Logic Programming with Multiplicative Structures

Matteo Acclavio*

University of Sussex, Brighton (UK)
and
University of Southern Denmark, Odense (DK)

Roberto Maieli†

Dipartimento di Matematica e Fisica
Università Roma Tre
Roma, Italy

In the logic programming paradigm, a program is defined by a set of methods, each of which can be executed when specific conditions are met during the current state of an execution. The semantics of these programs can be elegantly represented using sequent calculi, in which each method is linked to an inference rule. In this context, proof search mirrors the program's execution. Previous works introduced a framework in which the process of constructing proof nets is employed to model executions, as opposed to the traditional approach of proof search in sequent calculus.

This paper further extends this investigation by focussing on the pure multiplicative fragment of this framework. We demonstrate, providing practical examples, the capability to define *logic programming methods* with context-sensitive behaviors solely through specific resource-preserving and context-free operations, corresponding to certain generalized multiplicative connectives explored in existing literature. We show how some of these methods, although still multiplicative, escape the purely multiplicative fragment of Linear Logic (MLL, containing only $⅋$ and $⊗$).

1 Introduction

Proof theory provides various paradigms for interpreting computations as proofs and their transformations. The renowned Curry-Howard correspondence interprets proofs as programs, and proof reduction (i.e., cut-elimination) as program execution. This correspondence offers an elegant model for *functional programming*, where the primary computation mechanism is substitution. In this context, well-typed programs are expected to terminate their execution after computing results derived by the complete initial information. However, this paradigm appears to face challenges in representing programs where the main computational mechanism is not substitution, as well as the ones characterized by non-termination, partial information, and strong concurrency (see, e.g., distributed systems, database servers, and microservices architectures). By means of example, consider the way of modeling the Curry-Howard correspondence in the case of non-terminating programs, where we need to consider *infinitary proof systems* to be able to represent infinite programs as *non-wellfounded derivations*. In these systems, even basic results like soundness, completeness, and cut-elimination require complex techniques [17, 22, 23, 4, 56, 5]. Therefore, it may appear more intuitive to interpret the rules of operational semantics for these programs as rules of a sequent system, and the program execution as the process of proof search [51, 50, 13, 15]. This approach naturally handles issues related to partial information, the concurrency of rule applications, and the possibility of non-termination.

Alternatively, in the *logic programming* paradigm, a program is provided by a set of *methods*, which are elementary programs that can be executed when specific preconditions are met. The conventional proof-theoretical interpretation of logic programming associates a sequent of formulas with each state of the computation, and a sequent calculus rule with each program method. This establishes an intuitive connection between program execution and the process of proof search within the calculus. In this

---

*The author is supported by Villum Fonden, grant no. 50079.

†The author is supported by INdAM-GNSAGA.

Paradigm	Curry-Howard	Logic programming (sequent calculus)	Logic programming (multiplicative structures)
State	Proof - with cuts; - without proper axioms (i.e., without open premises);	Proof - without cuts; - possibly with proper axioms (i.e., with open premises);	Multiplicative structure: - (transitory) component; - possibly with inputs;
Computation step	Proof reduction: Cut-elimination	Proof construction: Proper axioms elimination	Proof net expansion: Proof structure expansion
Final state	Cut-free proof	Derivation with closed branches	Structure without inputs
Program type	Formula	Formula	Network behavior

Figure 1: A summary of the interpretation of proofs-as-programs in the paradigms of functional programming, and logic programming using sequent calculus and using proof net expansion.

context, a derivation tree where all leaves are axioms of the system represents a successfully completed computation.

It's worth noting that two forms of non-determinism arise during the process of proof search, corresponding to two distinct notions of non-determinism in program executions. Using the terminology from [12, 38, 43], The first type of non-determinism arises from the possibility of applying multiple methods to separate sub-sequents, which is a consequence of the limitations of sequent calculus[1]. The second form of non-determinism is observed when different methods are applied to non-disjoint sequents.

In this paper we continue the investigation on the interpretation of logic programming based on linear logic *proof structure expansion* instead of sequent calculus proof search, as in [14, 15, 29, 35]. In this approach, the set of inputs of a proof structure is interpreted as the current state of an execution, and the process of connecting new proof structures to its input (called *expansion*) is interpreted as the application of a method. The motivation to employ proof structures is due to their efficacy in capturing the non-determinism arising from the constraints of sequent calculus syntax: the graphical syntax relieves us from the bureaucracy of rules permutations between independent rule applications. Additionally, proof structures offer a more flexible structure, enabling us to define methods corresponding to the expansion of multiple branches simultaneously.

Contributions of the paper. In this paper, we study a logic programming framework built upon linear logic proof structures, offering a generalization of the standard MLL-*proof structure* [30] and the focused bipolar proof structures [15]. We focus on the multiplicative fragment of this framework, where the Danos-Regnier *correctness criterion* [21] can be easily generalized.

We introduce the concept of a *component* as an acyclic multiplicative structure where each of its part can interact with a context, analogous to the notion of an *open derivation* in sequent calculus. After establishing the topological conditions necessary for ensuring the composability of components, we proceed to define the foundational blocks of a logic programming framework based in the expansion of proof structures. Within this framework, as main novelty, we offer a computational interpretation of a specific family of *generalized multiplicative connectives*, which are connectives provided with linear and context-free introduction rules proposed in early works on linear logic [21, 32], but which lacked of any concrete computational interpretation prior to this work. We conclude by illustrating methods, defined within a linear and context-free setting, whose behavior is "locally additive" (in the sense of [32]), which cannot be expressed using the conventional MLL connectives $⅋$ and $\otimes$ (see [44, 9, 47]).

---

[1] In sequent calculus, two independent rules which can be applied to a same sequent must be sequentialized because the syntax does not allow for the application of rules to portions of a sequent. At the same time, if proof search produces a branching, then the two branches of the proof search can be performed independently in a true concurrent way.

Outline of the paper. In Section 2 we recall some notions on hypergraphs, partitions and the definition of multiplicative linear logic and its proof nets. In Section 3 we recall the definition and results from [44, 9] on the generalized multiplicative connectives (theorized in [21, 32]) we use in this paper. In Section 4 we give an overview of the way logic programming program executions are represented in using sequent calculi and how this paradigm has been extended in [15] to proof net expansion. In Section 5 we define our framework by extending the definition of proof structures, and proving the results about their compositionality. In Section 6 we provide a computational interpretation of certain generalized multiplicative connectives in our framework. We show that these connectives, beside being linear and context-free operators, are still able to capture non-linear and context-sensitive behaviors.

2　Preliminary Notions

In this section we recall basic definitions for hypergraphs and partitions we use in this paper. We then recall the definition of multiplicative linear logic and the syntax of its proof nets.

2.1　Hypergraphs

A **hypergraph** $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ is given by a set of **vertices** $V_{\mathcal{G}}$ and a set of **hyperedges** $E_{\mathcal{G}}$, that is, a set of pairs of list of vertices in $V_{\mathcal{G}}$. In a hyperedge $h = \langle \mathsf{in}(h), \mathsf{out}(h) \rangle \in E_{\mathcal{G}}$ we call the vertices occurring in $\mathsf{in}(h)$ (resp. in $\mathsf{out}(h)$) the **inputs** (resp. the **outputs**) of h and we define the **border** $\mathsf{B}(h)$ as the multiset of vertices occurring in $\mathsf{in}(h)$ and in $\mathsf{out}(h)$. A **sub-hypergraph** of a hypergraph $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ is a hypergraph $\mathcal{G}' \langle V_{\mathcal{G}'}, E_{\mathcal{G}'} \rangle$ such that $V_{\mathcal{G}'} \subseteq V_{\mathcal{G}}$ and $E_{\mathcal{G}'} \subseteq \{h \in E_{\mathcal{G}} \mid \mathsf{B}(h) \subseteq V_{\mathcal{G}'}\}$.

An **input** (resp. **output**) of the hypergraph $\mathcal{G}$ is a vertex which does not occur as output (resp. input) of any hyperedge of $\mathcal{G}$. We denote by $\mathsf{I}_{\mathcal{G}}$ (resp. $\mathsf{O}_{\mathcal{G}}$) the set of inputs (resp. outputs) of $\mathcal{G}$, and we define the **border** of $\mathcal{G}$ as the set of its inputs and the outputs, that is, $\mathsf{B}_{\mathcal{G}} = \mathsf{I}_{\mathcal{G}} \cup \mathsf{O}_{\mathcal{G}}$. A hypergraph is **linear** if each vertex occurs at most once as an input of a hyperlink and as an output of another link[2].

In a hypergraph $\mathcal{G}$, a **path** (of length n) from $x \in V_{\mathcal{G}}$ to $y \in V_{\mathcal{G}}$ is an alternating list of vertices and hyperedges of the form $x = v_0 h_1 v_1 \cdots h_n v_n = y$ such that $v_{i-1} = \mathsf{out}(h_i)$ and $v_i = \mathsf{in}(h_i)$ for all $i \in \{1, \ldots, n\}$; in this case we say that x is connected to y. A **cycle** is a path with $h_1 = h_n$, or $n > 0$ and $v_0 = v_n$; it is **elementary** if there are no i and j such that $i \neq j$ and $v_i = v_j$ or $h_i \neq h_j$. A hypergraph is **acyclic** if it contains no elementary cycles.
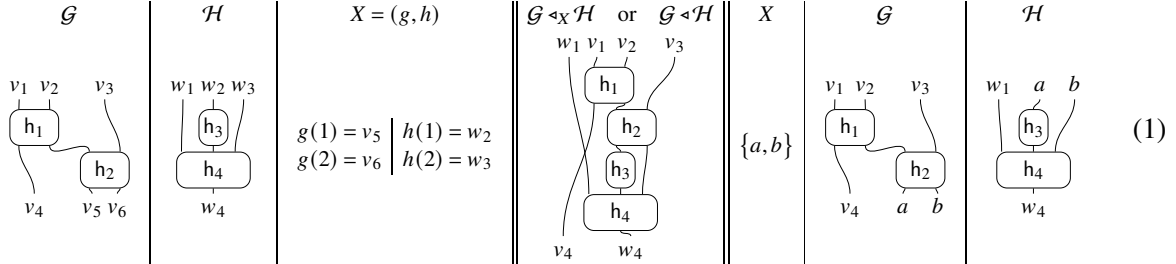
An **undirected hypergraph** $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ is given by a set of **vertices** $V_{\mathcal{G}}$ and a set of **undirected hyperedges** $E_{\mathcal{G}}$, that is, a set of subset of vertices in $V_{\mathcal{G}}$. A **graph** is an undirected hypergraph in which each hyperedge is an **edge**, that is, a set of two vertices $\{v_1, v_2\}$. **Paths** and **cycles** in an undirected hypergraph are defined analogously to the ones in hypergraph. Two vertices are **connected** if there is a path from one to the other. A **connected component** of an undirected hypergraph is a maximal subset of pairwise connected vertices. The **undirected hypergraph** associated to a linear hypergraph $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ is defined as the undirected hypergraph $\langle V_{\mathcal{G}}, \{\mathsf{B}(h) \mid h \in E_{\mathcal{G}}\} \rangle$. We say that a hypergraph $\mathcal{G}$ is **connected** if the undirected hypergraph associated to it is connected.

Notation 1. We drawing hyperedges with inputs on top and output on the bottom. By convention, we enumerate the inputs and outputs from left to right.

