The Sum-Product Algorithm For Quantitative Multiplicative Linear Logic

Thomas Ehrhard
Université de Paris Cité, CNRS, IRIF, F-75013, Paris, France

Claudia Faggian
Université de Paris Cité, CNRS, IRIF, F-75013, Paris, France

Michele Pagani
Université de Paris Cité, IRIF, F-75013, Paris, France

Abstract

We consider an extension of multiplicative linear logic which encompasses bayesian networks and expresses samples sharing and marginalisation with the polarised rules of contraction and weakening. We introduce the necessary formalism to import exact inference algorithms from bayesian networks, giving the sum-product algorithm as an example of calculating the weighted relational semantics of a multiplicative proof-net improving runtime performance by storing intermediate results.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Denotational semantics; Mathematics of computing → Variable elimination

Keywords and phrases Linear Logic, Proof-Nets, Denotational Semantics, Probabilistic Programming

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.8

Funding This work was partially supported by ANR PRC project PPS ANR-19-CE48-0014.

1 Introduction

Linear logic [18] provides a linear algebra flavour to logic, associating linear algebra operations with logical connectives, e.g. tensor $\otimes$ is seen as a form of conjunction, direct sum $\oplus$ as a disjunction and duality as an involutive negation $(\cdot)^\perp$. This perspective has given many insights. In denotational semantics, we have *quantitative semantics*, e.g. [25, 21, 10, 11, 6, 24]: a family of models denoting λ-terms and functional programs with some notion of analytic maps or power series that can be locally approximated by multilinear functions, these latter denoting linear logic proofs. In proof-theory, we have *proof-nets*: a representation of proofs and programs expressing the interdependences of these algebraic operations in a graph-theoretical fashion.

Quantitative semantics turns out to be particularly suitable for probabilistic programming, giving fully abstract semantics [14, 15, 17], denoting probabilistic programs with very regular functions (absolutely monotone) even on "continuous" datatypes (e.g. real numbers) [16, 5, 13], giving a compositional analysis of various operational behaviours, such as runtime or liveness [24], providing suitable notions of program metrics [12]. Due to this expressivity, calculating the quantitative denotations for a Turing complete programming language is obviously non-computable, but we can fix on relevant fragments supporting an effective procedure. Effectiveness is a relevant feature for a denotational model, as it can provide automatic tools for verifying program correctness, as well as the other mentioned operational properties.

Let us focus our attention to one of the simplest fragments of linear logic: the multiplicative fragment (MLL), which has the $\otimes$ conjunction, its unit 1 and their respective duals, the par $⅋$ (a disjunction different from $\oplus$) and $\perp = 1^\perp$. From a programming perspective, this fragment contains (although it is not restricted to) an exponential-free fragment of the linear λ-calculus with tuples, e.g. [1]: the linear functional type $F \multimap G$ is in fact represented by $F^\perp \parr G$. Although very simple, this fragment is already surprisingly expressive on probabilistic data. First, positive types (i.e. combinations of 1, $\oplus$ and $\otimes$) express linear combinations of the values of a finite data-type. For example, the quantitative denotation of $1 \oplus 1$ contains linear combinations of booleans and can be used to model boolean random variables[1]. Moreover, it is known since the inception of polarised linear logic that positive formulas are endowed with a polarised version of the structural rules of weakening and contraction ([19] and Remark 3), so one can represent λ-terms having multiple occurrences of a same boolean variable without breaking the linearity features of MLL. In probabilistic programming, these occurrences duplicate the *samples* from a random variable, but not the random variable itself. Finally, we can allow semantical boxes expressing matrices indexed by finite data-types, which can express conditional probabilities. We call this system *quantitative* MLL (Section 2).

As for the semantics, let us focus on the $\mathbb{R}_{\geq 0}$*-weighted relation semantics* (see Section 3 and [24]), which is one of the most basic examples of quantitative semantics, allowing to model probabilistic programs over countable data-types. The denotation of a proof-net is then a vector of dimension equal to the number of the possible samples of a probabilistic distribution computed by the proof-net. This vector is computable for quantitative MLL and the standard semantical definitions yield a recursive procedure (Figure 1c) to compute it. In practice, this procedure is unfeasible, as it is exponential in time and in space with respect to the size of the proof-net. The goal of this paper is to inaugurate a new approach for improving it by taking inspiration from bayesian networks, which have partially a similar graph-theoretical structure as proof-nets.

For example, the $\mathbb{R}_{\geq 0}$-weighted denotation of a proof-net describing a probabilistic distribution over a tuple of n booleans is a vector of dimension 2^n (the number of the possible outcomes of a random variable over n booleans), independently whether the values of some of these booleans depend each other or not (Example 9). The proof-net carries very clearly these interdependences via paths over boolean edges: may we reduce the dimension of its denotation by following such a structure? On a different note, the composition of two proof-nets on a tuple of n booleans yields a sum of 2^n terms (Example 11). However, this composition can be ordered by following the switching paths over the corresponding cuts. May we refactor the sum according to this order and gain in efficiency by memorising some intermediate factors?

Similar questions are typical of the research on Bayesian networks ([27], see as reference [8]), these latter being directed graphs expressing the conditional dependences between different random variables. The benefit of this approach is to provide a battery of algorithms computing, e.g., marginal distributions in a quite efficient way by taking advantage of the graph-theoretical structure of a network. Our general goal is to inaugurate a new approach to quantitative semantics which pays attention to the cost of computing the semantics, and we do so by exploiting techniques form Bayesian inference. One paradigmatic example is the *sum-product variable elimination algorithm* [29]: we propose here a formalism for computing the semantics of a quantitative MLL proof-net by adapting this algorithm (here Algorithm 1).

[1] It is known that the space of random variables ranging over a *finite* set of outcomes of cardinality n can be described by the finite additive disjunction $\bigoplus_{i \leq n} 1$ of the tensor unit, see e.g. [17]. This formula is not in MLL, as $\oplus$ is not a multiplicative connective, but it appears in our setting as these spaces of finite random variables are associated with the positive atomic formulas of MLL (see Example 7).

Related works. Bayesian networks form, mutatis mutandis, a strict subset of quantitative MLL proof-nets (Remark 1), morally the set of those proof-nets which do not contain formulas alternating polarities, e.g. alternation of $\otimes$ and $\parr$ connectives. This correspondence has been already acknowledged, with a slight different terminology, by the recent literature about the semantical foundations of Bayesian programming. We mention in particular [3, 22] which represent Bayesian networks as string diagrams and analyse the notion of disintegration. The paper [26] proposes a game semantics based on event structures for a variant of the linear λ-calculus underlined by quantitative MLL. The paper [28] studies an equational theory and provides a denotational semantics based on matrices for this calculus when restricted to ground data-types. However, to our knowledge, our paper is the first time that the efficiency of computing the semantics is taken into consideration. Moreover, we show that the techniques of Bayesian networks can be adapted to the more general framework of quantitative MLL without so much effort.

Paper outline. Section 2 introduces quantitative MLL proof-nets and Section 3 its associated $\mathbb{R}_{\geq 0}$-weighted relational semantics. Section 4 revisits the standard notion of factor in Bayesian inference so to apply it to atomic proof-nets in Section 5 and to general proof-nets in Section 6. Section 7 concludes by mentioning some future developments.

2 Quantitative Multiplicative Linear Logic

Metavariables X, Y, Z will vary over a countable set of propositional variables. The grammar of the formulas of MLL is given by (together with its metavariables):

$$F, G, H ::= X^+ \mid X^- \mid 1 \mid \bot \mid F \otimes G \mid F \parr G. \tag{1}$$

We call X^+ (resp. X^-) a *positive atomic formula* (resp. *negative atomic formula*) over the variable X, the superscript symbol $+$ (resp. $-$) being its *polarity*. We will write X° for a generic atomic formula over X, if we do not want to precise its polarity. The linear logic negation is introduced as syntactical sugar: $(X^+)^\perp ::= X^-$, $1^\perp ::= \bot$, $(F \otimes G)^\perp ::= F^\perp \parr G^\perp$, and for the dual cases $(X^-, \bot, \parr)$, $(F^\perp)^\perp ::= F$.

A *sequent* is a finite sequence $F_1, \ldots, F_n$ of MLL formulas. Capital Greek letters $\Gamma, \Delta, \ldots$ will vary over sequents. Given a sequent $\Gamma = F_1, \ldots, F_n$, we write $\Gamma^\perp$ for the sequent $F_1^\perp, \ldots, F_n^\perp$. Moreover, if $n > 0$, we write $\parr\Gamma$ (resp. $\otimes\Gamma$) for the formula $F_1 \parr (\cdots \parr F_n)$ (resp. $F_1 \otimes (\cdots \otimes F_n)$). If Γ is empty (i.e. $n = 0$), $\parr\Gamma$ (resp. $\otimes\Gamma$) will mean $\bot$ (resp. 1).

As accustomed in linear logic, sequent proofs are represented by special graphs, called *proof-nets*. Figure 1e gives an example of two proof-nets: $\mathcal{N}$ at the left side of the arrow $\xrightarrow{*}$, and $\mathcal{N}_0$ at the right side. A proof-net is a labelled directed acyclic graph[2] (DAG for short) such that the edges are labelled by MLL formulas and the nodes by deduction rules of our extended MLL, i.e. by a symbol among: `ax` (axiom), `cut` (cut), 1 (one), $\otimes$ (tensor), $\bot$ (bottom), $\parr$ (par), `w` (weakening), `c` (contraction), `b` (semantical box or simply box). The nodes of the proof-nets in Figure 1e are represented just by their labels, except for the box which is depicted as a rectangular and labeled by an enumerated occurrence of `b`. The label of a node determines the number of incoming edges (called *premises* of the node) and the

[2] More formally, a directed graph is a quadruplet $(V, E, \mathtt{t}, \mathtt{s})$ of a set V of vertices and a set E of edges, and two maps $\mathtt{t}, \mathtt{s} : E \mapsto V$ associating an edge with a target and a source, respectively. We alllow directed graphs with pending edges, i.e. `t` and `s` may be partial partial. The edges not in the domain of `t` or `s` are called *pending*. A directed graph is acyclic (a DAG for short), if there is no directed cycle.

number of outgoing edges (called *conclusions* of the node), as well as the type of formulas labelling these edges, according to the rules sketched in Figure 1a. The edges will be oriented top-bottom, so that axioms, ones, bottoms, weakenings and boxes have no premises, while cuts have no conclusions. Figure 1e does not explicit all formulas labelling the edges of $\mathcal{N}$ and $\mathcal{N}_0$, in fact these formulas can be recovered by the axioms and boxes labelling and the rules sketched in Figure 1a. Proof-nets have edges without targets which are called *the conclusions of the proof-net.* Both $\mathcal{N}$ and $\mathcal{N}_0$ have one single conclusion, labelled by $X_4^+ \otimes X_5^+$.

