal{C}$ in $\mathcal{D}_1$ such that $B \to X \in \mathcal{C}$. By definition of a maximal cycle, that is contradictory. □

Thus, we can now consider the DepSC calculus by replacing the condition ($\lhd^*$) of the $\lhd$ rule by the condition ($\lhd^{**}$), that will improve the complexity of the proof search in MNL.

4 Proof nets of MNL

The notion of *proof net* has been introduced by Girard [14,16] in order to cope with the problems due to the intrinsic concurrence of the linear sequent calculus. It corresponds to a concise graph-theoretic representation of linear logic deductions. This concept, initially defined for MLL, has been defined for various LL fragments like MCyLL [3] or MALL [16]. We observe that, taking into account the sequentialization problem, it is difficult to find the most natural and adequate definition of a proof net in a given logic.
In this section we recall the main results about MNL proof nets, mainly the relationships between proof nets and proofs with order varieties [1].

Definition 13 *The* links *of MNL are the following graphs, the vertices of which are labelled with MNL formulae:*

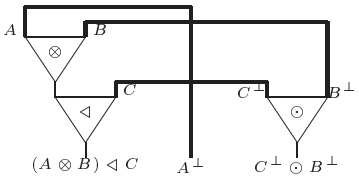


Axiom-links have two conclusions and no premise. Cut links have two premisses and no conclusion. In the other cases, A and B are respectively the first and second premise and the third formula is the conclusion.

Definition 14 *A* proof structure *(of MNL) is a graph built from links (of MNL) such that every occurrence of formula is either the conclusion of exactly*

one link or the premise of at most one link. An occurrence that is not a premise of a link is called terminal and it is a conclusion of the structure.

A proof structure of MLL (resp. McyLL) is a proof structure labelled with only MLL (resp. McyLL) formulae. We can also define a proof structure from a pair $(Dectree, Axlinks)$ where, for a given sequent, $Dectree$ is the set of its subformulae arranged as the decomposition tree and $Axlinks$ is a set of axiom-links (or connections) between positive and negative occurrences of the same atomic formula. Such a pair is a proof structure if each atom of $Dectree$ is the conclusion of exactly one axiom-link. For instance, for the MNL sequent $\vdash (A \otimes B) \lhd C, A^{\perp}, C^{\perp} \odot B^{\perp}$ we have the following proof structure (associated to it).



There are different correctness criteria that decides if a proof structure corresponds to a (graphical) proof of a given sequent or not, i.e., if a proof structure is a proof net. It is not trivial to define criteria that are effectively useful, for instance to analyze non-provability and also with a good complexity [15,20]. We will see that the above proof structure is not a proof net.
Since several years, in opposition to the problem of verifying proof structures, we have focus on the use of proof nets for proof-search and then we have proposed algorithms for automatic proof nets construction in various LL fragments [12,9]. Moreover, we have analyzes their relationships with so-called connection methods in linear logic fragments [8]. Here, we propose a new formulation of MNL proof nets and consequently a new criterion for proof structure verification but from a proof-search point of view. Let us recall the existing definitions of MNL proof nets: one based on the notion of trip and another on the notion of order variety. For that we need to introduce the concept of switching.

Definition 15 *For any link l of a proof structure π, one defines a set $S(l)$ of partial functions, called* switching positions*, given in figure 3. Any function s such that, for any l of π, $s(l) \in S(l)$, is called a switching of π. A switching for π is $\lhd_3$-free if for every $\lhd$-link, $s(l) \neq \lhd_3$.*

Definition 16 *Let π be a proof structure and s be a switching for π, then $s(\pi)$ is the oriented graph with nodes being the decorated formulae of π and an oriented edge from A^x to B^y if and only if either there exists a link l such that $B^y = s(l)(A^x)$, or we have $A^x = C^{\downarrow}$ and $B^y = C^{\uparrow}$ and C is a conclusion of π. A* trip *is a cycle or a maximal path in $s(\pi)$.*

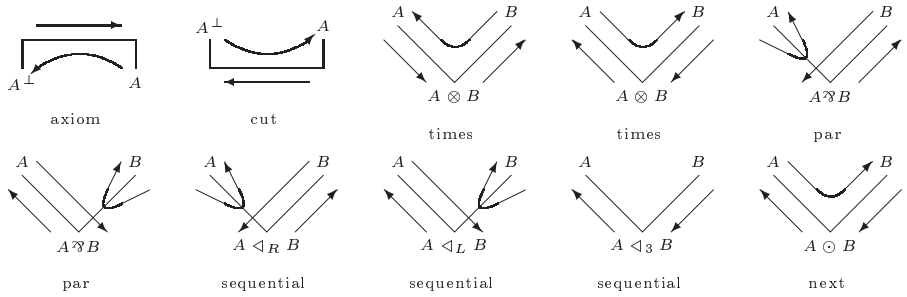


Fig. 3. Switching positions for MNL

Definition 17 *Let π be a proof structure and $s(\pi)$ a switching for π, a trip v in $s(\pi)$ is said* long *if v is a cycle and any occurrence formula A appears twice: one time decorated with $\downarrow$, another time with $\uparrow$. A cycle v in $s(\pi)$ is said* bilateral *if v has not the form $A^x, .., B^y, .., A^{\overline{x}}, .., B^{\overline{y}}, .., A^x$ where A and B are formulae occurrences of π.*

Theorem 18 *Let π be a proof structure of MNL, then π is a proof net if and only if for any switching s of π : a) there exists exactly one cycle σ in $s(\pi)$, b) σ contains all the conclusions, c) σ is bilateral.*

This theorem extends the notion of trip to MNL proof nets and is proved in [1]. The next theorem gives a characterization of a MNL proof net π from its commutative translation π^* in MLL. The interest is that we can benefit of results on proof nets in MLL, by including some refinements or specializations. Details are given in [1].