Definition 2. Let $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ and $\mathcal{H} = \langle V_{\mathcal{H}}, E_{\mathcal{H}} \rangle$ be two hypergraphs with disjoint sets of vertices. The **disjoint union** of $\mathcal{G}$ and $\mathcal{H}$ (denoted $\mathcal{G} \| \mathcal{H}$) is defined as the union of vertices and hyperedges of

---

[2]In works on hypergraphs with interfaces (e.g., [19]), this property is referred to as *linearity* or *monogamicity*. Note that, by definition, no vertex can occur at the same time as an input and an output of a linear hyperedge.

$\mathcal{G}$ and $\mathcal{H}$ that is, $\mathcal{G}\|\mathcal{H} := \langle V_{\mathcal{G}} \cup V_{\mathcal{H}}, E_{\mathcal{G}} \cup E_{\mathcal{H}}\rangle$. Note that in defining $\mathcal{G}\|\mathcal{H}$ we always assume $\mathcal{G}$ and $\mathcal{H}$ having disjoint sets of nodes. An **(linear) interface** $X = (g, h)$ is a pair of bijections from a finite set $\{1, \ldots, n\}$ to $V_{\mathcal{G}}$ and $V_{\mathcal{H}}$ respectively. We define the **composition** of $\mathcal{G}$ and $\mathcal{H}$ via an interface X as the hypergraph $\mathcal{G} \triangleleft_X \mathcal{H}$ obtained by identifying in $\mathcal{G}\|\mathcal{H}$ the vertices $g(i)$ and $h(i)$ for each $i \in \{1, \ldots, n\}$, as shown on the left of Equation (1).

$$
\begin{array}{c|c|c|c|c|c|c}
\mathcal{G} & \mathcal{H} & X = (g,h) & \mathcal{G}\triangleleft_X\mathcal{H} \text{ or } \mathcal{G}\triangleleft\mathcal{H} & X & \mathcal{G} & \mathcal{H}
\end{array}
\tag{1}
$$

To improve readability, we may simply identify the vertices $g(i)$ and $h(i)$ for all $i \in \{1, \ldots, n\}$, therefore considering X as a set of vertices in $V_{\mathcal{G}} \cap V_{\mathcal{H}}$ and simply writing $\mathcal{G} \triangleleft \mathcal{H}$, as shown in the right of Equation (1).

2.2   Partitions

Given a set X, a **partition** of X is a set of disjoint subsets of X (we call each a **block**) such that their union is X. In order to improve readability, when writing sets of partitions, in which three parentheses are nested inside each other, even if blocks and partitions are sets (not permutations, nor multisets), we use parentheses $(-)$ to denote blocks (subsets of X), and square brackets $[-]$ to denote partitions (sets of subsets of X). For example, we write $[(1,3),(2)]$ to denote the partition of the set $\{1,2,3\}$ with one block containing 1 and 3 and one block containing only 2. We denote by $\mathbb{P}_X$ (resp. $\mathbb{P}_n$) the set of partitions over X (resp. over $\{1, \ldots, n\}$). If $p \in \mathbb{P}_X$ and $Y \subset X$, we define the **restriction of** p **on** Y as the partition $p_{|Y} \in \mathbb{P}_Y$ such that $x, y \in Y$ belongs to the same block $\gamma_{|Y} \in p_{|Y}$ iff x and y belongs in a same block $\gamma \in p$. By means of example, if $p = [(1,3,4),(2,5),(6)] \in \mathbb{P}_6$, then $p_{|\{1,2,3\}} = [(1,3),(2)]$.
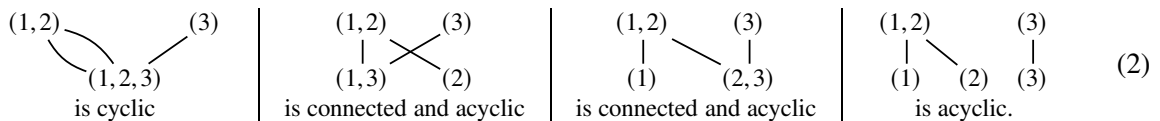
Definition 3 (Orthogonality [21])**.** Let X be a set and $p, q \in \mathbb{P}_X$. We define **graph of incidence of** p **and** q as the graph $\mathcal{G}(p,q)$ with vertices the blocks in p and in q and with an edge between a block $\gamma_p \in p$ and a block in $\gamma_q \in q$ for each $i \in \gamma_p \cap \gamma_q$ (see examples in Equation (2)). That is, the graph $\mathcal{G}(p,q) = \langle V_{\mathcal{G}(p,q)}, E_{\mathcal{G}(p,q)}\rangle$ has set of vertices and edges respectively

$$V_{\mathcal{G}(p,q)} = \{\gamma \mid \gamma \in p \text{ or } \gamma \in q\} \quad \text{and} \quad E_{\mathcal{G}(p,q)} = \{\{\gamma_i^p, \gamma_i^q\} \mid \gamma_i^p \in p \text{ and } \gamma_i^q \in q \text{ and } i \in \gamma_i^p \cap \gamma_i^q\}.$$

We say that $p, q \in \mathbb{P}_X$ are **weakly orthogonal**, denoted $p \perp_w q$, if their graph of incidence $\mathcal{G}(p,q)$ is acyclic. They are **orthogonal**, denoted $p \perp q$, if their graph of incidence is connected and acyclic.

The notion of weak orthogonality and orthogonality extends to sets of partitions: if $P, Q \subset \mathbb{P}_X$ then $P \perp_w Q$ (respectively $P \perp Q$) if $p \perp_w q$ (respectively $p \perp q$) for all $p \in P$ and for all $q \in Q$. The **orthogonal set** of a set of partitions $P \subset \mathbb{P}_n$ is defined as $P^{\perp} = \{q \in \mathbb{P}_n \mid p \perp q \text{ for all } p \in P\}$. We write $P \perp\!\!\!\perp Q$ if $P \perp Q$ and $P^{\perp} \perp Q^{\perp}$.

Example 4. Consider the partitions $p = [(1,2),(3)]$, $q_1 = [(1,2,3)]$, $q_2 = [(1,3),(2)]$ $q_3 = [(1),(2,3)]$ and $q_4 = [(1),(2),(3)]$. We have that $p \not\perp q_1$, $p \perp q_2$, $p \perp q_3$, $p \perp_w q_4$ because

$$
\begin{array}{c|c|c|c}
(1,2) \quad (3) & (1,2) \quad (3) & (1,2) \quad (3) & (1,2) \quad (3) \\
(1,2,3) & (1,3) \quad (2) & (1) \quad (2,3) & (1) \quad (2) \quad (3) \\
\text{is cyclic} & \text{is connected and acyclic} & \text{is connected and acyclic} & \text{is acyclic.}
\end{array}
\tag{2}
$$

$$\frac{}{\vdash a, a^\perp}\ \mathsf{ax} \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \mathbin{\bindnasrepma} B}\ \mathbin{\bindnasrepma} \qquad \frac{\vdash \Gamma, A \quad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta}\ \otimes \quad \Big| \quad \frac{\vdash \Gamma \quad \vdash \Delta}{\vdash \Gamma, \Delta}\ \mathsf{mix} \quad \Big| \quad \frac{\vdash \Gamma, A \quad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}\ \mathsf{cut}$$

Figure 2: Sequent calculus rules for the multiplicative linear logic, and rules mix and cut.

2.3 Multiplicative Linear Logic

Multiplicative linear logic formulas are generated from a countable set $\mathcal{A} = \{a, b, \dots\}$ of **propositional variables** by the following grammar:

$$A, B ::= a \mid A^\perp \mid A \mathbin{\bindnasrepma} B \mid A \otimes B$$

We consider formulas modulo the **involutivity of the negation** $A^{\perp\perp} = A$ and the **De Morgan** laws ($A \otimes B)^\perp = A^\perp \mathbin{\bindnasrepma} B^\perp$ and $(A \mathbin{\bindnasrepma} B)^\perp = A^\perp \otimes B^\perp$.

A **sequent** is a set of occurrences of formulas (as in, e.g., [17, 5]). The **sequent systems** MLL = $\{\mathsf{ax}, \mathbin{\bindnasrepma}, \otimes\}$ and MLL° = MLL $\cup \{\mathsf{mix}\}$ are defined using the rules in Figure 2[3]. We call **active** (resp. **principal**) a formula occurrence in one of the premises (resp. in the conclusion) of a rule, not occurring in conclusion (resp. in any of its premises).

Multiplicative Proof Nets *Proof nets* are a graphical syntax for multiplicative linear logic proofs capturing the proof equivalence generated by independent rules permutations (see, e.g., [31, 41, 36, 40, 10, 37, 1, 11]). A **proof structure** $\mathcal{S} = \langle V_{\mathcal{S}}, E_{\mathcal{S}} \rangle$ is a hypergraph whose vertices are labeled by MLL-formulas and whose hyperedges (called **links**) are labeled by rules in MLL (such labels are called **types**) in such a way the following local constraints are respected:

(3)

with A and B formulas and $a \in \mathcal{A}$.

Since we are considering a sequent as a set of occurrences of formulas, it is possible to easily trace formula occurrences in a derivation, defining a proof structure that encodes a given derivation.

Definition 5. Let π be a derivation in MLL. We define the proof structure **representing** π as the proof structure $\mathcal{P}_\pi$ having a vertex for each occurrence of an active formulas of a rule in π, and a link of type ρ with inputs (resp. with output) the vertices corresponding to the active formulas (resp. the principal formula) for each occurrence of a rule ρ in π. A **proof net** is a proof structure $\mathcal{S} = \mathcal{P}_\pi$ representing a derivation π in MLL.

By definition not all proof structures are proof nets. For this reason, a **correctness criterion**, that is, a topological characterization of those proof structures which are proof nets, is needed. Beside various criteria have been developed in the literature [30, 21, 55, 34], we here report the so-called **Danos-Regnier criterion** (or **DR-criterion** for short), which is the most relevant to our purposes.

---

[3]In the figure we include the rule cut required to define compositionality of proofs via *modus ponens*. We do not include it in the definition of MLL and MLL° since this rule is proven to be admissible [30] and it plays no role in the framework we are presenting in this paper.
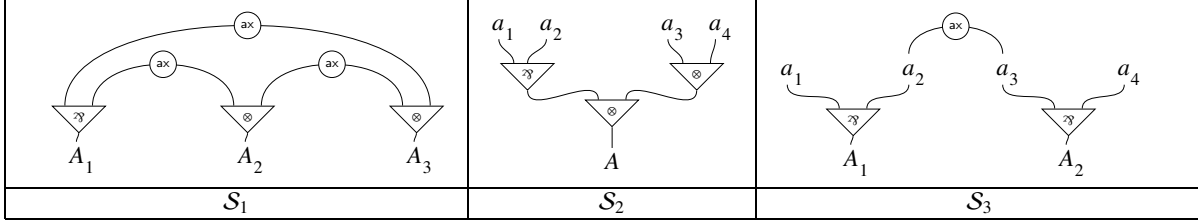
Figure 3: Examples of proof structures: $\mathcal{S}_1$ is a proof net (therefore a module), $\mathcal{S}_2$ is a module, and $\mathcal{S}_3$ is not a module (it admits a test in which a_2 and a_3 are disconnected from any other vertex, therefore $\mathcal{S}_3$ cannot be a sub-hypergraph of a proof net).