Not all DAGs of MLL nodes are proof-nets: the set of *proof-nets* is the subset of the set of all DAGs which can be generated inductively by the rules sketched in Figure 1b. We call *atomic* a proof-net whose edges are only labelled with atomic formulas. Notice that atomic proof-nets can contain only axioms, cuts, weakening, contractions and semantical boxes.

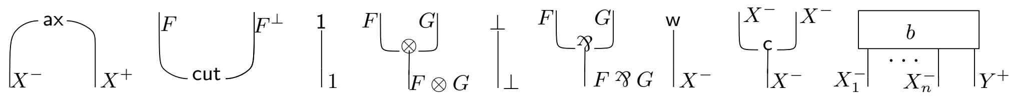
▶ **Example 1.** The (atomic) proof-net $\mathcal{N}_0$ in Figure 1e is mutatis mutandis an example of a Bayesian network as expressed by quantitative MLL. The propositional variables $X_1, \dots, X_5$ are place-holders for (sets of the possible outcomes of) random variables and the semantical boxes are place-holders for their associated "conditional probabilistic tables" (we borrow here the terminology of [8]). For example, the box b_4 is a place-holder for a probabilistic distribution over the variable X_4 conditioned by the outcomes of the variables X_2 and X_3. The polarities discriminate between input and output occurrences in a conditional probabilistic table. These place-holders will be instantiated with concrete conditional distributions by the semantics, as detailed in Section 3.

The acquainted reader in Bayesian graphs should be convinced that these latter are depicted plainly in this syntax just by adding cuts transforming outputs into inputs. Notice that by inverting the orientation of the edges labelled by negative atoms, we get exactly the same directed paths between the nodes of the corresponding Bayesian network. Of course, MLL allows for more nets than Bayesian graphs, for example the proof-net $\mathcal{N}$ at left of the $\xrightarrow{*}$ arrow is not bayesian, namely it has par nodes. But yet, Remark 54 will allude to a correspondence between $\mathcal{N}$ and a run of the sum-product algorithm over $\mathcal{N}_0$. Our goal is to show how Bayesian graph algorithms can be imported in this more general setting.

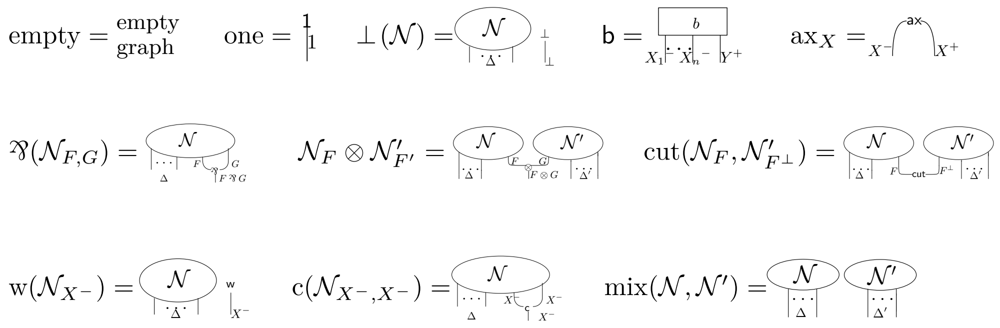
▶ Remark 2. Some papers, e.g. [3, 22], represent Bayesian graphs as string diagrams, which is a graphical syntax omitting the axiom and cut nodes. Although one can present MLL in a similar way by using Lafont's interaction nets [23], we prefer to keep axioms and cuts explicit as they condense the main threats to an efficient computation of the semantics which is a core topic of this paper.

▶ Remark 3. We allow for structural rules (weakening and contraction) on negative atomic formulas. In fact, as it will be clear in Section 3, negative atoms will be interpreted by finite products of bottoms $\with_{x\in S} \perp$ (although we do not detail here the additive connectives & and $\oplus$ and the exponential modalities ? and !). It is well-known since the inception of polarized linear logic [19] that $\perp$ is isomorphic to the exponential formula ?0, so that the structural rules of ? can be lifted to $\with_{x\in S} \perp$, extending the expressivity of MLL. One might even allow the structural rules to all formulas of negative polarity, but we preferred to restrict to atomic formulas to ease the presentation, namely cut reduction.

▶ Remark 4. A less standard extension is given by the semantical boxes b, which are place holders for conditional distributions or, more generally, matrices. For technical convenience, we restrict their conclusions (as well as those of MLL axioms) to be atomic formulas with exactly one occurrence of a positive formula. The structural rules of contractions and weakenings take then a precise operational meaning: a cut between the positive conclusion of a box and a contraction duplicates the *samples* of the probabilistic distribution associated with



(a) Labelling of MLL nodes with their incident edges. Edges are oriented top-down.


empty = empty graph
one =
$\perp(\mathcal{N})$ =
b =
ax_X =
$\parr(\mathcal{N}_{F,G})$ =
$\mathcal{N}_F \otimes \mathcal{N}'_{F'}$ =
$\mathrm{cut}(\mathcal{N}_F, \mathcal{N}'_{F^\perp})$ =
$\mathrm{w}(\mathcal{N}_{X^-})$ =
$\mathrm{c}(\mathcal{N}_{X^-,X^-})$ =
$\mathrm{mix}(\mathcal{N}, \mathcal{N}')$ =

(b) Sequent rules generating the set of proof-nets. The notation $\mathcal{N}_\Gamma$ in the subscript of a rule stands for the pair of a proof-net $\mathcal{N}$ and a sequence Γ of conclusions of $\mathcal{N}$, which will be "active" in the rule.

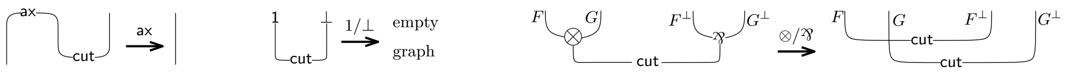
$$\llbracket \mathrm{empty} \rrbracket_\star = \llbracket \mathrm{one} \rrbracket_\star = 1 \qquad \llbracket \perp(\mathcal{N}) \rrbracket_{(\vec{d},\star)} = \llbracket \mathcal{N} \rrbracket_{\vec{d}} \qquad \llbracket \mathsf{b} \rrbracket_{(\vec{x},y)} = \iota(\mathsf{b})_{(\vec{x},y)} \qquad \llbracket \mathrm{ax}_X \rrbracket_{x,x'} = \delta_{x,x'}$$

$$\llbracket \parr(\mathcal{N}) \rrbracket_{(\vec{d},(x,y))} = \llbracket \mathcal{N} \rrbracket_{(\vec{d},x,y)} \qquad \llbracket \mathcal{N} \otimes \mathcal{N}' \rrbracket_{(\vec{d},\vec{d}',(x,y))} = \llbracket \mathcal{N} \rrbracket_{(\vec{d},x)} \llbracket \mathcal{N}' \rrbracket_{(\vec{d}',y)}$$

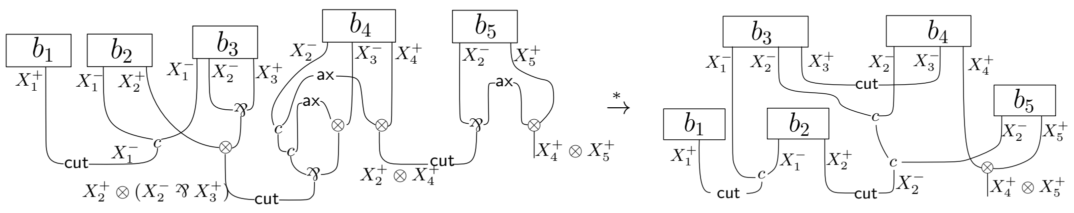
$$\llbracket \mathrm{cut}(\mathcal{N},\mathcal{N}') \rrbracket_{\vec{d},\vec{d}'} = \sum_{x \in \llbracket F \rrbracket} \llbracket \mathcal{N} \rrbracket_{\vec{d},x} \llbracket \mathcal{N}' \rrbracket_{\vec{d}',x}$$

$$\llbracket \mathrm{w}(\mathcal{N}) \rrbracket_{(\vec{d},x)} = \llbracket \mathcal{N} \rrbracket_{\vec{d}} \qquad \llbracket \mathrm{c}(\mathcal{N}) \rrbracket_{(\vec{d},x)} = \llbracket \mathcal{N} \rrbracket_{(\vec{d},x,x)} \qquad \llbracket \mathrm{mix}(\mathcal{N},\mathcal{N}') \rrbracket_{(\vec{d},\vec{d}')} = \llbracket \mathcal{N} \rrbracket_{\vec{d}} \llbracket \mathcal{N}' \rrbracket_{\vec{d}'}$$

(c) Inductive definition of the interpretation $\llbracket \mathcal{N} \rrbracket^\iota$ by induction on a sequence of sequent rules giving $\mathcal{N}$, we omit to explicit the valuation ι as well as the active sequent in the rule.



(d) MLL cut-reduction rewriting steps.



(e) Example of two proof-nets of conclusion $X_4^+ \otimes X_5^+$ such that $\mathcal{N} \xrightarrow{*} \mathcal{N}_0$. The labelling of some edges is omitted.

Figure 1 The proof-net syntax and weighted-relational semantics of quantitative MLL.

the box, while weakenings maginalise out this distribution. These operations are categorically axiomatised by the so-called *CD-structure*, for "*copier*" and "*discarder*", e.g. [22, 28]. We explicit here how the structural polarised linear logic rules perfectly fulfil this rôle, showing a natural Curry-Howard correspondence with Bayesian programming.

Given a proof-net $\mathcal{N}$ and an edge e of $\mathcal{N}$, we write by $e : F$ whenever e is labelled by the formula F. By ease of notation, we often write the sequent $F_1, \dots, F_n$ synonymously for an enumeration $e_1 : F_1, \dots, e_n : F_n$ of labelled edges, if the edges $e_1, \dots, e_n$ are clear from the context or inessential. We write $\mathcal{N} : \Delta$ whenever the sequent Δ enumerates the (formulas labelling the) conclusions of $\mathcal{N}$, also speaking about Δ as simply the conclusions of $\mathcal{N}$.