Definition 19 *Let π be a proof structure of MNL, s be a switching for π, v a trip in $s(\pi)$. Let l be a $\lhd$ of π with A first premise, B second premise, when it exists, the part of v from $B^{\uparrow}$ and $A^{\downarrow}$, is called the* inner part *of l in v. Inner parts* do not overlap *if either one is included into the other or they are disjoints.*

Theorem 20 *Let π be a MNL proof structure, π is a MNL proof net if and only if its commutative translation π^* (obtained by replacing in π $\odot$ by $\otimes$ and $\lhd$ by $\parr$) is a proof net of MLL such that for every $\lhd_3$-free switching s for π, the inner parts of $\lhd$-links in $s(\pi)$ contain no conclusion and do not overlap.*

Let us come back to order varieties and analyze the relationships between proof nets and order varieties.

Definition 21 *Let π be a MNL proof net and s is a switching for π, the oriented graph with the conclusions of π as vertices and oriented edges from A to B iff there is no conclusion between $B^{\uparrow}$ and $A^{\downarrow}$ in the unique cycle σ in $s(\pi)$, is* the cycle of the conclusions in $s(\pi)$

Definition 22 *Let π be a proof net of MNL with conclusion Γ,*

a) If s is a switching for π, $\alpha_{\pi,s}$ is the total order variety on Γ corresponding to the cycle of the conclusions in $s(\pi)$.
b) $\alpha_\pi = \natural(\bigcap_s \alpha_{\pi,s})$.

These definitions give a way to directly compute the order variety that corresponds to a proof net, from the conclusion cycles. One associates, in an obvious way, a proof structure $\overline{D}$ with conclusion Γ to a proof D of conclusion $\vdash \Gamma\ \langle\alpha\rangle$ in the sequent calculus.

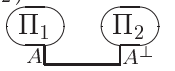
Theorem 23 *If D is a sequential proof with conclusion $\vdash \Gamma\ \langle\alpha\rangle$, then $\overline{D}$ is a proof net and $\alpha = \alpha_{\overline{D}}$ (adequacy).*
Let π be a cut-free proof net with conclusion Γ. There exists a sequent calculus proof D, with conclusion $\vdash \Gamma\ \langle\alpha\rangle$ such that $\pi = \overline{D}$ (sequentialization).

The proof is given in [1]. The definition 22 (through theorem 23) shows the relationships between the cycles of conclusion of a proof net and the order variety associated to the conclusion of the corresponding sequent proof. Following the relationships, the condition on the application of the $\lhd$ rule in the sequent calculus is translated in the proof net calculus into the condition about inner parts of $\lhd$ links.

5 Dependency proof nets for MNL

Here, we consider for each proof net Π a set of pairs of its conclusions $Dep(\Pi)$, called a dependency set. This set is inductively built in parallel with the proof net structures (see next definition).
$|Dep(\Pi)|_A$ (resp. $|Dep(\Pi)|_{A,B}$) represents the set of dependencies built from π with cancellation of the dependencies including A (resp. A or B). Moreover, if Π is a proof net then $Cc(\Pi)$ is the set of conclusions of Π.

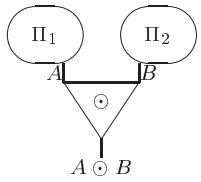
Definition 24 *The proof nets with dependencies (DepPN) for MNL, denoted $(\Pi, Dep(\Pi))$ are inductively defined as follows:*

- *Let Π be* *and $Dep(\Pi) = \{A \to A^\perp, A^\perp \to A\}$ then $(\Pi, Dep(\Pi)$ is an elementary DepPN.*
- *Let $(\Pi_1, Dep(\Pi_1)$ and $(\Pi_2, Dep(\Pi_2)$ be two distinct DepPNs, with respectively A and $A^\perp$ as conclusions then, $\Pi \equiv$*  *with*

$$
\begin{aligned}
Dep(\Pi) = {} & |Dep(\Pi_1)|_A \cup |Dep(\Pi_2)|_B \cup \\
& \{E \to F | E \in Cc(\Pi_1) \setminus A, F \in Cc(\Pi_2) \setminus A^\perp, E \to A \in Dep(\Pi_1), \\
& A^\perp \to F \in Dep(\Pi_2)\} \cup \\
& \{E \to F | E \in Cc(\Pi_2) \setminus A^\perp, F \in Cc(\Pi_1) \setminus A, E \to A^\perp A \in Dep(\Pi_2), \\
& A \to F \in Dep(\Pi_1)\}
\end{aligned}
$$

is a DepPN.