Definition 6. Let $\mathcal{M}$ be a proof structure. A **test** for $\mathcal{M}$ is the undirected hypergraph with the same vertices of $\mathcal{M}$ having a hyperedge $\{a, a^{\perp}\}$ for each ax-link $(\emptyset, \langle a, a^{\perp} \rangle)$, a hyperedge $\{A, B, A \otimes B\}$ for each $\otimes$-link $(\langle A, B \rangle, \langle A \otimes B \rangle)$, and either one edge $\{A, A \,\mathcal{B}\, B\}$ or one edge $\{B, A \,\mathcal{B}\, B\}$ for each $\mathcal{B}$-link $(\langle A, B \rangle, \langle A \,\mathcal{B}\, B \rangle)$. The proof structure $\mathcal{M}$ is **DR-correct** if it has no inputs and if all of its tests are connected and acyclic (undirected) hypergraph.

Theorem 7 ([21]). *A proof structure $\mathcal{S}$ is a proof net iff $\mathcal{S}$ is DR-correct.*

It is worth noticing that by dropping connectedness condition in Definition 6, we obtain a notion of correctness for MLL°-proof net, that is, if any test of a proof structure $\mathcal{M}$ is an acyclic hypergraph, then we can associate to $\mathcal{M}$ a derivation in MLL°.

Definition 8. A **module** is a proof structure which is a connected sub-hypergraph of a proof net.

Remark 9 (Definitions of module in the literature). The definition of module we consider in this paper differs from the definition of module given in [21] where a module is defined as a pair $\langle \mathcal{S}, \mathsf{Y} \rangle$ with $\mathcal{S}$ a proof structure such that $\sigma(\mathcal{S})$ is acyclic for all $\sigma \in \Sigma(\mathcal{S})$, and a subset of its border $\mathsf{Y} \subseteq \mathsf{B}_{\mathcal{S}}$.

3 Generalized Connectives in Multiplicative Linear Logic

The notion of generalized (multiplicative) connectives for multiplicative linear logic was introduced since the early works on linear logic [21]. We say that an inference rule of the sequent calculus is **linear** if each occurrence of subformula (except the principal formula of the rule) occurring in the conclusion of the rule occurs exactly once in its premise(s), and it is **context-free** if no conditions on the non-principal formulas affect the application of the rule. A rule is **multiplicative** if linear and context-free.

Example 10. Consider the three rules in Equation (4) below. Only the leftmost is multiplicative: the central one is not linear since the subformula B does not occur in the premise, while the rightmost one is not context-free since the rule requires the sequent to contain an odd number of formulas.

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B, C}{\vdash \Gamma, \Delta, A \otimes (B \,\mathcal{B}\, C)} \, a \otimes (b \,\mathcal{B}\, c) \qquad \frac{\Gamma, A}{\vdash \Gamma, A \,\mathcal{B}\, B} \, \mathsf{W}_{\mathcal{B}} \qquad \frac{\vdash C_1, \dots, C_{2k-1}, A \quad \vdash C_2, \dots, C_{2k}, B}{\vdash C_1, C_2, \dots, C_{2k-1}, C_{2k}, A \otimes B} \, \mathsf{odd}_{\otimes} \quad (4)$$

In [21] the authors observe that any multiplicative rule can be fully described by a partition (having a block for each of the rule premises) keeping track of how active formulas are distributed among the premise of the rule. Thus, we can define so called **synthetic rules** (see, e.g., [21, 32, 52]), allowing us to gather multiple inference of multiplicative rules to construct a formula by a single rule application, as shown in the following example.

Example 11. Consider the following formulas, their *synthetic rules* and the associated partitions.

Formula	$F = a \otimes (b \,\mathcal{B}\, c)$	$G = (a \,\mathcal{B}\, b) \otimes (c \,\mathcal{B}\, d)$	$H = a \,\mathcal{B}\, (b \otimes c)$
Derivation(s)	$\dfrac{\Gamma, a \quad \dfrac{\Delta, b, c}{\Delta, b \,\mathcal{B}\, c}\mathcal{B}}{\Gamma, \Delta, a \otimes (b \,\mathcal{B}\, c)}\otimes$	$\dfrac{\dfrac{\Delta, a, b}{\Delta, a \,\mathcal{B}\, b}\mathcal{B} \quad \dfrac{\Delta, c, d}{\Delta, c \,\mathcal{B}\, d}\mathcal{B}}{\Delta, \Delta, (a \,\mathcal{B}\, b) \otimes (c \,\mathcal{B}\, d)}\otimes$	$\dfrac{\dfrac{\Gamma, a, b \quad \Delta, c}{\Gamma, \Delta, a, b \otimes c}\otimes}{\Gamma, \Delta, a \,\mathcal{B}\, (b \otimes c)}\mathcal{B}$ and $\dfrac{\dfrac{\Gamma, a, c \quad \Delta, b}{\Gamma, \Delta, a, b \otimes c}\otimes}{\Gamma, \Delta, a \,\mathcal{B}\, (b \otimes c)}\mathcal{B}$
Synthetic Rule(s)	$\dfrac{\vdash \Gamma, a \quad \vdash \Delta, b, c}{\vdash \Gamma, \Delta, a \otimes (b \,\mathcal{B}\, c)}\mathcal{R}_F$	$\dfrac{\vdash \Gamma, a, b \quad \vdash \Delta, c, D}{\vdash \Gamma, \Delta, (a \,\mathcal{B}\, b) \otimes (c \,\mathcal{B}\, D)}\mathcal{R}_G$	$\dfrac{\vdash \Gamma, a, b \quad \vdash \Delta, c}{\vdash \Gamma, \Delta, a \,\mathcal{B}\, (b \otimes c)}\mathcal{R}_{H_1}$ and $\dfrac{\vdash \Gamma, a, c \quad \vdash \Delta, b}{\vdash \Gamma, \Delta, a \,\mathcal{B}\, (b \otimes c)}\mathcal{R}_{H_2}$
Associated partitions	$[(1),(2,3)]$	$[(1,2),(3,4)]$	$[(1,2),(3)]$ and $[(1,3),(2)]$

Conversely, given a set of partitions P in $\mathbb{P}_n$, we can define a rule introducing an n-ary **generalized connective** C_P for each partition in P. In this case, we say that P is the **behavior** of C_P. Consider the following examples.

Behavior	$P = \big\{[(1,2),(3,4)],(1,4),(2,3)\big\}$	$Q = \big\{[(1,3),(2),(4)],[(1),(2,4),(3)]\big\}$
Rules	$\dfrac{\vdash \Gamma, A, B \quad \vdash \Delta, C, D}{\vdash \Gamma, \Delta, \mathsf{C}_P(A, B, C, D)}\,[(1,2),(3,4)]$ $\dfrac{\vdash \Gamma, A, D \quad \vdash \Delta, B, C}{\vdash \Gamma, \Delta, \mathsf{C}_P(A, B, C, D)}\,[(1,4),(2,3)]$	$\dfrac{\vdash \Gamma, A, C \quad \vdash \Delta, B \quad \vdash \Sigma, D}{\vdash \Gamma, \Delta, \Sigma, \mathsf{C}_Q(A, B, C, D)}\,[(1,3),(2),(4)]$ $\dfrac{\vdash \Gamma, A \quad \vdash \Delta, B, D \quad \vdash \Sigma, C}{\vdash \Gamma, \Delta, \Sigma, \mathsf{C}_Q(A, B, C, D)}\,[(1),(2,4),(3)]$

(5)

However, as shown in various works [21, 32, 44, 9], not all sets of partitions can be considered to be satisfactory in order to define connectives. In fact, we allow to use a set of partitions $P \subset \mathbb{P}_n$ to describe a connective only if P admits a set Q such that $P \perp Q$. This condition is mandatory to guarantee the possibly to define a dual connective whose rules well-behave with respect to cut-elimination.

In [44, 9] it has been proved that there are families of sets of partitions which can be used to describe behaviors different from any synthetic rule defined using $\otimes$ and $\mathcal{B}$ rules. Moreover, in [9] it is also shown that no satisfactory sequent calculus can be defined in presence of generalized connectives due to the lack of the so-called **initial coherence** [52, 16] (also called *packaging problem* in [21]), that is, the possibility of having a proof system in which it is always possible to prove "A implies A" using atomic axiom only.

3.1 Generalized Connectives in Multiplicative Proof Nets

Sets of partitions have been used to define generalized connectives in the proof net syntax in [21, 32, 48, 44, 9], overcoming the aforementioned problem of initial coherence. We here give some intuitions on these connectives, while more precise definitions are provided in Section 5 where we properly define the formal setting required to accommodate them.

Generalized connectives in multiplicative proof structures use sets of partitions to define the *behavior* of new connectives, that is, the way *tests* are constructed. Intuitively, the behavior of $\mathcal{B}$ (defined as $\{[(1),(2)]\}$) and $\otimes$ (defined as $\{[(1,2)]\}$) provide the topological constraints of the definition of the test: for $\mathcal{B}$ the link is replaced by a hyperedge connecting only one of the two inputs (connecting the output to one of the two blocks) while for the $\otimes$ the link is replaced by hyperedge connecting both inputs (since both belong to the same block). Similarly, in defining a test for a link with a given behavior is replaced by certain hyperedges connecting the vertices in a same block.

In this case, given an n-ary connective and a partition $P \subset \mathbb{P}_n$, the condition of the existence of a Q such that $P \perp Q$ is not enough to guarantee the existence of a dual connective well-behaving with respect

Formula	Proof Structure	Tests
$\kappa(a,b,c)$ $=$ $a \otimes (b \,⅋\, c)$	(diagram: b, c joined by $⅋$, with a, joined by $\otimes$ to $\kappa(a,b,c)$)	(diagram 1: a, b, c, $b\,⅋\,c$, $\kappa(a,b,c)$) (diagram 2: b, c, a, $b\,⅋\,c$, $\kappa(a,b,c)$)
$\kappa^{\perp}(a^{\perp},b^{\perp},c^{\perp})$ $=$ $a^{\perp} \,⅋\, (b^{\perp} \otimes c^{\perp})$	(diagram: $b^{\perp}$, $c^{\perp}$ joined by $\otimes$, with $a^{\perp}$, joined by $⅋$ to $\kappa^{\perp}(a,b,c)$)	(diagram 1: $b^{\perp}$, $c^{\perp}$, $a^{\perp}$, $b \otimes c$, $\kappa^{\perp}(a,b,c)$) (diagram 2: $b^{\perp}$, $c^{\perp}$, $a^{\perp}$, $b \otimes c$, $\kappa^{\perp}(a,b,c)$)
$\widetilde{\kappa}(a,b,c)$	(diagram: a, b, c joined by $\widetilde{\kappa}$ to $\widetilde{\kappa}(a,b,c)$)	(diagram 1: a, b, c, $\widetilde{\kappa}(a,b,c)$) (diagram 2: a, b, c, $\widetilde{\kappa}(a,b,c)$) (diagram 3: a—b, c, $\widetilde{\kappa}(a,b,c)$)

Figure 4: The behaviors of the synthetic connectives associated to the formula $F(a,b,c) = a \otimes (b \,⅋\, c)$, to its dual formula, and to a generalized multiplicative connective whose behavior strictly contains the one of $F(a,b,c)$ in such a way is the same of $F(a,b,c)$ if restricted on the inputs only.

to cut-elimination. Thus the stronger condition $P \perp\!\!\!\perp Q$ is needed[4].