Cut-reduction is defined as a graph-rewriting, replacing a subgraph containing a cut (the *redex*) with a new subgraph (the *contractum*) having the same pending edges. Figure 1d sketches the three different kinds of MLL redexes: ax, $1/\bot$, $\otimes/\parr$. We will write $\mathcal{N} \to \mathcal{N}'$ if $\mathcal{N}$ rewrites into $\mathcal{N}'$ by one single rewriting step. The fact that $\mathcal{N}'$ is still a proof-net is proven by using the so-called correctness criteria (see [18] for details). We denote by $\xrightarrow{*}$ the reflexive and transitive closure of $\to$. A *normal form* is a proof-net which contains no redex of any kind $\{\mathsf{ax}, 1/\bot, \otimes/\parr\}$. Cut-reduction is confluent and strong normalising [18].

▶ **Example 5.** Figure 1e gives an example of a proof-net $\mathcal{N}$ that rewrites into the normal form $\mathcal{N}_0$. Notice that cuts between structural nodes (weakening and contraction) and boxes are not reduced (see Remark 6) so that the normal form $\mathcal{N}_0$ yet contains some cuts. Notice also that different sequences of rewriting steps may start from $\mathcal{N}$ but all of them can be eventually completed into $\mathcal{N}_0$, in accordance with the confluence property.

▶ Remark 6. Weakening and contraction do not erase nor duplicate semantical boxes as this rewriting would break the correspondence with Bayesian networks mentioned in Example 1. In fact, if we rewrote a cut between a contraction and a box b into two distinct copies of b, then this would correspond to create two independent and identically distributed random variables out of a single one and not to duplicate a sample of this latter. The sharing nodes in bayesian networks share samples of random variables but do not duplicate random variables (see [7]). We will discuss this point also in Example 8 using the weighted relational semantics.

3 Weighted Relational Semantics

The quantitative semantics of linear logic refers to a family of denotational models based on linear algebra constructions (tensors, linear functions, direct sums, dual spaces, etc.). Many examples are known in the literature, such as finiteness and Koethe spaces [11, 10], weighted relations [24], probabilistic coherence spaces [6], coherent Banach spaces [20] etc. The common idea is to associate types with a mathematical structure underlying a notion of vector space (or a module) and the poofs with linear maps represented by matrices, or simply vectors in case of proof-nets. We consider here one of the most basic examples of quantitative semantics, the "relations" weighted by non-negative real numbers, but the results of this paper can be adapted trivially to any quantitative semantics mentioned above.

The model of $\mathbb{R}_{\geq 0}$-weighted relations is a variant of the relational semantics of linear logic (see e.g. [2]), where the notion of a subset of a set S, seen as a vector $(b_x)_{x \in S}$ of booleans expliciting whether an element $x \in S$ belongs or not to the subset, is generalised to a vector of non-negative real numbers. This model is known and we thus just sketch here the interpretation of quantitative MLL proof-nets, referring the reader to [24] for more details.

Let us fix some basic notation. Metavariables S, T, U range over finite sets[3]. We denote by $\mathbb{R}_{\geq 0}$ the cone of the non-negative real numbers. Metavariables ϕ, ψ, ξ will range over vectors in $\mathbb{R}^S_{\geq 0}$, ϕ_x denoting the scalar associated with $x \in S$ by $\phi \in \mathbb{R}^S_{\geq 0}$. The identity matrix over a set S, also called diagonal matrix or Kronecker delta, is denoted $\delta \in \mathbb{R}^{S\times S}_{\geq 0}$ and defined by $\delta_{a,a'} = 1$ if $a = a'$, otherwise $\delta_{a,a'} = 0$.

▶ **Example 7.** The simplest example we consider is the singleton set $\{\star\}$, for some irrelevant element $\star$. The singleton will be associated with multiplicative units 1 and $\perp$ and it induces the module of scalars, as ${\mathbb{R}_{\geq 0}}^{\{\star\}} \simeq \mathbb{R}_{\geq 0}$. The module of couples of non-negative real numbers is instead induced by any set of cardinality 2, like the set of booleans $\{\mathtt{t}, \mathtt{f}\}$: $\mathbb{R}^{\{\mathtt{t},\mathtt{f}\}}_{\geq 0} \simeq \mathbb{R}^2_{\geq 0}$. In all the examples of this paper, we will in fact associate the propositional variables with the set $\{\mathtt{t}, \mathtt{f}\}$, so that a proof-net with only one atomic conclusion will be interpreted with a vector $(\lambda_\mathtt{t}, \lambda_\mathtt{f})$, giving a "score" to the two booleans. Notice that $\mathbb{R}^2_{\geq 0} = \mathbb{R}_{\geq 0} \oplus \mathbb{R}_{\geq 0}$, with $\oplus$ denoting the direct sum over modules. This is reflected in linear logic by encoding the type of booleans with the formula $1 \oplus 1$, where $\oplus$ refers to the additive disjunction. We however avoid this notation as we do not consider the full additive connectives here.

More in general, the interpretation of a MLL formula F is a finite set $[\![F]\!]^\iota$ defined once we have fixed a *valuation* ι as a function mapping the propositional variables to finite sets. The definition of $[\![F]\!]^\iota$ is by induction on F, as follows:

$$[\![X^+]\!]^\iota = [\![X^-]\!]^\iota ::= \iota(X), \quad [\![1]\!]^\iota = [\![\perp]\!]^\iota ::= \{\star\}, \quad [\![F \otimes G]\!]^\iota = [\![F \parr G]\!]^\iota ::= [\![F]\!]^\iota \times [\![G]\!]^\iota.$$

It is easy to check that the usual isomorphisms of linear logic (like associativity and commutativity of the binary connectives) are validated by set isomorphisms. In particular, we can use tuples $(x_1, \dots, x_n)$ for denoting elements in the interpretation of a n-fold connective, e.g. $[\![F_1 \parr (\cdots \parr F_n)]\!]^\iota \simeq \{(x_1, \dots, x_n) \mid \forall i \leq n, x_i \in [\![F_i]\!]^\iota\}$.

Weighted relational semantics equates much more than just linear logic isomorphisms, as for example $[\![F]\!]^\iota = [\![F^\perp]\!]^\iota$ for any formula F. More precisely, this semantics has the structure of a compact closed category. There are more refined examples of quantitative semantics which are not compact closed, e.g. probabilistic coherence spaces. Let us stress that our results do not suppose compact closeness.

The interpretation $[\![\mathcal{N}]\!]^\iota$ of a proof-net $\mathcal{N}$ of conclusions Γ is a vector in $\mathbb{R}^{[\![\parr\Gamma]\!]^\iota}_{\geq 0}$, which can be equivalently seen as a multidimensional matrix indexed by the tuples in $[\![\parr\Gamma]\!]^\iota$. The interpretation can be given inductively as sketched by Figure 1c, once we have associated with each box b of conclusions $X_1^-, \dots, X_n^-, Y^+$ a vector $\iota(\mathsf{b}) \in \mathbb{R}^{[\![(\parr_i X_i^-)\parr Y^+]\!]^\iota}_{\geq 0}$. This interpretation is invariant under the cut-reduction rules of Figure 1d, i.e. $\mathcal{N} \to \mathcal{N}'$ implies $[\![\mathcal{N}]\!] = [\![\mathcal{N}']\!]$.

▶ **Example 8.** Consider the proof-net $\mathcal{N}'$ of conclusion $X_2^+ \otimes (X_2^- \parr X_3^+)$ contained in the proof-net $\mathcal{N}$ depicted at left of Figure 1e and characterised by the three boxes $\mathsf{b}_1, \mathsf{b}_2, \mathsf{b}_3$ and the tensor and par above the cut over $X_2^+ \otimes (X_2^- \parr X_3^+)$. Notice that there is only one sequence of the generating rules of Figure 1b producing this proof-net: one first applies a par rule under the b_3 conclusions X_2^- and X_3^+, then a tensor between the resulting proof-net and b_2, then a contraction between the two X_1^- conclusions and finally a cut between the conclusion of this contraction and b_1. Figure 1c applied to this sequence of rules gives:

$$[\![\mathcal{N}']\!]^\iota_{(x',(x'',x'''))} = \sum_{y \in [\![X_1^+]\!]^\iota} \iota(\mathsf{b}_1)_y \iota(\mathsf{b}_2)_{(y,x')} \iota(\mathsf{b}_3)_{(y,(x'',x'''))}.$$

[3] This kind of denotational semantics are defined for countable sets S in general. Infinite sets are necessary to model linear logic exponential modality as well as the full λ-calculus. Since we focus here to only MLL, we can restrict to finite sets.

Notice that the cut composes the semantics of b_1 with that of the proof-net containing b_2 and b_3, producing the sum over $y \in [\![X_1^+]\!]^\iota$. Notice also that the contraction imposes that the same index y is shared between the two different boxes (b_2 and b_3): contraction duplicates the indexes of the vectors, but it does not yield different copies of the vectors themselves. This is in accordance with Remarks 4 and 6: sharing of sampled values corresponds here to sharing vector indices, which is different from duplicating whole vectors. If we consider in fact the proof-net $[\![\mathcal{N}'']\!]^\iota$ given by a tensor between b_2 and b_3 and two *distinct* copies of b_1, one cut with the X_1^- conclusion of b_2 and the other one with that of b_3, then we would have:

$$[\![\mathcal{N}'']\!]^\iota_{(x',(x'',x''))} = \sum_{y,y' \in [\![X_1^+]\!]^\iota} \iota(\mathsf{b}_1)_y \iota(\mathsf{b}_2)_{(y',x')} \iota(\mathsf{b}_1)_{y'} \iota(\mathsf{b}_3)_{(y',(x'',x''))}.$$

▶ **Example 9.** Let us consider a proof-net $\mathcal{N}$ which is a bunch of $n+1$ axioms over a tree of n contractions, of which edges are labelled by X^-, so that $\mathcal{N}$ has conclusions $X^-, X^+, \dots, X^+$. The denotation $[\![\mathcal{N}]\!]^\iota$ is then a vector indexed by the $(n+2)$-tuples of elements in $\iota(X)$. In fact, by using Figure 1c, one can check that $[\![\mathcal{N}]\!]^\iota$ is a very sparse vector, having zero everywhere but on the tuples of equal elements, i.e. $(x, x, \dots, x)$ for $x \in \iota(X)$, in which case $[\![\mathcal{N}]\!]^\iota$ returns 1. We have here a first source of inefficiency of this kind of semantics, representing the denotation of a proof-net with a vector of dimension exponential in the number of its conclusions, where it would suffice a much more compact structure to store the same information. Section 5 will provide this structure with the notion of *component factor*.