- *Let* $(\Pi_1, Dep(\Pi_1))$ *and* $(\Pi_2, Dep(\Pi_2))$ *be two distinct DepPNs, then*

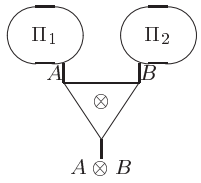
$\Pi \equiv$ 

and

$$\begin{aligned} Dep(\Pi) = {} & |Dep(\Pi_1)|_A \cup |Dep(\Pi_2)|_B \cup \\ & \{E \to F | E \in Cc(\Pi_1) \setminus A, F \in Cc(\Pi_2) \setminus B, \\ & E \to A \in Dep(\Pi_1), B \to F \in Dep(\Pi_2)\} \cup \\ & \{A \odot B \to E | E \in Cc(\Pi_1) \setminus A, \\ & A \to E \in Dep(\Pi_1)\} \cup \\ & \{F \to A \odot B | F \in Cc(\Pi_2) \setminus B, \\ & F \to B \in Dep(\Pi_2)\} \cup \\ & \{E \to A \odot B | E \in Cc(\Pi_1) \setminus A, \\ & E \to A \in Dep(\Pi_1), |Cc(\Pi_2)| = 1\} \cup \\ & \{A \odot B \to F | F \in Cc(\Pi_2) \setminus B, \\ & B \to F \in Dep(\Pi_2), |Cc(\Pi_1)| = 1\} \end{aligned}$$

is a DepPN.

- *Let* $(\Pi_1, Dep(\Pi_1))$ *and* $(\Pi_2, Dep(\Pi_2))$ *be two distinct DepPNs, then*

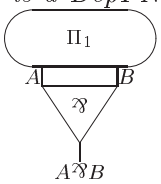
$\Pi \equiv$ 

and

$$\begin{aligned} Dep(\Pi) = {} & |Dep(\Pi_1)|_A \cup |Dep(\Pi_2)|_B \cup \\ & \{E \to F | E \in Cc(\Pi_1) \setminus A, F \in Cc(\Pi_2) \setminus B, \\ & E \to A \in Dep(\Pi_1), B \to F \in Dep(\Pi_2)\} \cup \\ & \{A \otimes B \to E | E \in Cc(\Pi_1) \setminus A, \\ & A \to E \in Dep(\Pi_1)\} \cup \\ & \{F \to A \otimes B | F \in Cc(\Pi_2) \setminus B, \\ & F \to B \in Dep(\Pi_2)\} \cup \\ & \{F \to E | E \in Cc(\Pi_1) \setminus A, F \in Cc(\Pi_2) \setminus B, \\ & A \to E \in Dep(\Pi_1), F \to B \in Dep(\Pi_2)\} \cup \\ & \{A \otimes B \to F | F \in Cc(\Pi_2) \setminus B, \\ & B \to F \in Dep(\Pi_2)\} \cup \\ & \{E \to A \otimes B | E \in Cc(\Pi_1) \setminus A, \\ & E \to A \in Dep(\Pi_1)\} \end{aligned}$$

is a DepPN.

- *if* Π_1 *is a DepPN with* A *and* B *as conclusions then*

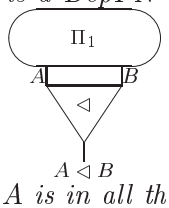
$\Pi \equiv$ 

and

$$\begin{aligned} Dep(\Pi) = {} & Dep(\Pi_1)[A \parr B/A, A \parr B/B] \\ & \{E \to F | E, F \in Cc(\Pi_1) \setminus \{A, B\}, \\ & E \to A, A \to F \in Dep(\Pi_1)\} \cup \\ & \{E \to F | E, F \in Cc(\Pi_1) \setminus \{A, B\}, \\ & E \to B, B \to F \in Dep(\Pi_1)\} \end{aligned}$$

is a DepPN.

- *if* Π_1 *is a DepPN with* A *and* B *as conclusions then*

$\Pi \equiv$ 

and $Dep(\Pi) = Dep(\Pi_1)[A \lhd B/A, A \lhd B/B]$ *is a DepPN if* $B \to A$ *is in all the maximal cycles in* $Dep(\Pi_1)$ $(\lhd^*)$.

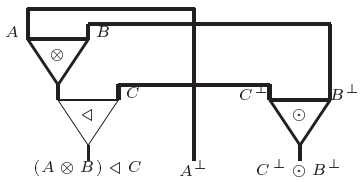
We observe that the construction of the dependencies in this inductive definition is similar to the one for the DepSC calculus given in the definition 8. The only syntactical differences come from the differences between sequents and proof nets in terms of notations. As observed in the DepSC definition, if we erase the dependencies sets, then the inductive definition is exactly the one proposed for MLL proof nets [6], considering $\odot$ and $\lhd$ respectively as $\otimes$ and $\parr$.

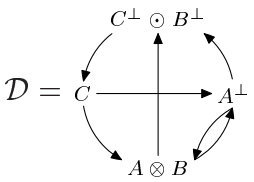
Let us comment this definition. The construction of $Dep(\Pi)$ is more complex for $\otimes$ than for $\odot$ because we extend the dependencies like for $A \odot B$ but also for $B \odot A$. In the case of $\parr$, we are obliged to apply the transitivity of the dependencies because the substitution of A and B by $A \parr B$ make lost the relationships between A and B. In the case of $\lhd$, the dependencies are the same than for $\parr$, except that, because of the condition $(\lhd^*)$, we know that $B \to A \in Dep(\Pi_1)$ and thus the transitivity is not necessary. As proved before for DepSC, we can replace the condition $(\lhd^*)$ by the equivalent condition $(\lhd^{**})$.