Remark 12. As noticed in [44, 9], the definition of more-than-binary generalized connectives requires to include the information about which block of inputs is connected to the output of the link connective. This information is only required to ensure a sound cut-elimination procedure, and it is lost after removing cuts. Said differently, the contextual equivalence defined by cut-elimination is not able to distinguish certain behaviors differing in the way set of inputs are connected to the output. This information is not relevant for the standard MLL connectives nor for *synthetic connectives* (that is, the ones which can indirectly defined by means of combination of $\otimes$ and $⅋$; see Example 11), since it can be indirectly derived using the less complex nature of these connectives, which are defined inductively using binary ones. Nevertheless, this information is not negligible in the general case, since this information may define different tests as shown in the following example explaining in detail Figure 4.

Example 13. Consider the formula $F(a,b,c) = a \otimes (b \,⅋\, c)$ and the synthetic connectives $\kappa(a,b,c) :=$ $F(a,b,c)$ and $\kappa^{\perp}(a^{\perp},b^{\perp},c^{\perp}) := F^{\perp}(a^{\perp},b^{\perp},c^{\perp})$ respectively associated to the sets of partitions $\mathfrak{B}_{\kappa} =$ $\{[(1,2),(3)],[(1,3),(2)]\}$ and $\mathfrak{B}_{\kappa^{\perp}} = \{[(1),(2,3)]\}$ (see Figure 4).

We can now define the connective $\widetilde{\kappa}$ associated to the same set of partitions of κ (that is, the set of partitions $\mathfrak{B}_{\widetilde{\kappa}} = \mathfrak{B}_{\kappa} = \{[(1,2),(3)],[(1,3),(2)]\}$) but in which we allow an extra test which enforces no new partitions among the inputs. See the bottom-most row of the table in Figure 4, where the new test (the right-most one) enforces the partition $[(1,2),(3)] \in \mathfrak{B}_{\kappa}$ over inputs.

Since κ and $\widetilde{\kappa}$ are defined by the same set of partitions over their inputs, they are both orthogonal to $\kappa^{\perp}$. Moreover, both DR-correctproof structures of $\kappa(a,b,c) \,⅋\, \kappa^{\perp}(a^{\perp},b^{\perp},c^{\perp})$ and $\widetilde{\kappa}(a,b,c) \,⅋\,$ $\kappa^{\perp}(a^{\perp},b^{\perp},c^{\perp})$ are correct, and the result of cut-elimination of a κ- or a $\widetilde{\kappa}$-gate against a $\kappa^{\perp}$-gate reduces

---

[4]Note that in [21, 32] each multiplicative connective is defined by a pair of sets of dual partitions over the same finite set satisfying an orthogonality condition. This condition is sufficient to fully describe these connectives in sequent calculus style, and we here show that it is also sufficient for our proof net expansion paradigm. However, it is well-known that in a Curry-Howard oriented interpretation of proof-as-program paradigm stronger conditions are required in order to guarantee a sound dynamic of cut-elimination (that is, not only the two partitions must be orthogonal, but also their orthogonal sets must be so).

to a proof structure with the same behavior. Note that this implies that κ and $\widetilde{\kappa}$ are indistinguishable with respect to the notion of context equivalence usually considered on proof structures (see, e.g., [33, 25, 27]).

4 Logic Programming with Multiplicative Structures

In this section we recall the results from [14, 15] (restricted to the multiplicative linear logic fragment) on the possibility to define a logic programming framework based on proof net construction.

    The classical interpretation of logic programming (see, e.g., [12, 14, 52]), a program is defined by a set of sequent calculus rules and its execution is conceived as the process of expanding the open branches of the derivation tree of a given formula. This correspondence can easily be extended using synthetic (linear) inference rules as the ones from Example 11 to define the following methods:

$$\mathbf{F} :- \mathbf{a}, (\mathbf{b} \,\mathscr{V}\, \mathbf{c}) \quad \Big| \quad \mathbf{G} :- (\mathbf{a} \,\mathscr{V}\, \mathbf{b}), (\mathbf{c} \,\mathscr{V}\, \mathbf{d}) \quad \Big| \quad \mathbf{H_1} :- (\mathbf{a} \,\mathscr{V}\, \mathbf{b}), \mathbf{c} \qquad \mathbf{H_2} :- (\mathbf{a} \,\mathscr{V}\, \mathbf{c}), \mathbf{b} \tag{6}$$

    In particular, a specific family of formulas (called *bipoles*) can be used to define methods.

Definition 14 ([14])**.** Given a set of **negative** atoms $\mathcal{A}$ whose negations are **positive**, a **monopole** is a disjunction ($\mathscr{V}$) of negative atoms. A **bipole** is a conjunction ($\otimes$) of monopoles and positive atoms which contains at least one positive atom. Given a set of bipoles $\mathfrak{F}$, the **focussing bipolar sequent calculus** $[\mathfrak{F}, \mathcal{A}]$ is given by the set of inference rules of the following form, where F is a bipole in $\mathfrak{F}$.

$$\frac{\vdash \mathsf{i}_{1,1}, \dots, \mathsf{i}_{1,k_1} \qquad \cdots \qquad \vdash \mathsf{i}_{i,1}, \dots, \mathsf{i}_{i,k_i}}{\vdash \mathsf{o}_1, \dots, \mathsf{o}_n} \quad F = \mathsf{o}_1^{\perp} \otimes \cdots \otimes \mathsf{o}_n^{\perp} \otimes (\mathsf{i}_{1,1} \,\mathscr{V}\, \cdots \mathscr{V}\, \mathsf{i}_{1,k_1}) \otimes \cdots \otimes (\mathsf{i}_{i,1} \,\mathscr{V}\, \cdots \mathscr{V}\, \mathsf{i}_{i,k_i}) \tag{7}$$

    As shown in [12], a bipole F can be seen as a logic programming method having as **head** (or **trigger**) the subformula containing the positive atoms of F (a conjunction), and as **body** the subformula containing the negative atoms F (a CNF formula). Intuitively, each bipole F induces a synthetic rule with principal formula F and and whose active formulas are its positive atoms gathered in a same premise if they belong to the same conjunct. By means of example the rule for the bipole F in Equation (7) can be seen as a synthetic rule introducing the formula F corresponding to the following derivation

$$\frac{\dfrac{\dfrac{\mathsf{i}_{1,1}, \dots, \mathsf{i}_{1,k_1}}{\mathsf{i}_{1,1} \,\mathscr{V}\, \cdots \mathscr{V}\, \mathsf{i}_{1,k_1}} \,\mathscr{V} \quad \cdots \quad \dfrac{\mathsf{i}_{i,1}, \dots, \mathsf{i}_{i,k_i}}{\mathsf{i}_{i,1} \,\mathscr{V}\, \cdots \mathscr{V}\, \mathsf{i}_{i,k_i}} \,\mathscr{V}}{(\mathsf{i}_{1,1} \,\mathscr{V}\, \cdots \mathscr{V}\, \mathsf{i}_{1,k_1}) \otimes \cdots \otimes (\mathsf{i}_{i,1} \,\mathscr{V}\, \cdots \mathscr{V}\, \mathsf{i}_{i,k_i})} \otimes \quad \dfrac{\dfrac{}{\vdash \mathsf{o}_1^{\perp}, \mathsf{o}_1} \mathsf{ax} \quad \cdots \quad \dfrac{}{\vdash \mathsf{o}_n^{\perp}, \mathsf{o}_n} \mathsf{ax}}{\mathsf{o}_1^{\perp} \otimes \cdots \otimes \mathsf{o}_n^{\perp}, \mathsf{o}_1, \cdots, \mathsf{o}_n} \otimes}{\mathsf{o}_1^{\perp} \otimes \cdots \otimes \mathsf{o}_n^{\perp} \otimes (\mathsf{i}_{1,1} \,\mathscr{V}\, \cdots \mathscr{V}\, \mathsf{i}_{1,k_1}) \otimes \cdots \otimes (\mathsf{i}_{i,1} \,\mathscr{V}\, \cdots \mathscr{V}\, \mathsf{i}_{i,k_i}), \mathsf{o}_1, \cdots, \mathsf{o}_n} \otimes \tag{8}$$

In [13] it has been proved that the focussing bipolar sequent system with one rule for each MLL bipole is sound and complete with respect to MLL.

4.1 Bipolar Proof Nets

The idea of using the focussing bipolar sequent calculus has been further developed in [15], where the authors proposed to model such a framework using proof nets construction instead of proof search in sequent calculi. The main advantage of the graphical syntax with respect to the bipolar sequent calculus is that in this latter, even if this rule admits a non-singleton trigger, a rule can expand only a single branch of a derivation. In fact, the tree-like structure of sequent calculus syntax allows us to expand one leaf of the derivation tree at a time by applying a rule. For instance, consider Figure 5 where the concurrent
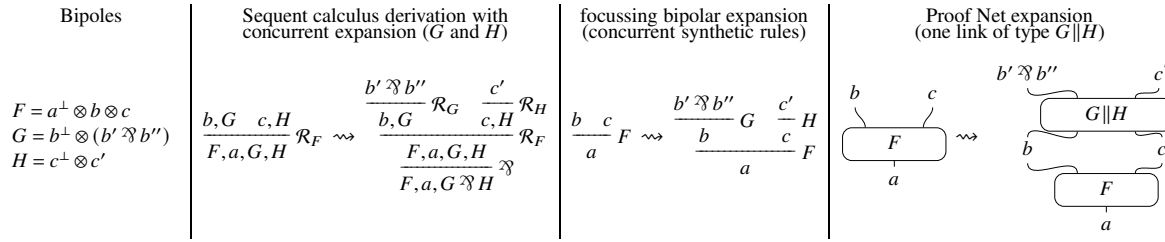
Figure 5: A concurrent application of the bipoles G and H after F represented in different formalisms

application of two methods on two different branches of a derivation can be represented by an expansion of a proof structure with a single link.

More precisely, we can use bipoles to define new link types in the same manner a bipole defines a new synthetic inference rule in the sequent calculus. Each test replaces such a link with one of the possible tests of the proof structure representing the bipole. By means of example, the bipolar rule from Equation (7) could be used to define the link below on the left, and tests would replace such a link with any test of the proof structure below on the right, which represents the open derivation in Equation (8).

$$(9)$$
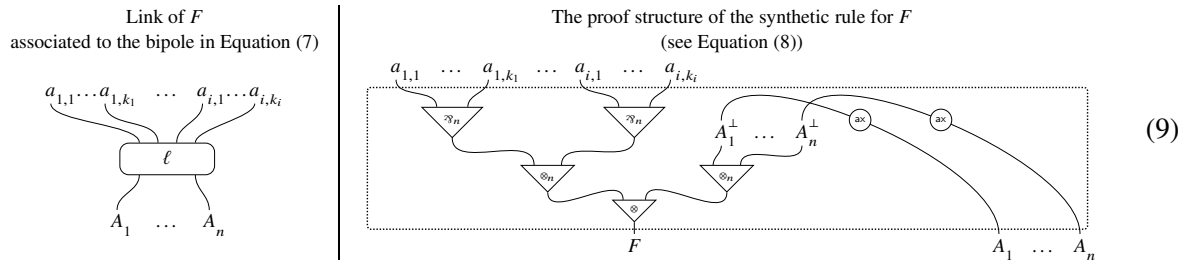
Note that the link above on the left has outputs $o_1, \ldots, o_n$, while the proof structure on the right has an additional output F. In the next section we provide a solution to address this mismatch (see no-output links in Definition 16), but it is worth noting that we can define links representing concurrent application of bipoles by simply connecting those additional outputs via a $\mathfrak{N}$-link. Analyzing the shape of the proof structure describing a *concurrent bipole*.