If $\mathcal{N}$ has several cuts, the computation of $[\![\mathcal{N}]\!]^\iota$ can be considerably simplified by using the following lemma, which is reminiscent of the notion of experiment introduced in [18].

▶ **Lemma 10** (Cut bundles). *Let* $\mathrm{Cut}_\Gamma(\mathcal{N})$ *be a proof-net of conclusions* Δ *that can be decomposed into a proof-net* $\mathcal{N}$ *of conclusions* $\Delta, \Gamma, \Gamma^\perp$ *and a bundle of cuts between the formulas in* Γ *and* $\Gamma^\perp$. *Then, for every* $\vec{d} \in [\![\Delta]\!]^\iota$, *we have:* $[\![\mathrm{Cut}_\Gamma(\mathcal{N})]\!]^\iota_{\vec{d}} = \sum_{\vec{c} \in [\![\Gamma]\!]} [\![\mathcal{N}]\!]^\iota_{(\vec{d},\vec{c},\vec{c})}$.

▶ **Example 11.** Let us compute the semantics of the proof-net $\mathcal{N}_0$ in Figure 1e, by using Lemma 10 and Figure 1c. We have that, for any $(x_4, x_5) \in [\![X_4^+ \otimes X_5^+]\!]^\iota$:

$$[\![\mathcal{N}_0]\!]^\iota_{(x_4,x_5)} = \sum_{\substack{x_i \in \iota(X_i^+) \\ \text{for } i \in \{1,2,3\}}} \iota(\mathsf{b}_1)_{x_1} \iota(\mathsf{b}_2)_{(x_1,x_2)} \iota(\mathsf{b}_3)_{(x_1,x_2,x_3)} \iota(\mathsf{b}_4)_{(x_2,x_3,x_4)} \iota(\mathsf{b}_5)_{(x_2,x_5)}$$

With a bit more of effort (due to the presence of axioms) also $[\![\mathcal{N}]\!]^\iota$ can be associated with the above summation. If we suppose that for every i, $\iota(X_i) = \{\mathtt{t}, \mathtt{f}\}$, this summation has a total of 2^3 terms, so that computing the whole vector $[\![\mathcal{N}_0]\!]^\iota$ requires $\sim 2^5$ basic operations[4], i.e. a quantity exponential in the number of the semantical boxes.

By carefully inspecting the summation, one can however realise that it can be refactored so to split factors over independent variables, getting for example the expression:

$$\sum_{x_3} \Big(\sum_{x_2} \big(\sum_{x_1} \iota(\mathsf{b}_1)_{x_1} \iota(\mathsf{b}_2)_{(x_1,x_2)} \iota(\mathsf{b}_3)_{(x_1,x_2,x_3)} \big) \iota(\mathsf{b}_4)_{(x_2,x_3,x_4)} \Big) \iota(\mathsf{b}_5)_{(x_2,x_5)}$$

which, by memorising the intermediate sums, performs the same computation of $[\![\mathcal{N}]\!]^\iota$ in just $\sim 2^3$ operations.This kind of refactoring is at the core of many algorithms for exact inference in Bayesian graphs and the next sections will show how to import these methods.

[4] We are supposing that multiplication, addition and coefficient access are operations of constant cost.

4 Factors

We adapt from Bayesian networks (e.g. [8]) the notion of factor (Definition 18) and of product and projection of factors. A factor carries both a vector *and* a "sharing structure" about what entries of this vector will be shared with possibly other factors so that we avoid the dimension explosion which is the source of inefficiency in Example 9. Bayesian networks use random variables for expressing such a "sharing structure", while we reduce this latter into the very basic definition of set-family, which encompasses the former (Example 15) and generalises to whole quantitative MLL. The terminology "factor" is standard in Bayesian networks, in fact this notion refers to the terms in the multiplication giving a joint distribution as the outcome of the variable elimination algorithm (Algorithm 1). We introduce also a notion of renaming (Definition 29) and of factor renaming (Definition 34) necessary to follow the compositional structure of MLL (see discussion in Example 41).

▶ **Definition 12** (Set-family). *We call* set-family *a finite, indexed family of finite sets, i.e. a map $\mathbb{X}$ from a finite set $\mathcal{I}(\mathbb{X})$ of indices to a set* $\mathsf{Sets}(\mathbb{X})$ *of finite sets. We denote by $\mathbb{X}(a)$ the set associated with index $a \in \mathcal{I}(\mathbb{X})$ in $\mathbb{X}$. Meta-variables $\mathbb{X}$, $\mathbb{Y}$, $\mathbb{Z}$ will range over such set-families.*

Two families $\mathbb{X}$ and $\mathbb{Y}$ are compatible *whenever for all $a \in \mathcal{I}(\mathbb{X}) \cap \mathcal{I}(\mathbb{Y})$, $\mathbb{X}(a) = \mathbb{Y}(a)$. Set-theoretical operations lift to compatible set-families by applying the former to the graph of these latter, e.g. the intersection $\mathbb{X} \cap \mathbb{Y}$ is the set-family defined by $\mathcal{I}(\mathbb{X} \cap \mathbb{Y}) ::= \mathcal{I}(\mathbb{X}) \cap \mathcal{I}(\mathbb{Y})$ and $(\mathbb{X} \cap \mathbb{Y})(a) ::= \mathbb{X}(a) = \mathbb{Y}(a)$ for every $a \in \mathcal{I}(\mathbb{X} \cap \mathbb{Y})$. Similarly, we will consider the union $\mathbb{X} \cup \mathbb{Y}$ and the set-theoretical difference $\mathbb{X} \setminus \mathbb{Y}$. In the same spirit, we write $\mathbb{Y} \subseteq \mathbb{X}$, for $\mathcal{I}(\mathbb{Y}) \subseteq \mathcal{I}(\mathbb{X})$ and for every $a \in \mathcal{I}(\mathbb{Y})$, $\mathbb{Y}(a) = \mathbb{X}(a)$.*

Given a set-family $\mathbb{X}$, we denote by $[\![\mathbb{X}]\!]$ the cartesian product $\prod_{a\in\mathcal{I}(\mathbb{X})} \mathbb{X}(a)$ of the sets in $\mathsf{Sets}(\mathbb{X})$*, where the same set in* $\mathsf{Sets}(\mathbb{X})$ *can appear multiple times in the product if associated with multiple indices. We denote the elements of $[\![\mathbb{X}]\!]$ with the vectorial notation $\vec{x}$, to underline that it is an element in a cartesian product rather than in a simple set.*

▶ **Notation 13.** *Any element $\vec{x} \in [\![\mathbb{X}]\!]$ can be seen as a collection $(x_a)_{a\in\mathcal{I}(\mathbb{X})}$ of elements in* $\mathsf{Sets}(\mathbb{X})$*. In particular, given $\mathbb{Y} \subseteq \mathbb{X}$, we denote by $\vec{x}|_{\mathbb{Y}}$ the projected element $(x_a)_{a\in\mathcal{I}(\mathbb{Y})} \in [\![\mathbb{Y}]\!]$. Similarly, given two set-families $\mathbb{X}, \mathbb{Y}$ having disjoint sets of indexes, so clearly compatible, the elements of $[\![\mathbb{X} \uplus \mathbb{Y}]\!]$ can be written as $(\vec{x}, \vec{y})$, for $\vec{x} \in [\![\mathbb{X}]\!]$ and $\vec{y} \in [\![\mathbb{Y}]\!]$.*

Notice that if $\mathbb{X}$ is empty, then $[\![\mathbb{X}]\!]$ is the singleton set $\{()\}$.

▶ **Notation 14.** *Since finite, set-families can be given by enumerating their graph, like in $\mathbb{X} = \{(a_1, S_1), \dots, (a_n, S_n)\}$. In this case we have: $\mathcal{I}(\mathbb{X}) = \{a_1, \dots, a_n\}$ and* $\mathsf{Sets}(\mathbb{X}) = \{S_1, \dots, S_n\}$*. In this latter set, the possible repetitions are equated, so* $\mathsf{Sets}(\mathbb{X})$ *might have less than n elements.*

▶ **Example 15.** A finite set $\{X_1, \dots, X_n\}$ of finite random variables defines the set-family $\mathbb{X} = \{(X_1, [\![X_1]\!]), \dots, (X_n, [\![X_n]\!])\}$, where $[\![X_i]\!]$ denotes the finite set of the possible outcomes taken by the random variable X_i. Notice that $[\![\mathbb{X}]\!]$ is then the set of samples of the joint distribution over $X_1, \dots, X_n$. To be more explicit, suppose that each random variable X_i is boolean, i.e. $[\![X_i]\!] = \{\mathtt{t}, \mathtt{f}\}$ for all $i \leq n$, then $\mathsf{Sets}(\mathbb{X}) = \{\{\mathtt{t}, \mathtt{f}\}\}$, while $[\![\mathbb{X}]\!] = \{(b_1, \dots, b_n) \mid b_i \in \{\mathtt{t}, \mathtt{f}\}\}$.

▶ **Example 16.** Consider a sequent $\Gamma = X_1^\circ, \dots, X_n^\circ$ of atomic formulas. A natural set-family that can be associated with Γ and a valuation ι, has indices the sequent positions $\{1, \dots, n\}$ and it maps a position i to the set $\iota(X_i)$. This set-family however is not the only possible one: for example, one may take as indices the propositional variables $X_1, \dots, X_n$, where multiple occurrences of the same variable are equated, and map X_i to $\iota(X_i)$. The two set-families are quite different if Γ contains repetitions. Namely, let $\Gamma = X^+, X,^+ X^-, Y^-$, with $[\![X]\!] = [\![Y]\!] = \{\mathtt{t}, \mathtt{f}\}$. The two set-families are:

$$\mathbb{X} = \{(1, \{\mathtt{t}, \mathtt{f}\}), (2, \{\mathtt{t}, \mathtt{f}\}), (3, \{\mathtt{t}, \mathtt{f}\}), (4, \{\mathtt{t}, \mathtt{f}\})\}, \qquad \mathbb{Y} = \{(X, \{\mathtt{t}, \mathtt{f}\}), (Y, \{\mathtt{t}, \mathtt{f}\})\}.$$

▶ Remark 17. Notice that $\mathbb{Z}, \mathbb{Y} \subseteq \mathbb{X}$ implies that both $\mathbb{Z}$ and $\mathbb{Y}$ are compatible. Henceforth we will always consider families which are subset of a fixed "universal" family (underlined by a proof-net), so that the compatibility condition in Definition 12 is not an issue and hence will be often not mentioned.