Theorem 25 *A proof structure $(\pi, Dep(\pi))$ is a dependency proof net iff π is a MNL proof net.*

PROOF. From theorem 20, a proof structure π is a proof net iff : i) π^* is a proof net of MLL, ii) for every $\lhd_3$-free switching s for π, the inner parts of $\lhd$-links in $s(\pi)$ contain no conclusion and do not overlap.
First, if we ignore $Dep(\Pi)$, the inductive structure Π is exactly π^*, considering $\odot$ and $\lhd$ as, respectively, $\otimes$ and $\parr$. Thus, every structure π, extracted from $\pi, Dep(\pi)$ is such that π^* is a proof net of MLL. Conversely, if π is a proof structure such that π^* is a proof net of MLL, then π can be constructed iff the condition of the $\lhd$-rule is verified for each $\lhd$-link in π.
Let π be extracted from a DepPN $(\pi, \mathcal{D})$, we prove, in theorem 27, that the maximal cycles of $\mathcal{D}$ correspond exactly to the cycles of conclusions in π. Thus, the condition $(\lhd^*)$ in the inductive definition insures that for every $\lhd$-link $A \lhd B$ and every switching s in π, there is no conclusion between $B^{\uparrow}$ and $A^{\downarrow}$ in the unique cycle in $s(\pi)$. By induction, we prove that the inner parts of $\lhd$-link do not overlap. A detailed proof is given in [11]. Finally, we prove that π is a MNL proof net. Moreover, a MNL proof net can be built using the inductive definition. This is a direct consequence of theorem 20, since the maximal cycles exactly correspond to the cycles of conclusions. Indeed, the condition on the inner parts of $\lhd$-links ensures that the condition on the application of the $\lhd$-rule is verified. $\square$

If we consider our example, with $\vdash (A \otimes B) \lhd C, A^{\perp}, C^{\perp} \odot B^{\perp}$, we have the following proof structure and dependency graph:





We have the same dependency graph as in the section dedicated to the DepSC calculus. Then we have to verify the condition $(\lhd^*)$ of $\lhd$ rules, or equivalently the condition $(\lhd^{**})$. Therefore the condition is not satisfied and the $\lhd$ rule is not applicable. As the choice for the axiom links to build is unique, we

cannot perform any backtracking, and then cannot build a proof net for the given sequent. Thus, we can deduce that the given sequent is not provable. In this case, we aim to consider countermodels from dependency graphs, that are semantic structures that can justify the non-provability. This point will be developed in further work. Let us now consider some properties of dependency proof nets.

Definition 26 *Let $\mathcal{C}$ be an oriented cycle, we define $\overline{\mathcal{C}}$ as the inverse of $\mathcal{C}$, i.e., for every couple of vertices (A, B), $A \to B \in \mathcal{C}$ iff $B \to A \in \overline{\mathcal{C}}$.*

Theorem 27 *Let π be a proof net, $\mathcal{C}$ is a maximal cycle in $Dep(\pi)$ iff $\overline{\mathcal{C}}$ is a cycle of conclusions in π.*

PROOF. The construction of a dependency set is made in respect with the switching positions of the various links. Thus, when we focus on maximal cycles we actually build the cycles of conclusions of the proof net. For the complete proof, we proceed by induction on π. □

From a MNL proof net, we can easily build a DepSC proof, and conversely, we can also build a proof net from a DepSC proof.

Definition 28 *To a proof D in DepSC with conclusion $\vdash \Gamma\ \mathcal{D}$, is associated a proof net with dependencies $\overline{D}$, with conclusion Γ.*

Theorem 29 *If D is a sequent calculus proof with conclusion $\vdash \Gamma\ \mathcal{D}$, then $\overline{D}$ is a proof net and $Dep\left(\overline{D}\right) = \mathcal{D}$ (**adequacy**).*
*Let π be a proof net with conclusion Γ, there exists a sequent calculus proof D, with conclusion $\vdash \Gamma$ $Dep(\pi)$ such that $\pi = \overline{D}$ (**sequentialization**).*

PROOF. Each inference rule in DepSC corresponds to an inductive rule in the DepPN calculus. Moreover, the calculus of dependency sets are similar in both definitions; the differences being only syntactical. Thus, the theorems of adequacy and sequentialization can be directly deduced from the theorems of adequacy and sequentialization of MLL proof nets vs sequent proofs in MLL. □

MNL has an initial proof nets theory [1] with different correctness criteria, for instance based on long trips or on contractibility notion [20] but mainly for cut-free proof nets and a question arises: is there a sequentialization theorem extended to proof nets with cuts (as in commutative LL) ? With our new proof net definition, and the related correctness criterion, we mainly work in fact in MLL (through π^*) with in addition the dependency sets. Thus, like in MLL, we can directly deal with proof nets with cuts and then give a natural solution to this question about sequentialization in MNL.
We can also consider the reduction rules ($\to$) of MNL proof structures as

in [1] and then show that if $(\pi, Dep(\pi))$ is a proof net and $\pi \rightarrow \pi'$, then $Dep(\pi') \subseteq Dep(\pi)$. From theorem 29, we can obtain the cut-elimination theorem for the DepSC calculus.