Definition 15. We introduce the following naming for specific proof structure (see examples in Figure 6):

- **body**: a $\otimes_n$-link collecting the outputs of $\mathfrak{N}_n$-links (i.e., the proof structure of a CNF-formula). The body gathers the clauses corresponding to the body of a method;

- **header**: a bundle of ax-links attached to a $\otimes_n$-link by exactly one of their two outputs each. The header gathers the outputs corresponding to the head of a method;

- **synchronizer**: a $\mathfrak{N}_n$-link collecting the outputs of $\otimes$-links (i.e., the proof structure of a DNF-formula). The synchronizer establishes a connection between headers of methods and their bodies.

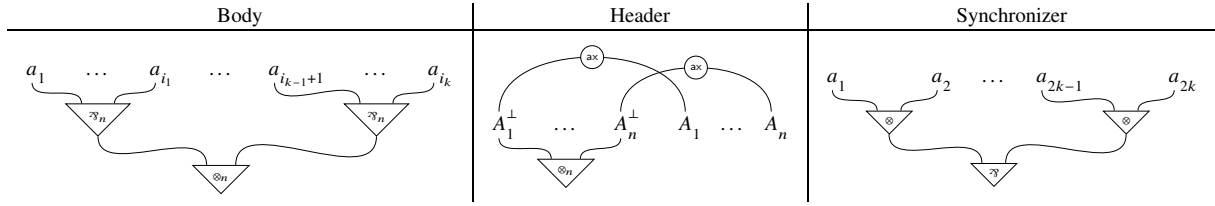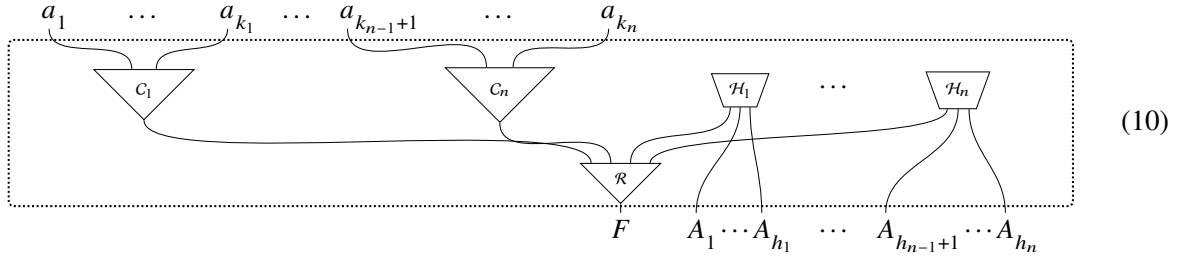A **concurrent bipole** is a proof structure made of a synchronizer whose inputs are attached to headers

Body	Header	Synchronizer

Figure 6: Components of a concurrent bipolar link.

and bodies. That is, a proof structure of the following shape:

$$(10)$$

where $\mathcal{H}_1, \ldots, \mathcal{H}_n$ are headers, $C_1, \ldots, C_n$ are bodies, and $\mathcal{R}$ is a synchronizer.

5 Generalizing Multiplicative Proof Structures

In this section we provide a general setting to define hypergraphs with hyperedges labeled by sets of partitions generalizing the syntax of proof structures, allowing us to accommodate generalized connectives, generalize the DR-correct, and define the notion of a *component* as a "proof structure which may be a piece of a proof net".

Definition 16. A **link type** (or simply **type**) is a triple $\langle n, m, \mathfrak{B} \rangle$ given by two natural numbers $n, m \in \mathbb{N}$ and a **behavior** $\mathfrak{B} \subseteq \mathbb{P}_{\{i_1, \ldots, i_n, o_1, \ldots, o_m\}}$. We define the following link types:

$$\mathsf{ax} = \langle 0, 2, \{[(o_1, o_2)]\} \rangle \quad \otimes = \langle 2, 1, \{[(i_1, i_2, o_1)]\} \rangle \quad \mathfrak{N} = \langle 2, 1, \{[(i_1, o_1), (i_2)], [(i_1), (i_2, o_1)]\} \rangle$$

$$\mathsf{cut} = \langle 2, 0, [(o_1, o_2)] \rangle \quad \mathfrak{N}_n^\bullet = \langle n, 0, [(o_1), \ldots, (o_n)] \rangle \quad \otimes_n^\bullet = \langle n, 0, [(o_1, \ldots, o_n)] \rangle$$

$$\otimes_n = \langle n, 1, \{[(i_1, \ldots, i_n, o_1)]\} \rangle \quad \mathfrak{N}_n = \left\langle n, 1, \{[(i_1), \ldots, (i_{k-1}), (i_k, o_1), (i_{k+1}), \ldots, (i_n)]\}_{k \in \{1, \ldots, n\}} \right\rangle$$

Remark 17. By definition, $\otimes_2 = \otimes$ and $\mathfrak{N}_2 = \mathfrak{N}$ and $\mathsf{cut} = \otimes_2^\bullet$. The type $\otimes_1 = \mathfrak{N}_1$ can be thought as an edge connecting the input with the output since they both have behavior $[(i_1, o_1)]$. The type $\otimes_1^\bullet = \mathfrak{N}_1^\bullet$ can be thought as a "dead-end" hyperedge with one input and no output (their behavior is $[(i_1)]$).

Definition 18. A **multiplicative structure** over a signature Σ is a linear hypergraph $\mathcal{H}$ such that each hyperedge ℓ is labelled with a $\langle n, m, \mathfrak{B} \rangle \in \Sigma$ such that $|\mathsf{in}(h)| = n$ and $|\mathsf{out}(h)| = m$.

In drawing multiplicative structures, we label hyperedges by the corresponding type. The definition of **sub-multiplicative structure**, as well as the definition of **sequential** and **parallel composition** of multiplicative structures are defined extending the ones for hypergraphs.

In order to extend the notion of DR-correct, we need to provide a way to define tests.

Definition 19. Let $\mathcal{S} = \langle V_\mathcal{G}, E_\mathcal{S} \rangle$ be a multiplicative structure. A **switching** for $\mathcal{S}$ is a map σ assigning to each link ℓ a single partition $\sigma(\ell) \in \mathfrak{B}_\ell$. We denote by $\Sigma(\mathcal{S})$ the set of all possible switchings for $\mathcal{S}$. The
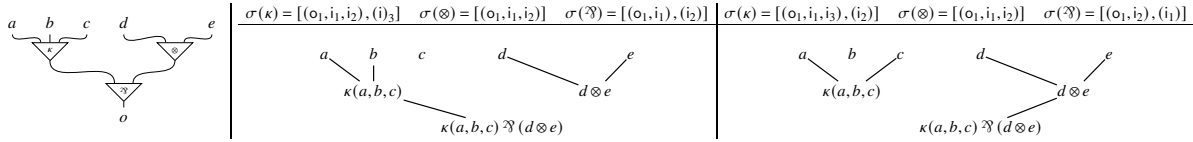
Figure 7: A proof structure where $\kappa = \langle 3, 1, \{[(o_1, i_1, i_2), (i)_3], [(o_1, i_1, i_3), (i_2)]\}\rangle$ and two of its test

test $\sigma(\mathcal{S})$ defined by the switching σ is the undirected hypergraph obtained by replacing each link in $E_{\mathcal{S}}$ with one undirected hyperedge for each block in $\sigma(\ell)$. Formally, $\sigma(\mathcal{S}) = \langle V_{\mathcal{S}}, \bigcup_{\ell \in E_{\mathcal{S}}} \{\gamma \mid \gamma \in \sigma(\ell)\}\rangle$.

The **behavior** of a test σ is the partition p_σ of the border of $\mathcal{S}$ defined by the connected components of $\sigma(\mathcal{S})$. That is, $x, y \in B_{\mathcal{S}}$ belongs to the same block $\gamma \in p_\sigma$ iff the vertices x and y are connected of $\sigma(\mathcal{S})$. The **behavior** of a multiplicative structure $\mathcal{S}$ is defined as the set of behaviors of its tests, that is, $\mathfrak{B}_{\mathcal{S}} = \{p_\sigma \in \mathbb{P}_{B_{\mathcal{S}}} \mid \sigma \in \Sigma(\mathcal{S})\}$.

The behavior of a multiplicative structure is the collection of the information on how a multiplicative structure interacts with any possible context. It keeps track of the connectivity the vertices in its border in all its tests (see an example in Figure 7).

Definition 20. Let $\mathcal{S}$ be a multiplicative structure. We say that $\mathcal{S}$ is **correct** if $\sigma(\mathcal{S})$ is connected and acyclic for any test $\sigma \in \Sigma(\mathcal{S})$. It is **multiplicative net** if correct and if $\mathcal{S}$ has no inputs and at least one output. If each test $\sigma(\mathcal{S})$ of $\mathcal{S}$ is acyclic and each of its vertices is connected to a vertex of the border, then we say that $\mathcal{S}$ is a **(multiplicative) component**. A **transitory** component (or T-**component**) is a component such that each input admits a test where it is connected to an output. A **module** $\mathcal{M}$ is a connected non-empty multiplicative structure such that $\mathcal{M} \subset \mathcal{S}$ for a multiplicative net $\mathcal{S}$.

Example 21. Consider the examples in Figure 3. The multiplicative structure $\mathcal{S}_1$ is a multiplicative net (therefore a T-component). $\mathcal{S}_2$ is a component and a module, but not a T-component (it has no outputs). The multiplicative structure $\mathcal{S}_3$ is not a component (it admits a test in which a_2 and a_3 are disconnected from any other vertex) nor a module (there is no multiplicative net containing $\mathcal{S}_3$ as a sub-multiplicative structure).

Theorem 22. *All modules are components.*

Proof. If $\mathcal{M}$ is a module, then each test $\sigma(\mathcal{M})$ is acyclic, otherwise there should be a cycle in a test of $\mathcal{S}$. Moreover, as consequence of the fact that $\mathcal{S}$ is connected, no sub-multiplicative structure $\mathcal{S}'$ of $\mathcal{S}$ such that $B_{\mathcal{S}'} = \emptyset$. Therefore each vertex in $\mathcal{M}$ must be connected to a vertex in $B_{\mathcal{M}}$ in each test $\sigma(\mathcal{M})$ otherwise $\mathcal{S}$ would not be connected. $\qquad\square$

Definition 23. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be components and $X \subseteq (I_{\mathcal{M}_1} \cap O_{\mathcal{M}_2})$ non-empty. We say that $\mathcal{M}_2$ **expands** $\mathcal{M}_1$ (on X) if $\mathcal{M}_1 \triangleleft_X \mathcal{M}_2$ is a component.

Theorem 24. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be components and $X \subseteq (I_{\mathcal{M}_1} \cap O_{\mathcal{M}_2})$ non-empty. Then*

$$\begin{array}{c} \mathcal{M}_2 \text{ expands } \mathcal{M}_1 \\ \text{on } X \end{array} \iff \left\{ \begin{array}{l} X \neq B_{\mathcal{M}_1} \cup B_{\mathcal{M}_2} \\ \mathfrak{B}_{\mathcal{M}_1|_X} \perp_w \mathfrak{B}_{\mathcal{M}_2|_X} \\ \text{each } x \in X \text{ is connected either to a } y \in (B_{\mathcal{M}_1} \setminus X) \text{ in each test of } \mathcal{M}_1, \\ \hspace{4cm} \text{or to a } z \in (B_{\mathcal{M}_2} \setminus X) \text{ in each test of } \mathcal{M}_2 \end{array} \right.$$

Proof. By definition of component, letting $\mathcal{M} = \mathcal{M}_1 \triangleleft_X \mathcal{M}_2$.