▶ **Definition 18** (Factor). *A* generalised factor*, or simply* factor*, ϕ is a pair* $(\mathsf{Fam}(\phi), \mathrm{Fun}(\phi))$ *of a set-family* $\mathsf{Fam}(\phi)$ *and a function* $\mathrm{Fun}(\phi)$ *from the set* $[\![\mathsf{Fam}(\phi)]\!]$ *to* $\mathbb{R}_{\geq 0}$.

We will short the notation $\mathrm{Fun}(\phi)$ *by writing just ϕ when it is clear from the context that we are considering the function associated with a factor and not the whole pair* $(\mathsf{Fam}(\phi), \mathrm{Fun}(\phi))$*. We often consider* $\mathrm{Fun}(\phi)$ *as a vector indexed by the elements of its domain, so that $\phi_{\vec{x}}$ stands for* $\mathrm{Fun}(\phi)(\vec{x})$*, for every* $\vec{x} \in [\![\mathsf{Fam}(\phi)]\!]$.

▶ **Example 19.** Let us recall the set-family $\mathbb{Y} = \{(X, \{\mathtt{t}, \mathtt{f}\}), (Y, \{\mathtt{t}, \mathtt{f}\})\}$ of Example 16, and consider the function $\mathrm{Fun}(\phi)$ given by $\{(\mathtt{t}_X, \mathtt{t}_Y) \mapsto 0.2, (\mathtt{t}_X, \mathtt{f}_Y) \mapsto 0.25, (\mathtt{f}_X, \mathtt{t}_Y) \mapsto 0.25, (\mathtt{f}_X, \mathtt{f}_Y) \mapsto 0.3\}$. The pair $\phi = (\mathbb{Y}, \mathrm{Fun}(\phi))$ is an example of factor. Intuitively, ϕ can be seen as the presentation $0.2e_{(\mathtt{t}_X,\mathtt{t}_Y)} + 0.25e_{(\mathtt{t}_X,\mathtt{f}_Y)} + 0.25e_{(\mathtt{f}_X,\mathtt{t}_Y)} + 0.3e_{(\mathtt{f}_X,\mathtt{f}_Y)}$ of a vector in $\mathbb{R}^4_{\geq 0}$ with respect to a set of basis vectors $e_{(b_X,b_Y)}$ associated with the elements in $(b_X, b_Y) \in [\![\mathbb{Y}]\!]$.

▶ **Definition 20** (Factor projection). *Let ϕ be a factor and let $\mathbb{X}$ be a set-family compatible with* $\mathsf{Fam}(\phi)$*,* the projection of ϕ to $\mathbb{X}$ *is the factor $\pi_{\mathbb{X}}(\phi)$ defined by:*

$$\mathsf{Fam}(\pi_{\mathbb{X}}(\phi)) ::= \mathbb{X}, \qquad \pi_{\mathbb{X}}(\phi)_{\vec{x}} ::= \sum_{\vec{y} \in [\![\mathsf{Fam}(\phi) \setminus \mathbb{X}]\!]} \phi_{(\vec{x}|_{\mathsf{Fam}(\phi)}, \vec{y})}, \qquad \textit{for } \vec{x} \in [\![\mathbb{X}]\!].$$

▶ **Example 21.** Recall the set-family $\mathbb{Y}$ and the factor $\mathrm{Fun}(\phi)$ given in Example 19, let $\mathbb{X} = \{(X, \{\mathtt{t}, \mathtt{f}\})\} \subseteq \mathbb{Y}$. We have that $\pi_{\mathbb{X}}(\phi) = \{\mathtt{t}_X \mapsto 0.45, \mathtt{f}_X \mapsto 0.55\}$. Let now $\mathbb{Z} = \mathbb{X} \uplus \{(Z, \{\mathtt{t}, \mathtt{f}\})\}$, we have that $\pi_{\mathbb{Z}}(\phi) = \{(\mathtt{t}_X, \mathtt{t}_Z) \mapsto 0.45, (\mathtt{t}_X, \mathtt{f}_Z) \mapsto 0.45, (\mathtt{f}_X, \mathtt{t}_Z) \mapsto 0.55, (\mathtt{f}_X, \mathtt{f}_Z) \mapsto 0.55\}$. Notice in particular that the factor projection to a set-family $\mathbb{Z}$ does not preserve in general the property of being a probability mass function, unless $\mathbb{Z} \subseteq \mathsf{Fam}(\phi)$.

▶ Remark 22. With the notations of Definition 20, if $\mathbb{X} \subseteq \mathsf{Fam}(\phi)$, then $\pi_{\mathbb{X}}(\phi)$ corresponds to what is called in Bayesian programming *summing out* $\mathsf{Fam}(\phi) \setminus \mathbb{X}$, which gives the marginal distribution over $\mathbb{X}$. Suppose on the contrary that $\mathbb{X}$ and $\mathsf{Fam}(\phi)$ are disjoint, then for every $\vec{x} \in [\![\mathbb{X}]\!]$, $\pi_{\mathbb{X}}(\phi)_{\vec{x}}$ is the total mass of ϕ, i.e. $\sum_{\vec{y} \in [\![\mathsf{Fam}(\phi)]\!]} \phi_{\vec{y}}$.

▶ Remark 23. Suppose that $\mathsf{Fam}(\phi)$ has n indices and that k is the maximum cardinality of a set in $\mathsf{Sets}(\mathsf{Fam}(\phi))$, then the computation of the whole vector $\pi_{\mathbb{X}}(\phi)$ is in $O(k^n)$.

▶ **Definition 24** (Binary factor product). *Given two factors ϕ and ψ, such that* $\mathsf{Fam}(\phi)$ *and* $\mathsf{Fam}(\psi)$ *are compatible, we define their* factor product *as the factor $\phi \odot \psi$ given by:*

$$\mathsf{Fam}(\phi \odot \psi) ::= \mathsf{Fam}(\phi) \cup \mathsf{Fam}(\psi), \qquad (\phi \odot \psi)_{\vec{z}} ::= \phi_{\vec{z}|_{\mathsf{Fam}(\phi)}} \psi_{\vec{z}|_{\mathsf{Fam}(\psi)}}, \qquad \textit{for } \vec{z} \in [\![\mathsf{Fam}(\phi \odot \psi)]\!].$$

▶ Remark 25. If $\mathsf{Fam}(\phi) \cup \mathsf{Fam}(\psi)$ has n indices and k is the maximum cardinality of a set in $\mathsf{Sets}(\mathsf{Fam}(\phi) \cup \mathsf{Fam}(\psi))$, then the computation of the whole vector $\phi \odot \psi$ is in $O(k^n)$.

▶ **Example 26.** In terms of MLL operations, factor products correspond to a $\otimes$ product plus a bunch of contractions on the common indexes. For example, let us take as indexes the propositional variables and as sets just $\{\mathtt{t}, \mathtt{f}\}$ (recall Example 16) and consider $\mathsf{Fam}(\phi) = \{X_2, X_3, X_4\}$ and $\mathsf{Fam}(\psi) = \{X_2, X_5\}$ (this choice is reminiscent of the variables in the proof-nets in Figure 1e, in fact ϕ and ψ can be associated with the boxes, respectively, b_4 and b_5). Then, $\mathrm{Fun}(\phi \odot \psi)$ is over $\{X_2, X_3, X_4, X_5\}$, so of dimension 2^4, while $\mathrm{Fun}(\phi) \otimes \mathrm{Fun}(\psi)$ is a vector indexed by tuples of 5 booleans, so of dimension 2^5.

The next proposition states expected properties of factor projection and product that are fundamental in the sequel.

▶ **Proposition 27.** *Factor product is associative and commutative, with neutral element the empty factor* $(\emptyset, 1)$*. Moreover:*

1. $\pi_{\mathbb{X}\cup\mathbb{Z}}(\pi_{\mathbb{X}\cup\mathbb{Y}}(\phi)) = \pi_{\mathbb{X}\cup\mathbb{Z}}(\phi)$*, whenever* $\mathbb{Y} \subseteq \mathsf{Fam}(\phi)$ *and* $\mathbb{Z} \cap \mathsf{Fam}(\phi) = \emptyset$*;*
2. $\pi_{\mathbb{X}}(\phi \odot \psi) = \pi_{\mathbb{X}}(\phi) \odot \psi$*, whenever* $\mathsf{Fam}(\psi) \subseteq \mathbb{X}$.

▶ **Definition 28** (n-factor product)**.** *Let I be a finite set. Given a collection of pairwise compatible factors $(\phi_i)_{i\in I}$, we define their* factor product *as the factor* $\bigodot_{i\in I} \phi_i ::= \phi_{i_1} \odot \cdots \odot \phi_{i_n}$*, for some enumeration of I. This is well-defined independently from the chosen enumeration because of Proposition 27.*

Section 5 will associate factors to MLL proof-nets and in order to make this association compositional (Theorem 48) we introduce the following notion of renaming, as the contraction and cut rules of Figure 1b may change the sharing structure associated with a proof-net.

▶ **Definition 29** (Renaming)**.** *A* renaming *f from a set-family $\mathbb{X}$ to a set-family $\mathbb{Y}$ is a map from $\mathcal{I}(\mathbb{X})$ to $\mathcal{I}(\mathbb{Y})$ such that for all $a \in \mathcal{I}(\mathbb{X})$, we have $\mathbb{X}(a) = \mathbb{Y}(f(a))$. Any such renaming f induces the map f° from $[\![\mathbb{Y}]\!]$ to $[\![\mathbb{X}]\!]$ by:*

$$\text{for } \vec{y} \in [\![\mathbb{Y}]\!], \qquad f^\circ(\vec{y}) ::= (y_{f(a)})_{a\in\mathcal{I}(\mathbb{X})} \in [\![\mathbb{X}]\!]. \tag{2}$$

Moreover, we say that a point $\vec{x} \in [\![\mathbb{X}]\!]$ agrees on *f whenever, for every $a, a' \in \mathcal{I}(\mathbb{X})$, $f(a) = f(a')$ implies that $x_a = x_{a'}$.*

▶ Remark 30. The notion of "agreeing on a renaming f" generalises the notion in Bayesian programming of a set of samples that "agrees on the same random variables" as used in e.g. [8].