Theorem 30 (cut-elimination of DepSC calculus) *If D is a sequent calculus proof with conclusion $\vdash \Gamma \ \mathcal{D}$, then $\vdash \Gamma \ \mathcal{D}'$ is provable in the DepSC calculus without the cut rule, with $\mathcal{D} \subset \mathcal{D}'$*

PROOF. By theorem 29 (adequacy), $\overline{D}$ is a proof net and we can show that $\overline{D} \rightarrow \pi$ with π a cut-free proof net such that $Dep(\pi) \subseteq Dep(\overline{D})$. By theorem 29 (sequentialization), there exists a sequent calculus proof D' such that $\pi = \overline{D'}$. $\square$

6 Proof search and dependency proof nets

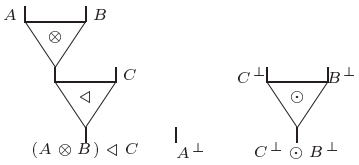
We have seen that both sequent calculi (with order varieties and with dependencies) are related on MLL. However, we have a procedure for automatic proof nets construction in MLL [9] that we can adapt to design a procedure for proof net construction in MNL. Let us mention the main principles of this procedure. From the inductive definition of MNL proof nets, we notice that we can build a new proof net by fusion and extension with a $\otimes$-link or a $\odot$-link from two disjoint proof nets and that we can build a new proof net by extension with a $\parr$-link or a $\lhd$-link from a given proof net with two conclusions. At this level, both commutative and non-commutative connectives have the same construction rules but their semantical differences are reflected in the dependency sets associated to the proof structures.
Thus, we apply the algorithm for MLL on the MNL sequent, without distinction between commutative and non-commutative connectives (through π^*), we build in parallel the dependency sets and we test the condition of the $\lhd$ rule. Therefore, our algorithm starts with the decomposition tree of the given sequent, that will be used as a frame to guide the proof net search. In addition, it manages a set $\mathcal{R}$ of intermediate proof nets that is initialized to the empty set. Then, starting from the literals (or leaves) of the decomposition tree, it builds step by step axiom-links, that are elementary proof nets added to $\mathcal{R}$. When conclusions of proof nets of $\mathcal{R}$ are premises of a $\otimes$ or $\odot$-link, they are linked and extended with this link to obtain a new proof net that replaces the two initial ones in $\mathcal{R}$. When conclusions of a given proof net of $\mathcal{R}$ are in fact premises of a $\parr$ or $\lhd$-link, then it is extended with this link to obtain a new proof net that replaces the initial one in $\mathcal{R}$. If two premises of a $\parr$ or $\lhd$-link (in the decomposition tree) are conclusions of two different nets in $\mathcal{R}$ then we postpone the extension of such a link. With respect to the permutability results in linear sequent calculus [13], we have to always apply, if possible, a $\otimes$ or $\odot$ (extension) rule before a $\parr$ or $\lhd$ (extension) rule. From these principles,

we can give a detailed algorithm, called MNL-PN (see appendix), for which we can prove that

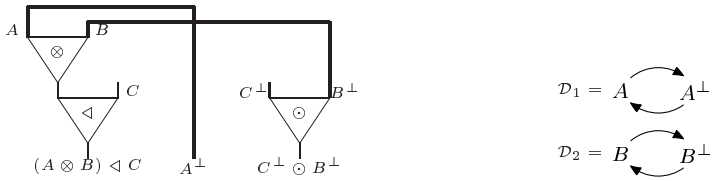
Theorem 31 *The MNL-PN algorithm, for dependency MNL proof nets, is correct and complete.*

PROOF. By the correctness and completeness of the similar procedure in MLL [8] and the definition of dependency proof nets.

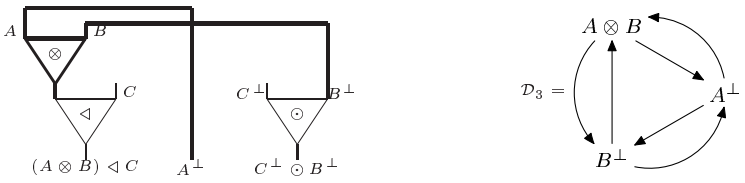
This result emphasizes one of the main contributions of this work, namely to consider provability in MNL from "classical" provability in MLL with simple refinements. Let us come back to the sequent $\vdash (A \otimes B) \lhd C, A^\perp, C^\perp \odot B^\perp$ and apply the different steps of the MNL-PN algorithm. The subnets under construction are represented in bold in the figures.
We first start by building the decomposition for the sequent to prove.



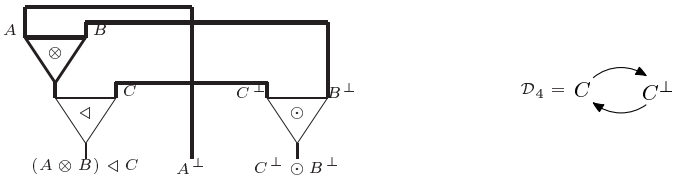
We then build the two elementary proof nets corresponding to the axioms $\vdash A^\perp, A$ and $\vdash B^\perp, B$.



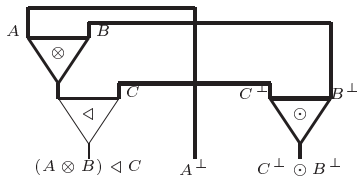
We can now apply the rule for the $\otimes$-link.

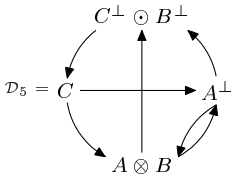


We now choose the literal $C^\perp$, and try to branch it to the free literal C. We obtain a new elementary proof net corresponding to the axiom $\vdash C^\perp, C$.