($\Rightarrow$) If $\mathcal{M}$ is a component, then $\mathsf{B}_\mathcal{M} = (\mathsf{B}_{\mathcal{M}_1} \cup \mathsf{B}_{\mathcal{M}_2}) \setminus X$ must be non-empty, thus $X \neq (\mathsf{B}_{\mathcal{M}_1} \cup \mathsf{B}_{\mathcal{M}_2})$.

If we assume that $\mathfrak{B}_{\mathcal{M}_1|_X} \not\perp_w \mathfrak{B}_{\mathcal{M}_2|_X}$, then there are $x, y \in X$ such that $\{x, y\} \subset \gamma_1 \in p \in \mathfrak{B}_{\mathcal{M}_1|_X}$ and $\{x, y\} \subset \gamma_2 \in q \in \mathfrak{B}_{\mathcal{M}_2|_X}$. That is, there is a path between x and y in both $\sigma_1(\mathcal{M}_1)$ and $\sigma_2(\mathcal{M}_2)$ with $\sigma_i \in \Sigma(\mathcal{M}_i)$ and $i \in \{1,2\}$. Thus there is a switch $\sigma \in \Sigma(\mathcal{M})$ such that $\sigma(\mathcal{M})$ contains a cycle. This contradicts the hypothesis of $\mathcal{M}$ being a module.

Finally, since each vertex of $\mathcal{M}$ is connected to a vertex in $\mathsf{B}_\mathcal{M} = (\mathsf{B}_{\mathcal{M}_1} \cup \mathsf{B}_{\mathcal{M}_2}) \setminus X$ in each test, then, each $x \in X$ is.

($\Leftarrow$) Since $\mathcal{M}_1$ and $\mathcal{M}_2$ are components, then $\sigma(\mathcal{M}_1)$ and $\sigma(\mathcal{M}_2)$ does not contain cycles (where, for both $i \in \{1,2\}$. Let us denote by $\sigma(\mathcal{M}_i)$ the test defined from $\mathcal{M}_i$ by the switch obtained by restringing $\sigma \in \Sigma(\mathcal{M})$ to $\mathcal{M}_i$). Then, if $\sigma(\mathcal{M})$ contains a cycle, then there are $x, y \in X$ such that x and y are connected in $\sigma(\mathcal{M}_1)$ and in $\sigma(\mathcal{M}_2)$. This implies the existence of a $\gamma_1 \in p \in \mathfrak{B}_{\mathcal{M}_1|_X}$ and a $\gamma_2 \in q \in \mathfrak{B}_{\mathcal{M}_2|_X}$ with γ_1 and γ_2 both containing x and y, therefore $p \not\perp_w q$, contradicting the hypothesis.

The fact that each vertex in $\mathcal{M}$ is connected to a vertex of the border in any test is consequence of the fact that each vertex in $\mathcal{M}$ (then, in particular, each $x \in X$) is connected to a vertex in $\mathsf{B}_\mathcal{M} = (\mathsf{B}_{\mathcal{M}_1} \setminus X) \cup (\mathsf{B}_{\mathcal{M}_2} \setminus X)$. □

Corollary 25. *Let $\mathcal{M}$ be a component, and $\mathcal{T}$ be T-component and $(\mathsf{I}_\mathcal{M} \cap \mathsf{O}_\mathcal{T}) \supseteq X \neq \emptyset$. If $\mathcal{M}$ is a transitory, then $\mathcal{T} \triangleleft_X \mathcal{M}$ is so.*

Proof. In light of Theorem 24, it suffices to remark that if $\mathcal{M}$ is a T-component, then every input $\mathsf{i} \in \mathsf{I}_\mathcal{M}$ is connected to an output $\mathsf{o} \in \mathsf{O}_\mathcal{M}$ in a test $\sigma_2(\mathcal{M})$. If $\mathsf{o} \in \mathsf{O}_\mathcal{M}$ we conclude. Otherwise $\mathsf{o} \notin \mathsf{O}_\mathcal{M}$ and $\mathsf{o} = x \in X$. Since $\mathcal{T}$ is T-component, then by definition there is a test $\sigma_1(\mathcal{T})$ in which x is connected to a vertex $\mathsf{o} \in \mathsf{O}_\mathcal{T}$. We conclude since each input of $\mathcal{T} \triangleleft_X \mathcal{M}$ is either an input of $\mathcal{T}$, or an input of $\mathcal{M}$; and in the latter case we have a switching σ defined as the union of σ_1 and σ_2 such that i is connected to an output $\mathsf{o} \in \mathsf{O}_\mathcal{T} \subseteq \mathsf{O}_{\mathcal{T} \triangleleft_X \mathcal{M}}$. □

Proposition 26. *We can check if a component $\mathcal{M}'$ expands a component $\mathcal{M}$ in polynomial time with respect to $|\mathsf{O}_{\mathcal{M}'}| + |\mathfrak{B}_{\mathcal{M}'}| + |\mathfrak{B}_\mathcal{M}|$.*

Proof. To check if $\mathfrak{B}_{\mathcal{M}'} \perp_w \mathfrak{B}_{\mathcal{M}|_{\mathsf{O}_\ell}}$ requires $|\mathfrak{B}_{\mathcal{M}'}| \times |(\mathfrak{B}_\mathcal{M})|_{\mathsf{O}_{\mathcal{M}'}}| \leq |\mathfrak{B}_{\mathcal{M}'}| \times |\mathfrak{B}_\mathcal{M}|$ orthogonality tests on partitions. Each test requires to build the graph $\mathcal{G}(p,q)$ (linear on $|\mathsf{O}_\ell|$) and check graph acyclicity since graph traversal is linear in $|V_{\mathcal{G}(p,q)}| + |\overset{\mathcal{G}(p,q)}{\frown}| \leq 2|\mathsf{O}_\ell|$ and $|\overset{\mathcal{G}(p,q)}{\frown}| = |\mathsf{O}_\ell|$, see [20]. □

6 Modelling behaviors beyond the scope of MLL-proof structures

In this section we recall the definition of two classes of generalized multiplicative connectives from [9], showing how the corresponding links can be used to define methods whose behaviors exhibit unexpected context-sensitive and non-linear characteristics in a multiplicative setting.

Definition 27. Let $n = uv$ be the product of two prime numbers $u, v \in \mathbb{N}$. We define a **basic partition with u blocks of v elements** to be a partition $p \in \mathbb{P}_n$ such that each block $\gamma \in p$ is either of the form $\gamma = (i, i+1, \ldots, i+v-1)$ if $i+v < n$, or of the form $\gamma = (i, \ldots, n, 1, 2, \ldots, i+v-(n+1))$ for a $i \in \{1, \ldots, n\}$ otherwise. We denote by $\mathfrak{B}_{\langle u,v \rangle} \subset \mathbb{P}_n$ the set of basic partitions with u blocks of v elements and $\mathfrak{B}^\perp_{\langle u,v \rangle}$ its orthogonal set of partitions.

$\mathsf{G}^{\perp}_{2,2}$	$\mathsf{G}_{2,2}$
$\{[(o,i_1,i_2),(i_3,i_4)],[(o,i_3,i_4),(i_1,i_2)]\}$ $\cup$ $\{[(o,i_1,i_4),(i_2,i_3)],[(o,i_2,i_3),(i_1,i_4)]\}$	$\{[(o,i_1,i_3),(i_2),(i_4)],[(o,i_2,i_4),(i_1),(i_3)],[(o,i_1,i_4),(i_2),(i_3)],[(o,i_2,i_3),(i_1),(i_4)]\}$ $\cap$ $\{[(o,i_1,i_3),(i_2),(i_4)],[(o,i_2,i_4),(i_1),(i_3)],[(o,i_1,i_2),(i_3),(i_4)],[(o,i_3,i_4),(i_1),(i_2)]\}$ $=$ $\{[(o,i_1,i_3),(i_2),(i_4)],[(o,i_2,i_4),(i_1),(i_3)]\}$

Figure 8: Connective $\mathsf{G}^{\perp}_{2,2}$ and its dual connective $\mathsf{G}_{2,2}$ interpreted as the union of the behaviors of DNF formula-trees and as the intersection of the behaviors of CNF formula-trees respectively.

For every $n = uv$ we define the following sets of partitions of the set $\{0,\dots,n\}$:

$$\mathfrak{G}_{\langle u,v\rangle}(0,1,\dots,n) = \bigcup_{k=1}^{u}\left\{p_k = \left[\gamma_{p_k},(i_1),\dots,(i_{n-u})\right] \mid p_{k|_{\{1,\dots,n\}}} \in \mathfrak{B}^{\perp}_{\langle u,v\rangle} \text{ and } 0 \in \gamma_{p_k}\right\}$$

$$\mathfrak{G}^{\perp}_{\langle u,v\rangle}(0,1,\dots,n) = \bigcup_{k=1}^{u}\bigcup_{j=1}^{v}\left\{p_k^j = \left[\gamma_1,\dots,\gamma_v\right] \mid p_{k|_{\{1,\dots,n\}}}^{j} \in \mathfrak{B}_{\langle u,v\rangle} \text{ and } 0 \in \gamma_j\right\}$$

and we define the following **Girard's types**: $\mathsf{G}_{u,v} := \left\langle uv, 1, \mathfrak{G}_{\langle u,v\rangle}\right\rangle$ and $\mathsf{G}^{\perp}_{u,v} := \left\langle uv, 1, \mathfrak{G}^{\perp}_{\langle u,v\rangle}\right\rangle$. [5]

Remark 28. The behavior $\mathfrak{G}_{\langle u,v\rangle}$ is the same of the intersection of the behavior of specific DNF-formulas, while $\mathfrak{G}^{\perp}_{\langle u,v\rangle}$ is the same of the union of behavior of specific CNF-formulas (see Figure 8). More precisely,

$$\mathfrak{G}_{\langle u,v\rangle} = \bigcap_{\tau\in\mathsf{Cy}_n} \mathfrak{B}_{\mathsf{DNF}(i_{\tau(1)},\dots,i_{\tau(n)})} \qquad \text{and} \qquad \mathfrak{G}^{\perp}_{\langle u,v\rangle} = \bigcup_{\tau\in\mathsf{Cy}_n} \mathfrak{B}_{\mathsf{CNF}(i_{\tau(1)},\dots,i_{\tau(n)})} \qquad \text{where}$$

- $\mathfrak{B}_{\mathsf{DNF}(1,\dots,n)}$ be the behavior of the multiplicative structure representing the formula tree of the disjunctive normal form formula $\mathsf{DNF}(1,\dots,n) = \left(a_1 \otimes \cdots \otimes a_v\right)\,\mathparagraph\,\cdots\,\mathparagraph\,\left(a_{n-v+1} \otimes \cdots \otimes a_n\right)$;

- $\mathfrak{B}_{\mathsf{CNF}(1,\dots,n)}$ be the behavior of the multiplicative structure representing the formula tree of the conjunctive normal form formula $\mathsf{CNF}(a_1,\dots,a_n) = \left(a_1\,\mathparagraph\,\cdots\,\mathparagraph\,a_v\right) \otimes \cdots \otimes \left(a_{n-v+1}\,\mathparagraph\,\cdots\,\mathparagraph\,a_n\right)$;

- Cy_n be the set of cyclic permutations over the set $\{1,\dots,n\}$ (assuming the standard order on $\mathbb{N}$).