▶ **Notation 31.** *Given a renaming f from $\mathbb{X}$ to $\mathbb{Y}$, and a set-family $\mathbb{X}' \subseteq \mathbb{X}$, we denote by $f(\mathbb{X}')$ the set-family having as indices the set $f(\mathcal{I}(\mathbb{X}')) \subseteq \mathcal{I}(\mathbb{Y})$ and that it associates with any $b \in f(\mathcal{I}(\mathbb{X}'))$ the set $\mathbb{Y}(b)$. Notice that $f(\mathbb{X}') \subseteq \mathbb{Y}$.*

▶ **Proposition 32.** *Given a renaming from $\mathbb{X}$ to $\mathbb{Y}$, the image set of f° is the subset of $[\![\mathbb{X}]\!]$ of the elements which agree on f. If, moreover, f is surjective over $\mathcal{I}(\mathbb{Y})$, then f° is an injective map from $[\![\mathbb{Y}]\!]$ to $[\![\mathbb{X}]\!]$, hence a bijection from $[\![\mathbb{Y}]\!]$ to $\{\vec{x} \in [\![\mathbb{X}]\!] \mid \vec{x}$ agrees on $f\}$. In this case, we denote its inversion by $f^\bullet$.*

▶ **Example 33.** Recall the set-families $\mathbb{X}$ and $\mathbb{Y}$ in Example 16, associated with the sequent $\Gamma = X^+, X,^+ X^-, Y^-$. Let us consider the following two specific renamings from $\mathbb{X}$ to $\mathbb{Y}$:

$$f = \begin{cases} 1,2,3 & \mapsto X \\ 4 & \mapsto Y \end{cases}, \qquad\qquad g = \begin{cases} 1,2,3,4 & \mapsto X \end{cases}.$$

and take for example $\vec{y} = (\mathtt{t}_X, \mathtt{f}_Y) \in [\![\mathbb{Y}]\!]$. We have (we use the natural order (1,2,3,4) to represent elements in $[\![\mathbb{X}]\!]$):

$$f^\circ(\vec{y}) = (\mathtt{t}, \mathtt{t}, \mathtt{t}, \mathtt{f}), \qquad g^\circ(\vec{y}) = (\mathtt{t}, \mathtt{t}, \mathtt{t}, \mathtt{t}).$$

Notice in fact that $f^\circ(\vec{y})$ agrees on f but not on g, while $g^\circ(\vec{y})$ agrees on both f and g. Also notice that f is surjective and in fact f° is an injection, while g is not surjective and in fact g° is not a injection, for example: $g^\circ(\mathtt{t}_X, \mathtt{f}_Y) = g^\circ(\mathtt{t}_X, \mathtt{t}_Y)$.

▶ **Definition 34** (Factor renaming). *Let f be a renaming from* $\mathsf{Fam}(\phi)$ *to a set-family* $\mathbb{X}$*. The* renaming of ϕ along f *is the factor $f(\phi)$ defined by:*

$$\mathsf{Fam}(f(\phi)) ::= f(\mathsf{Fam}(\phi)), \qquad f(\phi)_{\vec{x}} ::= \phi_{f^\star(\vec{x})}, \qquad \text{for } \vec{x} \in [\![f(\mathsf{Fam}(\phi))]\!].$$

▶ **Example 35.** Recall the renaming f of Example 33 between the set-families $\mathbb{X}$ and $\mathbb{Y}$ given in Example 16. Consider the factor ϕ over $\mathbb{X}$ defined by $\phi_{(b_1,b_2,b_3,b_4)} ::= 1$ if $b_1 = b_2 = b_3$, 0 otherwise. This factor corresponds to the interpretation of the proof-net having a weakening producing Y^- and the axiom on top of a contraction giving X^+, X^+, X^-. Then $f(\phi)$ is over $\mathbb{Y}$ and its map is the constant function giving 1.

▶ Remark 36. Notice that one can formalise the notions of this section in a categorical way, considering a category of renamings as morphisms between set-families. We did not develop this more abstract presentation as not needed in the sequel.

5 Weighted Semantics by Factors, Atomic Case

We apply to MLL the notions introduced in Section 4. It is convenient to restrict to atomic proof-nets and then to extend the results to the non-atomic case in Section 6. Definitions 37 and 45 associate two different set-families with an atomic proof-net $\mathcal{N}$, the edge and the component set-families. The edge set-family permits to consider the standard weighted interpretation $[\![\mathcal{N}]\!]$ as a factor (Definition 39), while the component set-family yields a more compressed representation of $[\![\mathcal{N}]\!]$, the component factor (Definition 40), which has a form of compositionality (Theorem 48) and hence can be computed directly (Theorem 49) without using the rules of Figure 1c. Henceforth, this section fixes a valuation ι and considers only atomic proof-nets. We recall that an atomic proof-net can contain only axioms, cuts, weakening, contractions and semantical boxes.

▶ **Definition 37** (Edge set-family). *Let $\mathcal{N}$ be an atomic proof-net and ι be a valuation. The* edge set-family of $\mathcal{N}$*, written by* $\mathsf{Fam}^\iota_e(\mathcal{N})$*, has the edges of $\mathcal{N}$ as indices and associates with an edge $e : X^\circ$ the set $\iota(X)$. Given a sequence Γ of edges $e_1 : X_1^\circ, \dots, e_n : X_n^\circ$, we extend the metavariable Γ to denote also the edge set-family $\{(e_1, \iota(X_1)), \dots, (e_n, \iota(X_n))\} \subseteq$* $\mathsf{Fam}^\iota_e(\mathcal{N})$.

▶ Remark 38. Let Γ be $e_1 : X_1^\circ, \dots, e_n : X_n^\circ$. The convention of denoting by Γ both the underlined sequent $X_1^\circ, \dots, X_n^\circ$ and the edge set-family $\{(e_1, \iota(X_1)), \dots, (e_n, \iota(X_n))\}$ is coherent because the cartesian product $[\![\Gamma]\!]$ associated with the edge set-family is the same set as the weighted denotation $[\![\Gamma]\!]^\iota$, this latter also denoted simply by $[\![\Gamma]\!]$. This ease of notation is necessary to avoid a formalism overkill.

▶ **Definition 39.** *Given a valuation ι, the* edge factor *of an atomic proof-net $\mathcal{N}$ has as set-family the edge-set family induced by the conclusions of $\mathcal{N}$ and as function the weighted interpretation $[\![\mathcal{N}]\!]^\iota$. We take the liberty to denote this edge factor also by $[\![\mathcal{N}]\!]^\iota$.*

▶ **Definition 40** (Component set-family and renaming). *Let $\mathcal{N}^{\neg b}$ denote the graph obtained from an atomic proof-net $\mathcal{N}$ by removing all its semantical boxes, so keeping their conclusions as pending edges of the graph. Given an undirected connected component $\mathcal{M}$ of $\mathcal{N}^{\neg b}$, one can remark that all edges of $\mathcal{M}$ are atomic formulas over a unique variable, let us denote it $X_{\mathcal{M}}$. The* component set-family of $\mathcal{N}$, *written $\mathsf{Fam}^{\iota}_{c}(\mathcal{N})$, has as indices the undirected connected components of $\mathcal{N}^{\neg b}$ and associates with a component $\mathcal{M}$ the set $\iota(X_{\mathcal{M}})$. The* component renaming of $\mathcal{N}$, *written $\ell_{\mathcal{N}}$, is the renaming from $\mathsf{Fam}^{\iota}_{e}(\mathcal{N})$ to $\mathsf{Fam}^{\iota}_{c}(\mathcal{N})$ mapping the edges of $\mathcal{N}$ to the connected component of $\mathcal{N}^{\neg b}$ they belong to.*

▶ **Example 41.** Consider the atomic proof-net $\mathcal{N}_a$ obtained from the proof-net $\mathcal{N}_0$ in Figure 1e by removing the tensor node, so that $\mathcal{N}_a$ has conclusions X_4^{+}, X_5^{+}. Notice that $\mathcal{N}_a{}^{\neg b}$ has five connected components which correspond to the five propositional variables $X_1, \dots, X_5$. If however we consider the sub proof-net $\mathcal{N}'_a$ obtained from $\mathcal{N}_a$ by removing the cut and the contraction insisting on the edges typed by, e.g., X_1^{-}, we have that $\mathcal{N}'_a{}^{\neg b}$ has now seven connected components, in particular three of them supports the same variable X_1. This example shows that the generating rules of MLL (Figure 1b) require not to mix up the component set-family with the variables labelling the edges (see also Remark 55).

▶ Remark 42. The $\ell_{\mathcal{N}}$ information can be memorised once and for all by adding a further labelling over the edges of $\mathcal{N}$ giving the same index to the edges belonging to the same connected component of $\mathcal{N}^{\neg b}$. This labelling can be computed in linear time with respect to the size of $\mathcal{N}$, by adapting one of the many connected component algorithms.

▶ **Notation 43.** *Let $\Gamma = e_1 : X_1^{\circ}, \dots, e_n : X_n^{\circ}$ be a set of edges of $\mathcal{N}$, so that Γ is also the set-family $\{(e_1, \iota(X_1)), \dots, (e_n, \iota(X_n))\}$ contained in $\mathsf{Fam}^{\iota}_{e}(\mathcal{N})$. By Notation 31, the writing $\ell_{\mathcal{N}}(\Gamma)$ denotes the set-family $\{(\ell_{\mathcal{N}}(e_1), \iota(X_1)), \dots, (\ell_{\mathcal{N}}(e_n), \iota(X_n))\} \subseteq \mathsf{Fam}^{\iota}_{c}(\mathcal{N})$. Notice that $\ell_{\mathcal{N}}$ is surjective on $\mathcal{I}(\ell_{\mathcal{N}}(\Gamma))$, so we can apply Proposition 32, getting the two maps:*

- *$\ell^{\circ}_{\mathcal{N}}$ from $[\![\ell_{\mathcal{N}}(\Gamma)]\!]$ to $[\![\Gamma]\!]$,*
- *its inverse $\ell^{\bullet}_{\mathcal{N}}$ from $\{\vec{d} \in [\![\Gamma]\!] \mid \vec{d}$ agrees on $\ell_{\mathcal{N}}\}$ to $[\![\ell_{\mathcal{N}}(\Gamma)]\!]$.*

▶ Remark 44. In general, given a set of edges Γ of an atomic proof-net $\mathcal{N}$, we have that: $[\![\ell_{\mathcal{N}}(\Gamma)]\!] = [\![\Gamma]\!]$ if, and only if, all edges in Γ belong to pairwise different connected components of $\mathcal{N}^{\neg b}$. Moreover, if $[\![\ell_{\mathcal{N}}(\Gamma)]\!] = [\![\Gamma]\!]$, then every $\vec{d} \in [\![\Gamma]\!]$ agrees on $\ell_{\mathcal{N}}$.