We apply the rule for the $\odot$-link, and thus obtain a new partial proof net.





The condition for the $\triangleleft$ rule is not satisfied (see previous sections). Thus, the $\triangleleft$ rule is not applicable. As the choice for the axiom links to build is unique, we cannot perform any backtracking, and then cannot build a proof net for the given sequent. Thus, we can deduce that the given sequent is not provable.

7 Proof net verification

We can observe that Our algorithm for MNL proof net construction can also be used for proof net verification, that is verifying that a given proof structure is a proof net. There are some proposals of correctness criteria for MNL proof net, for instance one based on long trips [1], another in the Danos-Regnier style. Both are in exponential time but recently Mogbil has given a quadratic criterion in the Danos contractibility criterion style [20]. We propose here to analyze the correctness criterion that is related the new definition of dependency MNL proof nets. We can prove that it is in quadratic time.
Informally, we only have to apply the MNL-PN algorithm to a given proof structure (the decomposition tree + the axiom-links) and thus to replace the search (or choice) of an axiom-link by the one given by the proof structure. Let us note that there is no need for backtracking in case of failure (for each free leaf, there is only one choice of branching). Then, the algorithm rebuilds the given proof structure, from the fixed choice of axiom-links, and also the associated dependency sets (or graphs). In the case of $\triangleleft$ rule we have to verify the condition $(\triangleleft^{**})$ on the dependency graph. In fact, the related correctness criterion consists in rebuilding the proof structure with its dependency sets and to verify the condition for the sets associated to $\triangleleft$ rules. Let us show that this criterion is in quadratic time.
The main point is that our construction of a dependency MNL proof net $(\Pi, Dep(\Pi)$ is directly based on the construction of a MLL proof net Π^* (commutative translation of Π). For that, we consider in fact our MLL algorithm, initially proposed in 1992 [12], but in the context of verification (to verify the correctness of a given MLL proof structure). This algorithm was dedicated to proof nets construction (never used for correctness verification) and has been also adapted for MCyLL proof nets [9]. Recently, Guerrini has shown that the MLL proof net verification is linear in time [18]. In fact, in order to capture the relationships with this work, we can refine our MLL algorithm in order to deal with the same data structures as in [18] and we can show, by same proof arguments, that the verification of MLL proof structures using this algorithm

is linear in time.
The algorithm for MNL dependency proof net construction is based on the MLL algorithm, builds in parallel the dependency sets associated to sub-nets and then verifies the conditions for the application of $\lhd$ rules. Therefore we are first interested on the complexity of the construction of dependency sets. If we represent a dependency set $\mathcal{D}$ over a set of formulae occurrences Γ by, for each formula $A \in \Gamma$, the list of its predecessors and the list of its successors, we can write procedures for construction of dependency sets, that are linear in time. Moreover, the verification of the condition on the application of $\lhd$-rules can be performed in constant time, thanks to the condition $(\lhd^{**})$.
Finally, the verification of MNL proof structures using our algorithm for MNL proof net construction can be performed in quadratic time. Thus we have a new quadratic correctness criterion based on dependency sets and it has to be compared with the quadratic criterion of [20] based on Danos contractibility criterion. Further work will be devoted to improve our based-on dependency criterion in order to have a better complexity for proof net construction and verification.

8 Conclusion

In this paper, we propose new calculi for the multiplicative fragment of Mixed Linear Logic (MNL), that are based on the definition of dependency relations. They give new proof-theoretic formulations of MNL in the spirit of [7] and also define provability criteria in MNL from provability in MLL and verification of conditions (for the $\lhd$ connectives) in so-called dependency graphs. The use of dependencies (or constraints) is a natural and fruitful way to translate some proof-theoretical results into simple and tractable automated proof-search procedures, that are based on the construction of semantical structures [8,10,16]. We give an algorithm for proof net construction in MNL and derive a new correctness criterion (for proof structure) based on dependency relations and prove that he is quadratic in time.
In further work, we will compare our algorithm with the previous ones we have recently developed [11]. Moreover, we will refine, if possible, the definition and construction of dependency sets in order to have efficient based-on proof nets MNL decision algorithms (and a better complexity of the related correction criteria). We also study how to consider a dependency graph as a countermodel in case of non-provability in MNL. In this context, we will also try to propose related resource semantics for this logic. Finally, we will apply these proof-theoretical and algorithmic results to the verification of properties of programs or protocols expressed in MNL.

References

[1] M. Abrusci and P. Ruet. Non-commutative logic I : the multiplicative fragment. *Annals of Pure and Applied Logic*, 101:29–64, 2000.

[2] V.M. Abrusci. Phase semantics and sequent calculus for pure noncommutative classical linear propositional logic. *Journal of Symbolic Logic*, 56(4):1403–1451, 1991.

[3] V.M. Abrusci. Noncommutative proof nets. In J.Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 271–296. Cambridge University Press, 1995.

[4] V. Alexiev. Applications of Linear Logic to Computation: An Overview. *Logic Journal of the IGPL*, 2(1):77–107, 1994.

[5] G. Bellin and P.J. Scott. On the π-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–66, 1994.

[6] G. Bellin and J. van de Wiele. Subnets of proof-nets in MLL^-. In J.Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 249–260. Cambridge University Press, 1995.