Theorem 29 ([9])**.** *There is no multiplicative structure $\mathcal{S}$ over the signature $\{\mathparagraph,\otimes\}$ such that $\mathfrak{B}_\mathcal{S} = \mathfrak{B}_{\mathsf{G}_{u,v}}$ or $\mathfrak{B}_\mathcal{S} = \mathfrak{B}_{\mathsf{G}^{\perp}_{u,v}}$ for any $u, v \in \mathbb{N}$ prime numbers.*

Definition 30. We generalize the components of bipoles (see Definition 15) as follows:

- A **generalized body** is a component made of a $\otimes_n$ collecting the outputs of a multiplicative structure representing a CNF-formula (i.e., bodies) or $\mathfrak{G}^{\perp}_{\langle u,v\rangle}$-links.

- A **generalized synchronizer** is a component made of a $\mathparagraph_n^{\bullet}$ collecting the outputs of a multiplicative structure representing a DNF-formula (i.e., synchronizers) or $\mathfrak{G}_{\langle u,2\rangle}$-links.

A **generalized bipole** is a component made of a generalized synchronizer $\mathcal{R}$ collecting the outputs of certain generalized bodies $C_1,\dots,C_n$ and headers $\mathcal{H}_1,\dots,\mathcal{H}_n$ whose structure is similar to the one in Equation (10), but where no output F occurs (thanks to the $\mathparagraph_n^{\bullet}$ in the generalized synchronizer).

---

[5]In [9] it has been shown that there is no module $\mathcal{M}$ containing only $\mathparagraph$- and $\otimes$-link such that $\mathfrak{B}_\mathcal{M} = \mathfrak{B}_{\mathsf{G}_{u,v}}$ or $\mathfrak{B}_\mathcal{M} = \mathfrak{B}_{\mathsf{G}^{\perp}_{u,v}}$.
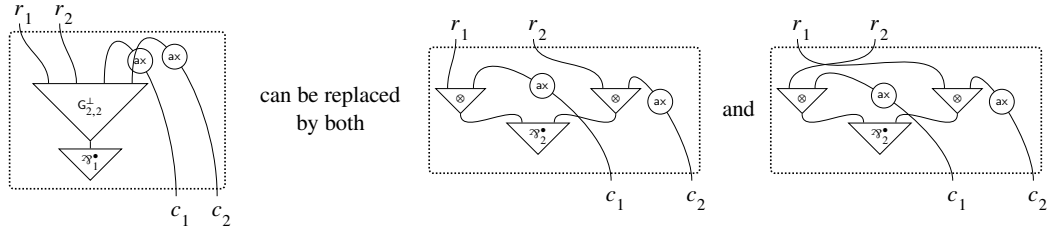
Figure 9: A multiplicative structure representing a non-deterministic application of two concurrent methods.
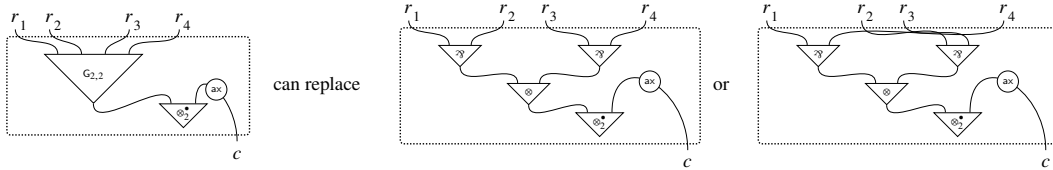

Figure 10: A link representing a method application with a dependent choice.

Remark 31. Headers and Generalized bodies and synchronizes are modules.

The following result is a corollary of Theorem 29.

Corollary 32. *There are generalized bipoles whose behavior is different from any behavior of a* MLL-*proof structure.*

Remark 33. In [15] the bipolar links have an additional output (the one labeled by the formula) as shown in Equation (9) because the syntax from [15] is closer to the representation of methods we use in Equation (8), where the name of the applied method (the formula F identifying the rule) occurs in the final sequent. In our syntax this superfluous information is discarded in the same way as in Equation (7), where the formula F does not occur in the conclusion. That is, the additional output F in Equation (10) would not occur in our generalized bipoles because in the definition of generalized synchronizes we use $\mathfrak{N}^{\bullet}_n$ instead of $\mathfrak{N}_n$, allowing us to formally discard this output.

We conclude this section by providing two toy-examples describing the way a server manages access requests to critical sections. The naïve idea behind these models is that if the vertex corresponding to a client is connected to the vertex corresponding to a resource, then there is a configuration of the model in which only that specific client accesses the resource.

Example 34 (basic union link). Consider a server receiving a request from two different clients c_1 and c_2 to access, at the same time, to one resource r_1 or r_2 in a critical section. In this case the server can execute four different methods of the form $\mathbf{r_i} : - \mathbf{c_j}$ each of which represents the resource r_i being allocated to the client c_j (for some $i, j \in \{1, 2\}$). Once any one of these methods is executed, the condition of critical section requires that no other user can access to this resource (until it is released). Therefore, either the server authorizes c_1 to access r_1, and authorizes c_2 to access r_2, or the server authorizes c_1 to access r_2, and authorizes c_2 to access r_1.

In both cases, the two methods representing the clients accessing the resources can be applied concurrently, and the multiplicative structures on the right-hand side of Figure 9 represent these two configurations. Note that none of the two multiplicative structures fully capture the described configuration: each solution makes a choice about which client has access to which resource since, and this kind of choices are beyond the scope of multiplicative linear logic. Using the basic union link $G^{\perp}_{2,2}$ we can define

the multiplicative structure in the left-hand side of Figure 9, whose behavior is the same of the union of the two multiplicative structures describing the two possible choices (see Remark 28). That is, this the $\mathsf{G}_{2,2}^\perp$-link can be interpreted as a synchronizer allowing such a concurrent choice.

Example 35 (basic intersection link)**.** Consider a server receiving a request from a single client c to access the set of resources $\{r_1, r_2, r_3, r_4\}$ with goal of either collect r_1 and r_3 (that is, to apply the method $\mathbf{c} :- \mathbf{r_1, r_3}$), or to collect r_2 and r_4 (that is, to apply the method $\mathbf{c} :- \mathbf{r_2, r_4}$). Because of our constraints, the server can either grant access to a resource in $\{r_1, r_2\}$ and one in $\{r_3, r_4\}$, or grant access to a resource in $\{r_1, r_4\}$ and one in $\{r_2, r_3\}$. These two different solutions are represented by the multiplicative structures in the right-hand side of Figure 10. However, if we consider singularly each multiplicative structure, it models more permissive configurations. For example, the right-most multiplicative structure admits a test in which c is connected to r_1 and r_4, which is not a configuration we want to allow. Indeed, the configurations we want to allow are exactly the configurations which are valid in both multiplicative structures. However, using the basic intersection link $\mathsf{G}_{2,2}$ we can define the multiplicative structure in the left of Figure 10, whose behavior is the same of the intersection of the two multiplicative structures on the right-hand side: the client c can only access at the same time to either r_1 and r_3, or r_2 and r_4.

7   Conclusion

In this paper, we extended the multiplicative fragment of the logic programming framework studied in [13, 14, 15, 29, 35]. Within this framework, as main novelty, we offered a computational interpretation of the generalized connectives discussed in [45, 9]. These connectives were initially introduced in early works on linear logic, but prior to this work, they had not been given a concrete computational interpretation: they cannot be expressed using combinations of the multiplicative connectives $\mathscr{F}$ and $\otimes$ and they describe "locally additive" behaviors [42, 2] such as non-deterministic or conditional choices. It is worth noting that the methodology used to define our framework appears to align with the definition of the basic building block of the *transcendental syntax* [33, 27] and its extensions [26, 25, 28].

Future works. As observed in Remark 28, the behavior of a basic union link can be seen as the union of behaviors of bodies (see Figure 8). This allows us to replace any basic union link within the multiplicative structure of any of these bodies, and preserving correctness. Leveraging this intuition, we could define a non-deterministic *expansion* operation on basic union link, returning their set of bodies. This operation can be further extended to multiplicative structures, providing a notion of expansion for multiplicative structures similar to the concept of *Taylor expansion* in *differential linear logic* [24]. Such an expansion, could be interpreted as representing specific instances of *delayed choice* [18]: during proof (net) construction we do not need to specify which of the possible bodies we want to use in a specific step, but we can use a basic union link containing it instead, delaying this decision. We also envision an extension of our model where links have probabilistic distributions on the set of switchings. In this setting basic union links could be equipped with probability distribution functions, transforming the non-deterministic expansion operator into a probabilistic one. Consequently, multiplicative structures could be employed to model Bayesian networks [54, 46]. Additionally, we foresee the possibility of defining refinements of the *attack trees* syntax with a linear treatment of resources but including specific non-deterministic choices [49, 39, 53]. Similarly, the linear constraints on the hypergraphs used in our model allow us to define a concurrent computational model with a more granular management of resource consumption, akin to what we experience in the management of critical sections in concurrent systems. Another possible direction is to study modules for proof structures built using the graphical connectives from [7, 8, 6, 3] to provide them with a computational meaning based on resources separation.

References

[1] V. Michele Abrusci & Roberto Maieli (2019): *Proof nets for multiplicative cyclic linear logic and Lambek calculus*. Mathematical Structures in Computer Science 29(6), p. 733–762, doi:10.1017/S0960129518000300.

[2] Vito Michele Abrusci & Roberto Maieli (2016): *Cyclic Multiplicative-Additive Proof Nets of Linear Logic with an Application to Language Parsing*. In Annie Foret, Glyn Morrill, Reinhard Muskens, Rainer Osswald & Sylvain Pogodalla, editors: *Formal Grammar*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 43–59, doi:10.1007/978-3-662-53042-9_3.

[3] Matteo Acclavio (2024): *Graphical Proof Theory I: Sequent Systems on Undirected Graphs*. arXiv:2305.12975.

[4] Matteo Acclavio, Gianluca Curzi & Giulio Guerrieri (2023): *Infinitary cut-elimination via finite approximations (extended version)*. arXiv:2308.07789.

[5] Matteo Acclavio, Gianluca Curzi & Giulio Guerrieri (2024): *Infinitary Cut-Elimination via Finite Approximations*. In Aniello Murano & Alexandra Silva, editors: *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024), Leibniz International Proceedings in Informatics (LIPIcs)* 288, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 8:1–8:19, doi:10.4230/LIPIcs.CSL.2024.8. Available at https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2024.8.

[6] Matteo Acclavio, Ross Horne, Sjouke Mauw & Lutz Straßburger (2022): *A graphical proof theory of logical time*. In: *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[7] Matteo Acclavio, Ross Horne & Lutz Straßburger (2020): *Logic beyond formulas: A proof system on graphs*. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pp. 38–52.

[8] Matteo Acclavio, Ross Horne & Lutz Straßburger (2022): *An Analytic Propositional Proof System on Graphs*. Logical Methods in Computer Science Volume 18, Issue 4, doi:10.46298/lmcs-18(4:1)2022. Available at https://lmcs.episciences.org/10186.

[9] Matteo Acclavio & Roberto Maieli (2020): *Generalized Connectives for Multiplicative Linear Logic*. In Maribel Fernández & Anca Muscholl, editors: *28th EACSL Annual Conference on Computer Science Logic (CSL 2020), Leibniz International Proceedings in Informatics (LIPIcs)* 152, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 6:1–6:16, doi:10.4230/LIPIcs.CSL.2020.6. Available at https://drops.dagstuhl.de/opus/volltexte/2020/11649.