▶ **Definition 45.** *Given a valuation ι, the* component factor *of an atomic proof-net $\mathcal{N}$ is the renaming $\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})$ of its edge factor, i.e. if Δ are the conclusions of $\mathcal{N}$, $\mathsf{Fam}(\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})) = \ell_{\mathcal{N}}(\Delta)$ and for $\vec{d} \in [\![\ell_{\mathcal{N}}(\Delta)]\!]$, $\mathrm{Fun}(\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})_{\vec{d}} = [\![\mathcal{N}]\!]^{\iota}_{\ell^{\circ}_{\mathcal{N}}(\vec{d})}$.*

▶ **Example 46.** Recall the proof-net $\mathcal{N}$ and the valuation ι of Example 9. Notice that $\mathsf{Fam}(\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota}))$ is a singleton as the $n+2$ conclusions of $\mathcal{N}$ belong to the same component, and $\mathrm{Fun}(\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})) = (1_{\mathtt{t}}, 1_{\mathtt{f}})$, which is a more parsimonious object than $[\![\mathcal{N}]\!]^{\iota}$, this latter of dimension exponential in n.

Proposition 47 details how to recover the original denotation of $\mathcal{N}$ out of its component factor.

▶ **Proposition 47.** *Let $\mathcal{N}$ be an atomic proof-net of conclusions Δ. For every $\vec{d} \in [\![\Delta]\!]$, we have that:*

$$[\![\mathcal{N}]\!]^{\iota}_{\vec{d}} = \begin{cases} \ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})_{\ell^{\bullet}_{\mathcal{N}}(\vec{d})} & \text{if } \vec{d} \text{ agrees on } \ell_{\mathcal{N}}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The following is the core theorem of this paper: MLL cuts correspond to a factor product plus a projection.

▶ **Theorem 48.** *Let $\mathcal{N} = \mathrm{Cut}_\Gamma(\mathcal{N}', \mathcal{N}'')$ be an atomic proof-net of conclusions Δ obtained by connecting by a bunch of cuts over a sequent Γ a sub proof-net $\mathcal{N}'$, of conclusions Γ, Δ', and $\mathcal{N}''$, of conclusions $\Gamma^\perp, \Delta''$, so that $\Delta = \Delta', \Delta''$. We have:*

$$\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^\iota) = \pi_{\ell_{\mathcal{N}}(\Delta)}(\ell_{\mathcal{N}}([\![\mathcal{N}']\!]^\iota) \odot \ell_{\mathcal{N}}([\![\mathcal{N}'']\!]^\iota)) \tag{4}$$

As a consequence, we can compute the component factor of $\mathcal{N}$ without passing via $[\![\mathcal{N}]\!]^\iota$:

▶ **Theorem 49.** *Let $\mathcal{N}$ be an atomic proof-net with conclusions Δ. We have: $\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^\iota) = \pi_{\ell_{\mathcal{N}}(\Delta)}(\bigodot_{\mathsf{b}\in\mathsf{b}(\mathcal{N})} \ell_{\mathcal{N}}(\iota(\mathsf{b})))$.*

▶ **Example 50.** Consider an atomic proof-net $\mathcal{N}$ of conclusions Δ such that no edge in Δ is connected in $\mathcal{N}^{\neg\mathsf{b}}$ with a conclusion of a box of $\mathcal{N}$. By Remark 22, $\pi_{\ell_{\mathcal{N}}(\Delta)}(\bigodot_{\mathsf{b}\in\mathsf{b}(\mathcal{N})} \iota(\mathsf{b}))$ is the constant function giving the total mass of the vectors associated with the boxes of $\mathcal{N}$.

Consider the atomic proof-net $\mathcal{N}_a$ obtained by removing the tensor node from the proof-net $\mathcal{N}_0$ of Figure 1e. If we apply Theorem 49 to $\mathcal{N}_a$, we will obtain exactly the same summation given in Example 11, in fact the edge and component set-families of the conclusions of $\mathcal{N}_a$ are the same. However, we have now the correct formalism to apply exact inference algorithms to refactor the expression $\pi_{\ell_{\mathcal{N}}(\Delta)}(\bigodot_{\mathsf{b}\in\mathsf{b}(\mathcal{N})} \ell_{\mathcal{N}}(\iota(\mathsf{b})))$, by taking advantage of the distributivity law of factor product over the projection (Proposition 27). We adapt here one among the simplest such algorithms, called the sum-product variable elimination algorithm, first introduced in [29], see [8] as a reference. The terminology "variable elimination" is because this procedure infers from a Bayesian network the marginal distribution of a random variable X out of a family[5] $\mathbb{X}$ of variables containing X, by "eliminating" all the other variables in $\mathbb{X}$. In our case, what we "eliminate" are the $\mathcal{N}^{\neg\mathsf{b}}$ components of the conclusions of the box factors containing no conclusion of the proof-net.

Algorithm 1 MLL Sum-Product Algorithm.

```
input:
 1: $\mathcal{N}$                      ▷ an atomic proof-net of conclusions $\Delta$
 2: $\iota$                                          ▷ a valuation map
 3: $\omega$    ▷ A linear order on the components in $\mathsf{Fam}^\iota_c(\mathcal{N})$ not having a conclusion in $\Delta$
output: the factor $\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^\iota)$
 4: $F \leftarrow \{\ell_{\mathcal{N}}(\iota(\mathsf{b})) \mid \mathsf{b} \in \mathsf{b}(\mathcal{N})\}$            ▷ Factors of $\mathsf{b}(\mathcal{N})$
 5: for $C$ in $\omega$ do
 6:     $F_c \leftarrow \{\phi \in F \mid C \in \mathcal{I}(\mathsf{Fam}(\phi))\}$
 7:     $\psi \leftarrow \bigodot_{\phi\in F_c} \phi$                               ▷ Product
 8:     $\rho \leftarrow \pi_{\mathsf{Fam}(\psi)\setminus\{C\}}(\psi)$                         ▷ Sum-out
 9:     $F \leftarrow \{\rho\} \cup (F \setminus F_c)$
    return $\bigodot_{\phi\in F} \phi$
```

Algorithm 1 is our adaptation of the sum-product algorithm. Given a linear order ω over the connected components of $\mathcal{N}^{\neg\mathsf{b}}$ which contains no conclusion of $\mathcal{N}$, the algorithm proceeds as follows: line 4 initialises a variable F with the set of factors to compute; line 5

[5] Any resemblance to the notations in Section 4 is purely voluntary.

takes from ω the next connected component C to process; line 6 gathers in a variable F_c all factors in F which have C as an index (i.e. a conclusion in C); line 7 computes the product of these factors and line 8 projects it on the components different from C (a.k.a. summing out C); line 9 updates F by replacing the processed factors with the result of this projection and then it jumps back to line 5. At the end of this loop, F contains a set of factors indexed over the components of $\mathcal{N}^{\neg \mathsf{b}}$ connected with the conclusions in $\mathcal{N}$ and then it returns their product.

Soundness follows from Proposition 27 and Theorem 49:

▶ **Theorem 51.** *Algorithm 1 returns $\ell_{\mathcal{N}}(\llbracket \mathcal{N} \rrbracket^{\iota})$ if fed with an atomic proof-net $\mathcal{N}$, a valuation ι and a linear order on the components in $\mathsf{Fam}^{\iota}_{c}(\mathcal{N})$ not containing any conclusion of $\mathcal{N}$.*

▶ **Example 52.** Consider the atomic proof-net $\mathcal{N}_a$ obtained by removing the tensor node from the proof-net $\mathcal{N}_0$ of Figure 1e (Example 1) and use the numbers $1, 2, 3, 4, 5$ to denote the five connected components of ${\mathcal{N}_a}^{\neg \mathsf{b}}$ such that component i is supported by variable X_i. The components to eliminate are $1, 2, 3$. By taking the order $\omega = 1 < 2 < 3$, Algorithm 1 will calculate the following intermediate factors: $\rho_1 = \pi_{\{2,3\}}(\iota(\mathsf{b}_1) \odot \iota(\mathsf{b}_2) \odot \iota(\mathsf{b}_3))$, $\rho_2 = \pi_{\{2,4\}}(\rho_1 \odot \iota(\mathsf{b}_4))$, $\rho_3 = \pi_{\{4,5\}}(\rho_2 \odot \iota(\mathsf{b}_5))$, the output being ρ_3. This yields exactly the factored equation in Example 11 and it allows to calculate the whole semantics of $\mathcal{N}_a$ in $O(k^3)$ basic operations, if k is the maximal cardinality of the sets associated with the propositional variables appearing in the proof-net.

▶ Remark 53. A run of Algorithm 1 depends on the chosen order ω. Different orders yield different factorisations and have different performances. For example, by taking the inverse order $3 < 2 < 1$ in Example 52 we get a run in $O(k^4)$, which is an order of magnitude slower than $1 < 2 < 3$, although yet more efficient than the immediate recursive algorithm induced by the standard semantics (Example 11).

In general, Algorithm 1 is in $O(nk^w)$, where n is the length of ω (i.e. the number of components to eliminate), k is the maximal cardinality of a set interpreting an atomic variable (in our examples we always suppose $k = 2$, for the two booleans) and w is the maximal cardinality of $\mathrm{Fam}(\phi)$, for ϕ a factor created/used by the algorithm (this parameter depends on the chosen order ω).

The quest for optimal orders is a major topic in Bayesian networks, which is however known to be a NP-hard problem [4]. Since probabilistic MLL contain Bayesian networks, we should focus on heuristics that yield good performances in most cases.