[7] T. Bräuner and V. De Paiva. A formulation of linear logic based on dependency-relations. In *Computer Science Logic, CSL'97, LNCS 1414*, pages 129–148. Springer Verlag, 1997.

[8] D. Galmiche. Connection Methods in Linear Logic and Proof nets Construction. *Theoretical Computer Science*, 232(1-2):231–272, 2000.

[9] D. Galmiche and B. Martin. Proof nets construction and automated deduction in non-commutative linear logic - extended abstract. *Electronic Notes in Theoretical Computer Science*, 17, 1998.

[10] D. Galmiche and D. Méry. Proof-search and countermodel generation in propositional BI logic - extended abstract -. In *4th Int. Symposium on Theoretical Aspects of Computer Software, TACS 2001, LNCS 2215*, pages 263–282, Sendai, Japan, 2001.

[11] D. Galmiche and J.M. Notin. Proof-search and proof nets in mixed linear logic. *Electronic Notes in Theoretical Computer Science*, 37, 2000.

[12] D. Galmiche and G. Perrier. A procedure for automatic proof nets construction. In *LPAR'92, International Conference on Logic Programming and Automated Reasoning, LNAI 624*, pages 42–53, St. Petersburg, Russia, July 1992.

[13] D. Galmiche and G. Perrier. On proof normalization in linear logic. *Theoretical Computer Science*, 135(1):67–110, 1994.

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

[15] J.Y. Girard. Linear logic: its syntax and semantics. In J.Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 1–42. Cambridge University Press, 1995.

[16] J.Y. Girard. Proof nets: the parallel syntax for proof theory. In A. Ursini and P. Agliano, editors, *Logic and Algebra*, New York, 1995. M. Dekker.

[17] D. Gray, G. Hamilton, J. Power, and D. Sinclair. Specifying and verifying TCP/IP using Mixed Intuitionistic Linear Logic. Technical report, Dublin City University, 2001.

[18] S. Guerrini. Correctness of multiplicative proof nets is linear. In *14h Symposium on Logic in Computer Science*, pages 454–463, Trento, Italy, July 1999. IEEE Computer Society Press.

[19] N. Kobayashi and A. Yonezawa. Asynchronous communication model based on linear logic. *Formal Aspects of Computing*, 3:279–294, 1994.

[20] V. Mogbil. Quadratic correctness criterion for non commutative logic. In *15th Int. Workshop on Computer Science Logic, CSL 2001, LNCS 2142*, pages 69–83, Paris, France, 2001.

[21] P. Ruet. Non-commutative logic II : sequent calculus and phase semantics. *Math. Struct. in Comp. Science*, 10:277–312, 2000.

[22] P. Ruet and F. Fages. Concurrent constraint programming and non-commutative logic. In *11th Int. Workshop on Computer Science Logic, CSL'97, LNCS 1414*, pages 406–423, Aarhus, Denmark, August 1997.

A The algorithm for MNL proof nets construction

The algorithm starts by building the decomposition tree of the sequent to prove, and choose a free leaf (atomic formula) to treat. Then, it try to build axiom links, that are elementary subnets, and apply the rules corresponding to the links (these rules are those of the inductive definition). Thus, it inductively builds new subnets from the others, and computes in parallel the associated dependency sets. If the construction is not possible, it backtracks (if it is possible) and tries another choice for the connections.
The algorithm could be presented in a similar way that the algorithms for MLL and MCyLL in [8,9] but here we want to emphasize the direct uses of stacks to deal with the various data. Then, we have

- *cur_ftt*: formula to treat.
- *sibling(cur_ftt)* is the second premise of the link having *cur_ftt* as first premise.
- *ftt*: stack of formulae to treat.

- *ParWaiting*: stack of disjunctive links with both premises treated but not elements of the same subnet.
- *saved_states*: stack where states are saved, in order to perform backtracking if necessary.

---

choose cur_ftt among the free leaves; ftt := ()
failure := false;
while cur_ftt $\neq \emptyset$ and not failure **do**
  if cur_ftt is a free_leaf **then**
    duals := list of free leafs that are duals of cur_ftt
    dual := pop(duals) *# choice of a literal to branch with cur_ftt*
    push(current_state, saved_states) *# save the current state, so we can eventually backtrack*
    push(dual, ftt) *# the dual of cur_ftt is added is the formulae to treat*
    create a new subnet corresponding to the axiom formed by cur_ftt and its dual
  else if cur_ftt is a premise of a conjunctive link L **then**
    if treated(sibling(cur_ftt)) **then**
      if cur_ftt and sibling(cur_ftt) in two distinct subnets Π_1 and Π_2 (1)
      then
        unify Π_1 and Π_2 in Π (2)
        compute the dependency set associated to Π (i)
        push(ftt, L)
        try to find links in ParWaiting that can be treated, and add them at the end of ftt (3)
        cur_ftt := pop(ftt)
      else
        if saved_states $\neq$ () **then** restore_state **else** failure := true **endif**
      end if
    else
      cur_ftt := a leaf beyond cur_ftt
    end if
  else if cur_ftt is a premise of a disjunctive link L **then**
    if treated(sibling(cur_ftt)) **then**
      if cur_ftt and sibling(cur_ftt) in the same subnet Π (1') **then**
        if L is a $\lhd$ link and the condition on $\lhd$ links is not satisfied **then**
          if saved_states $\neq$ () **then** restore_state **else** failure := true **endif**
        end if
        compute the new dependency set associated to Π (ii)
        push(ftt, L)
      else
        add L in ParWaiting
      end if
    end if

cur_ftt := pop(ftt)
else
cur_ftt is a conclusion of the structure
if ftt = () **then** add all elements of ParWaiting in ftt **endif**
cur_ftt := pop(ftt)
end if
end while
if all sub formulae are treated and in the same proof net Π **then** return Π
else return Failure **endif**