[10] Matteo Acclavio & Lutz Straßburger (2018): *From Syntactic Proofs to Combinatorial Proofs*. In Didier Galmiche, Stephan Schulz & Roberto Sebastiani, editors: *Automated Reasoning*, Springer International Publishing, Cham, pp. 481–497, doi:10.1007/978-3-319-94205-6_32.

[11] Matteo Acclavio & Lutz Straßburger (2022): *Combinatorial Proofs for Constructive Modal Logic*. 14, p. 15 – 36. Available at https://www.scopus.com/inward/record.uri?eid=2-s2.0-85171882219&partnerID=40&md5=52cad056a46da86e5e0e94b70ca1f569.

[12] Jean-Marc Andreoli (1992): *Logic programming with focusing proofs in linear logic*. Journal of logic and computation 2(3), pp. 297–347, doi:10.1093/logcom/2.3.297.

[13] Jean-Marc Andreoli (2001): *Focussing and proof construction*. Annals of Pure and Applied Logic 107(1), pp. 131 – 163, doi:10.1016/S0168-0072(00)00032-4.

[14] Jean Marc Andreoli (2002): *Focussing proof-net construction as a middleware paradigm*. In: *International Conference on Automated Deduction*, Springer, pp. 501–516, doi:10.1007/3-540-45620-1_39.

[15] Jean-Marc Andreoli & Laurent Mazaré (2003): *Concurrent Construction of Proof-Nets*. In Matthias Baaz & Johann A. Makowsky, editors: *Computer Science Logic*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 29–42, doi:10.1007/978-3-540-45220-1_3.

[16] Arnon Avron & Iddo Lev (2001): *Canonical Propositional Gentzen-Type Systems*. In: *in Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR 2001) (R. Goré, A Leitsch, T. Nipkow, Eds), LNAI 2083*, Springer Verlag, pp. 529–544, doi:10.1007/3-540-45744-5_45.

[17] David Baelde, Amina Doumane & Alexis Saurin (2016): *Infinitary proof theory : the multiplicative additive case* . Available at `https://hal.science/hal-01339037`. Working paper or preprint.

[18] J. C. M. Baeten & S. Mauw (1995): *Delayed choice: an operator for joining Message Sequence Charts*, pp. 340–354. Springer US, Boston, MA, doi:10.1007/978-0-387-34878-0_27. Available at `https://doi.org/10.1007/978-0-387-34878-0_27`.

[19] F. Bonchi, F. Gadducci, A. Kissinger, P. Sobocinski & F. Zanasi (2016): *Rewriting modulo symmetric monoidal structure*. *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–10, doi:10.1145/2933575.2935316.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein (2001): *Introduction to Algorithms*, 2nd edition. The MIT Press.

[21] Vincent Danos & Laurent Regnier (1989): *The structure of multiplicatives*. Archive for Mathematical logic 28(3), pp. 181–203, doi:10.1007/BF01622878.

[22] Anupam Das (2021): *On the Logical Strength of Confluence and Normalisation for Cyclic Proofs*. In Naoki Kobayashi, editor: *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference), LIPIcs 195*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 29:1–29:23, doi:10.4230/LIPIcs.FSCD.2021.29. Available at `https://doi.org/10.4230/LIPIcs.FSCD.2021.29`.

[23] Anupam Das & Damien Pous (2018): *Non-Wellfounded Proof Theory For (Kleene+Action)(Algebras+Lattices)*. In Dan Ghica & Achim Jung, editors: *27th EACSL Annual Conference on Computer Science Logic (CSL 2018), Leibniz International Proceedings in Informatics (LIPIcs) 119*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 19:1–19:18, doi:10.4230/LIPIcs.CSL.2018.19. Available at `http://drops.dagstuhl.de/opus/volltexte/2018/9686`.

[24] T. Ehrhard & L. Regnier (2006): *Differential interaction nets*. Theoretical Computer Science 364(2), pp. 166 – 195, doi:https://doi.org/10.1016/j.tcs.2006.08.003. Available at `http://www.sciencedirect.com/science/article/pii/S0304397506005299`. Logic, Language, Information and Computation.

[25] Boris Eng (2020): *Stellar Resolution: Multiplicatives - for the linear logician, through examples*. Available at `https://hal.science/hal-02977750`. Working paper or preprint.

[26] Boris Eng & Thomas Seiller (2020): *Stellar Resolution: Multiplicatives*. arXiv:2007.16077.

[27] Boris Eng & Thomas Seiller (2021): *A gentle introduction to Girard's Transcendental Syntax*. In: *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*, Rome (virtual), Italy. Available at `https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271496`.

[28] Boris Eng & Thomas Seiller (2021): *Multiplicative Linear Logic from Logic Programs and Tilings*. Available at `https://hal.science/hal-02895111`. Working paper or preprint.

[29] Christophe Fouquere & Virgile Mogbil (2004): *Modules and Logic Programming*. arXiv:cs/0411029.

[30] Jean-Yves Girard (1987): *Linear logic*. Theoretical Computer Science 50(1), pp. 1–101, doi:10.1016/0304-3975(87)90045-4.

[31] Jean-Yves Girard (1987): *Multiplicatives*. In G. Lolli, editor: *Logic and Computer Science: New Trends and Applications*, Rosenberg & Sellier, pp. 11–34.

[32] Jean-Yves Girard (2000): *On the meaning of logical rules II: multiplicatives and additives*. NATO ASI Series F Computer and Systems Sciences 175, pp. 183–212.

[33] Jean-Yves Girard (2017): *Transcendental syntax I: deterministic case*. Math. Struct. Comput. Sci. 27(5), pp. 827–849, doi:10.1017/S0960129515000407. Available at `https://doi.org/10.1017/S0960129515000407`.

[34] S. Guerrini (1999): *Correctness of multiplicative proof nets is linear*. In: *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pp. 454–463, doi:10.1109/LICS.1999.782640.

[35] Rémy Haemmerlé, François Fages & Sylvain Soliman (2007): *Closures and Modules Within Linear Logic Concurrent Constraint Programming*. In V. Arvind & Sanjiva Prasad, editors: *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 544–556, doi:10.1007/978-3-540-77050-3_45.

[36] Willem Heijltjes & Robin Houston (2014): *No Proof Nets for MLL with Units: Proof Equivalence in MLL is PSPACE-Complete*. In: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, Association for Computing Machinery, New York, NY, USA, doi:10.1145/2603088.2603126. Available at `https://doi.org/10.1145/2603088.2603126`.

[37] Willem B. Heijltjes, Dominic J. D. Hughes & Lutz Straßburger (2019): *Intuitionistic proofs without syntax*. In: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–13, doi:10.1109/LICS.2019.8785827.

[38] David Hemer, Robert Colvin, Ian Hayes & Paul Strooper (2002): *Don't Care Non-determinism in Logic Program Refinement*. Electronic Notes in Theoretical Computer Science 61, pp. 101–121, doi:`https://doi.org/10.1016/S1571-0661(04)00308-1`. Available at `https://www.sciencedirect.com/science/article/pii/S1571066104003081`. CATS'02, Computing: the Australasian Theory Symposium.

[39] Ross Horne, Sjouke Mauw & Alwen Tiu (2017): *Semantics for Specialising Attack Trees based on Linear Logic*. Fundamenta Informaticae 153, pp. 57–86, doi:10.3233/FI-2017-1531.

[40] Dominic Hughes & Willem Heijltjes (2016): *Conflict Nets: Efficient Locally Canonical MALL Proof Nets*. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, Association for Computing Machinery, New York, NY, USA, p. 437–446, doi:10.1145/2933575.2934559. Available at `https://doi.org/10.1145/2933575.2934559`.

[41] Dominic J.D. Hughes (2006): *Towards Hilbert's 24th Problem: Combinatorial Proof Invariants: (Preliminary version)*. Electronic Notes in Theoretical Computer Science 165, pp. 37–63, doi:`https://doi.org/10.1016/j.entcs.2006.05.036`. Available at `https://www.sciencedirect.com/science/article/pii/S1571066106005135`. Proceedings of the 13th Workshop on Logic, Language, Information and Computation (WoLLIC 2006).

[42] Olivier Laurent & Roberto Maieli (2008): *Cut Elimination for Monomial MALL Proof Nets*. In: *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pp. 486–497, doi:10.1109/LICS.2008.31.

[43] Chuck Liang & Dale Miller (2021): *Focusing Gentzen's LK proof system*. Available at `https://hal.science/hal-03457379`. Working paper or preprint.

[44] Roberto Maieli (2014): *Construction of Retractile Proof Structures*. In Gilles Dowek, editor: *Rewriting and Typed Lambda Calculi*, Springer International Publishing, pp. 319–333, doi:10.1007/978-3-319-08918-8_22.

[45] Roberto Maieli (2019): *Non decomposable connectives of linear logic*. Annals of Pure and Applied Logic 170(11), p. 102709, doi:10.1016/j.apal.2019.05.006.

[46] Roberto Maieli (2021): *Probabilistic logic programming with multiplicative modules*. In Christian Retoré & E. Pimentel, editors: *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*, Rome (virtual), Italy. Available at `https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271511`.

[47] Roberto Maieli (2022): *A Proof of the Focusing Theorem via MALL Proof Nets*. In Agata Ciabattoni, Elaine Pimentel & Ruy J. G. B. de Queiroz, editors: *Logic, Language, Information, and Computation - 28th International Workshop, WoLLIC 2022, Iaşi, Romania, September 20-23, 2022, Proceedings, Lecture Notes in Com-*

puter Science 13468, Springer, pp. 1–17, doi:`10.1007/978-3-031-15298-6_1`. Available at `https://doi.org/10.1007/978-3-031-15298-6_1`.

[48] Roberto Maieli & Quintijn Puite (2005): *Modularity of proof-nets*. Archive for Mathematical Logic 44(2), pp. 167–193, doi:`10.1007/s00153-004-0242-2`.

[49] Sjouke Mauw & Martijn Oostdijk (2006): *Foundations of Attack Trees*. 3935, pp. 186–198, doi:`10.1007/11734727_17`.

[50] Dale Miller (1995): *A survey of linear logic programming*. Computational Logic: The Newsletter of the European Network of Excellence in Computational Logic 2(2), pp. 63–67.

[51] Dale Miller (2004): *Overview of Linear Logic Programming*. In Thomas Ehrhard, editor: *Linear Logic in Computer Science*, Cambridge University Press, pp. 316–119.

[52] Dale Miller & Elaine Pimentel (2013): *A formal framework for specifying sequent calculus proof systems*. Theoretical Computer Science 474, pp. 98–116, doi:`10.1016/j.tcs.2012.12.008`.

[53] Stefano M. Nicoletti, E. Moritz Hahn & Mariëlle Stoelinga (2022): *BFL: a Logic to Reason about Fault Trees*. In: *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 441–452, doi:`10.1109/DSN53405.2022.00051`.

[54] Judea Pearl (1988): *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[55] Christian Retoré (1993): *Réseaux et séquents ordonnés*. Theses, Université Paris-Diderot - Paris VII. Available at `https://theses.hal.science/tel-00585634`.

[56] Alexis Saurin (2023): *A Linear Perspective on Cut-Elimination for Non-wellfounded Sequent Calculi with Least and Greatest Fixed-Points*. In Revantha Ramanayake & Josef Urban, editors: *Automated Reasoning with Analytic Tableaux and Related Methods*, Springer Nature Switzerland, Cham, pp. 203–222, doi:`10.1007/978-3-031-43513-3_12`.