▶ Remark 54. Recall the proof-net $\mathcal{N}$ in Figure 1e which cut reduces to $\mathcal{N}_0$. Notice that this proof-net does not resemble to a Bayesian network, e.g. it alternates par and tensor nodes. However, the reader may recognise the intermediate factors ρ_1, ρ_2 and ρ_3 computed by Algorithm 1 in Example 52 as the nested sub proof-nets of $\mathcal{N}$ of conclusions, respectively, $X_2^+ \otimes (X_2^- \parr X_3^+)$, $X_2^+ \otimes X_4^+$ and $X_4^+ \otimes X_5^+$. This is far from being a coincidence, as any run of Algorithm 1 can in fact be associated with a MLL proof-net, although this latter might need formulas with an arbitrary number of alternations between tensors and pars. We will investigate this point in a forthcoming paper.

▶ Remark 55. The component renaming $\ell_{\mathcal{N}}$ is omitted in the setting of Bayesian networks as encoded in the formula labelling, by imposing the following type constraint:

($\star$) any two edges of $\mathcal{N}$ which are supported by the same propositional variable lay in the same connected component of $\mathcal{N}^{\neg \mathsf{b}}$.

If $(\star)$ holds (e.g. as for the proof-net $\mathcal{N}_a$ discussed in Example 52), then $\ell_{\mathcal{N}}$ is equivalent to the renaming from the edge set-family to the variable set-family, an instance being given by the renaming f mentioned in Example 33. Such a shortcut is however misleading in our setting, as the rules generating MLL proof-nets (Figure 1b) are more "granular" than the ones for Bayesian networks, in particular $(\star)$ is not preserved by the $\mathrm{c}(\mathcal{N}_{X^-,X^-})$ rule (Example 41). A better alternative would be to introduce "term" variables, like the variables of simply typed λ-calculus, decorating edges.

6 The General Case

The results of the previous section can be extended to non-atomic proof-nets by using MLL cut-reduction. We just sketch here the main ideas, giving the details in the appendix. The reader can recall the proof-net $\mathcal{N}$ in Figure 1e to follow the reasoning with an example. Given a MLL proof-net $\mathcal{N}$ of conclusion Δ: (i) reduce $\mathcal{N}$ to its normal form $\mathcal{N}_0$ by using the cut-reduction rules of Figure 1d. (ii) Decompose $\mathcal{N}_0$ into the syntax forest $\mathcal{F}_\Delta$ of its conclusions and the atomic sub proof-net $\mathcal{N}_a$ of conclusions the atomic formulas $\mathsf{At}(\Delta)$ appearing in Δ, which are the leaves of $\mathcal{F}_\Delta$. Notice that there is a bijection between $[\![\Delta]\!]$ and $[\![\mathsf{At}(\Delta)]\!]$, relating an element in $\vec{d} \in [\![\Delta]\!]$ with a tuple $\mathsf{At}(\vec{d}) \in [\![\mathsf{At}(\Delta)]\!]$ enumerating the atomic components of $\vec{d}$. (iii) Apply Algorithm 1 in order to compute $\ell_{\overline{\mathcal{N}}}([\![\overline{\mathcal{N}}]\!])$. We have:

▶ **Corollary 56.** *Let $\mathcal{N}$ be a proof-net with conclusions Δ, and let $\mathcal{N}_0$ be the normal form of $\mathcal{N}$ and $(\mathcal{F}_\Delta, \mathcal{N}_a)$ be the decomposition of $\mathcal{N}_0$ described above. For every $\vec{d} \in [\![\Delta]\!]$, we have:*

$$[\![\mathcal{N}]\!]^{\iota}_{\vec{d}} = \ell_{\mathcal{N}}([\![\mathcal{N}_a]\!])_{\ell^{\bullet}_{\mathcal{N}_a}(\mathsf{At}(\vec{d}))}, \tag{5}$$

if $\mathsf{At}(\vec{d})$ agrees on $\ell_{\mathcal{N}_a}$, otherwise $[\![\mathcal{N}]\!]^{\iota}_{\vec{d}} = 0$.

The cut-reduction in step (i) is linear in the size of $\mathcal{N}_0$ as MLL cut-reduction shrinks the size of a proof-net. Also the construction of $\mathcal{N}_a$ out of $\mathcal{N}_0$, and the read of $\ell^{\bullet}_{\mathcal{N}_a}(\mathsf{At}(\vec{d}))$ out of $\vec{d} \in [\![\Delta]\!]$ are linear. So all the complexity of this procedure is the calculation of $\ell_{\mathcal{N}}([\![\mathcal{N}_a]\!])$ which has been discussed in the previous section.

7 Conclusion and Perspectives

We considered weighted relational semantics just as an instance of quantitative semantics, but these techniques can be applied verbatim to other web-based semantics, such as probabilistic coherence spaces [6] or finiteness or Köthe sequence spaces [11, 10].

One can wonder whether our results extend to richer linear logic fragments. The additive connectives $\oplus$ and & can be reasonably added to the picture. In fact, by adopting some form of additive boxes [9], one can revisit the sum-product algorithm as a refactorization of a proof-net modulo commutative additive cuts. The exponential modalities (so encompassing full simply typed probabilistic λ-calculus) are more challenging as they require infinite sets at the semantical level. This will deserve future investigation.

Related to the above point is the correspondence alluded to in Remark 54. We will detail in a future work how to map any run ρ of the sum-product algorithm on an atomic proof-net $\mathcal{N}_0$ into a non-atomic proof-net $\mathcal{N}_\rho$, which rewrites into $\mathcal{N}_0$ and such that the intermediate factors appearing in ρ correspond to sub-proof-nets of $\mathcal{N}_\rho$.

As mentioned by Remark 53, the performance of many exact inference algorithms, such as sum-product, depends on the order of the components to eliminate and the problem of finding optimal orders is known to be NP-hard [4]. Many heuristics have been given based on the graph-theoretical structure of Bayesian nets. One can wonder whether the additional proof-theoretical structure (e.g. switching paths, empires [18]) can suggest new heuristics.

References

1 Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. Linear lambda-calculus and categorical models revisited. In E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M. Richter, editors, *Proceedings of the Sixth Workshop on Computer Science Logic*, pages 61–84. Springer Verlag, 1993. URL: `citeseer.ist.psu.edu/benton92linear.html`.

2 Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics: the exponentials. *Ann. Pure Appl. Logic*, 109(3):205–241, 2001.

3 Kenta Cho and Bart Jacobs. Disintegration and bayesian inversion via string diagrams. *Math. Struct. Comput. Sci.*, 29(7):938–971, 2019. `doi:10.1017/S0960129518000488`.

4 Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2):393–405, 1990. `doi:10.1016/0004-3702(90)90060-D`.

5 Raphaëlle Crubillé. Probabilistic stable functions on discrete cones are power series. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 275–284. ACM, 2018. `doi:10.1145/3209108.3209198`.

6 Vincent Danos and Thomas Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Information and Computation*, 209(6):966–991, 2011.

7 Adnan Darwiche. Bayesian networks. In Frank van Harmelen, Vladimir Lifschitz, and Bruce W. Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 467–509. Elsevier, 2008. `doi:10.1016/S1574-6526(07)03011-8`.

8 Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009. URL: `http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521884389`.

9 Lorenzo Tortora de Falco. The additive mutilboxes. *Ann. Pure Appl. Log.*, 120(1-3):65–102, 2003. `doi:10.1016/S0168-0072(02)00042-8`.

10 Thomas Ehrhard. On Köthe sequence spaces and linear logic. *Math. Struct. Comput. Sci.*, 12:579–623, 2002.

11 Thomas Ehrhard. Finiteness spaces. *Math. Struct. Comput. Sci.*, 15(4):615–646, 2005.

12 Thomas Ehrhard. Differentials and distances in probabilistic coherence spaces. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.FSCD.2019.17`.

13 Thomas Ehrhard. Cones as a model of intuitionistic linear logic. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 370–383. ACM, 2020. `doi:10.1145/3373718.3394758`.

14 Thomas Ehrhard, Michele Pagani, and Christine Tasson. Probabilistic Coherence Spaces are Fully Abstract for Probabilistic PCF. In P. Sewell, editor, *The 41th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL14, San Diego, USA*. ACM, 2014. `doi:10.1145/2535838.2535865`.

15 Thomas Ehrhard, Michele Pagani, and Christine Tasson. Full abstraction for probabilistic pcf. *J. ACM*, 65(4), April 2018. `doi:10.1145/3164540`.

16 Thomas Ehrhard, Michele Pagani, and Christine Tasson. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. *PACMPL*, 2(POPL):59:1–59:28, 2018. `doi:10.1145/3158147`.

17 Thomas Ehrhard and Christine Tasson. Probabilistic call by push value. *Log. Methods Comput. Sci.*, 15(1), 2019. `doi:10.23638/LMCS-15(1:3)2019`.

18 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

19 Jean-Yves Girard. A new constructive logic: classical logic. *Math. Struct. Comput. Sci.*, 1(3):255–296, 1991.

20 Jean-Yves Girard. Coherent banach spaces: a continuous denotational semantics. *Theor. Comput. Sci.*, 227:297, 1999.

21 Jean-Yves Girard. Between logic and quantic: a tract. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Philip Scott, editors, *Linear Logic in Computer Science*, volume 316 of *London Math. Soc. Lect. Notes Ser.* CUP, 2004.

22 Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal inference by string diagram surgery. In Mikolaj Bojanczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 2019. doi:10.1007/978-3-030-17127-8_18.

23 Yves Lafont. From proof nets to interaction nets. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Math. Soc. Lect. Notes Ser.*, pages 225–247, 1995.

24 Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013.* IEEE Computer Society, June 2013.

25 François Lamarche. Quantitative domains and infinitary algebras. *Theor. Comput. Sci.*, 94(1):37–62, 1992. doi:10.1016/0304-3975(92)90323-8.

26 Hugo Paquet. Bayesian strategies: probabilistic programs as generalised graphical models. In Nobuko Yoshida, editor, *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12648 of *Lecture Notes in Computer Science*, pages 519–547. Springer, 2021. doi:10.1007/978-3-030-72019-3_19.

27 Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference.* Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

28 Dario Stein and Sam Staton. Compositional semantics for probabilistic programs with exact conditioning. *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2021.

29 N. Zhang and D. Poole. A simple approach to bayesian network computations. In *Proceedings of the 10th Biennial Canadian Artificial Intelligence Conference*, pages 171–178. AAAI Press / The MIT Press, 1994.