Let us comment some steps of this algorithm, mainly from the complexity point of view. The steps dealing with subnets can be implemented following the approach of [18]. Each subformula is marked with a token (a number), and a partition of tokens represents the subnets under construction. To check if two formulae belong or not to the same proof net ((1) and (1')) correspond to check that the tokens representing the formulae are or not in the same partition. The operation of unification (2) of two proof nets Π_1 and Π_2 simply modifies the two partitions representing Π_1 and Π_2.
Moreover, with a fine handling of ParWaiting, the step (3) can be very simple: it is sufficient to identify the partition for which the disjunctive link is waiting. Thus, when we unify this partition with another one, we can try to treat the disjunctive links waiting for it. Thanks to the strategy used in this algorithm, the partition of tokens can be implemented by a stack of numbers. Consequently, the operations of unification or verification if two tokens are whether or not in the same partition, are linear in time. If we represent a dependency set $\mathcal{D}$ over a set of formulae occurrences Γ by, for each formula $A \in \Gamma$, the list of its predecessors and the list of its successors, we can write procedures implementing (i) and (ii) that are linear in time. Moreover, the verification of the condition on the application of $\lhd$-rules can be performed in constant time, thanks to the condition ($\lhd^{**}$).

B A proof of theorem 9

Theorem 9 *Let Π be a proof in the DepSC dependency calculus, with conclusion $\vdash \Gamma\ \mathcal{D}$, there exists a maximal cycle in $\mathcal{D}$.*

PROOF. We proceed by induction on the structure of Π.

- if Π is an axiom, $\mathcal{D}$ is $\{A^\perp \to A, A \to A^\perp\}$ which is directly a maximal cycle.
- if Π ends by $\frac{\vdash \Gamma, A\ \mathcal{D}_1 \qquad \vdash B, \Delta\ \mathcal{D}_2}{\vdash \Gamma, A \odot B, \Delta\ \mathcal{D}} \odot$. By hypothesis, $\mathcal{D}_1$ contains a maximal cycle $A \to E_1 \to \cdots \to E_m \to A$, and $\mathcal{D}_2$ contains a maximal cycle $B \to F_1 \to \cdots \to F_n \to B$, where $E_1 \ldots E_m$ and $F_1 \ldots F_n$ are conclusions of

respectively Γ and Δ. Then, by construction of $\mathcal{D}$, we can deduce a maximal cycle in $\mathcal{D}$ from those in $\mathcal{D}_1$ and $\mathcal{D}_2$. Indeed, the cycle $A \odot B \to E_1 \to \cdots \to E_m \to F_1 \to \cdots \to F_n \to A \odot B$ is a maximal cycle in $\mathcal{D}$.

- if Π ends by $\frac{\vdash \Gamma, A \ \mathcal{D}_1 \qquad \vdash B, \Delta \ \mathcal{D}_2}{\vdash \Gamma, A \otimes B, \Delta \ \mathcal{D}} \otimes$, the argument is similar to the previous case.
- if Π ends by $\frac{\vdash \Gamma, A, B \ \mathcal{D}_1}{\vdash \Gamma, A \lhd B \ \mathcal{D}} \lhd$. Then, thanks to the condition on the application of the $\lhd$-rule, all the maximal cycles in $\mathcal{D}_1$ have the form: $B \to A \to E_1 \to \cdots \to E_m \to B$. Thus, by construction of $\mathcal{D}$, we can deduce a maximal cycle in $\mathcal{D}$, namely $A \lhd B \to E_1 \to \cdots \to E_m \to A \lhd B$.
- if Π ends by $\frac{\vdash \Gamma, A, B \ \mathcal{D}_1}{\vdash \Gamma, A \parr B \ \mathcal{D}} \parr$. Consider a maximal cycle in $\mathcal{D}_1 : B \to E_1 \cdots E_k \to A \to E_{k+1} \to E_m \to B$. From, this maximal cycle, we can build : $A \parr B \to E_1 \cdots E_k \to E_{k+1} \to E_m \to A \parr B$, which is a maximal cycle in $\mathcal{D}$.
- if Π ends by $\frac{\vdash \Gamma, A \ \mathcal{D}_1 \qquad \vdash A^\perp, \Delta \ \mathcal{D}_2}{\vdash \Gamma, \Delta \ \mathcal{D}} cut$. By hypothesis, $\mathcal{D}_1$ contains a maximal cycle $A \to E_1 \to \cdots E_m \to A$ and $\mathcal{D}_2$ contains a maximal cycle $A^\perp \to F_1 \to \cdots F_n \to A^\perp$. Thus, by construction of $\mathcal{D}$, we can deduce a maximal cycle in $\mathcal{D}$, namely $E_1 \to \cdots E_m \to F_1 \to \cdots F_n \to E_1$ $\square$

This proof of theorem 9 can be used for the proof of theorem 11, observing more deeply the construction of maximal cycles according to the rules for dependency sets construction.