First order linear logic and tensor type calculus for categorial grammars

Sergey Slavnov
National Research University Higher School of Economics
sslavnov@yandex.ru
January 3, 2022

Abstract

We study relationship between first order multiplicative linear logic (MLL1), which has been known to provide representations to different categorial grammars, and the recently introduced extended tensor type calculus (ETTC). We identify a fragment of MLL1, which seems sufficient for many grammar representations, and establish a correspondence between ETTC and this fragment. The system ETTC, thus, can be seen as an alternative syntax and intrinsic deductive system together with a geometric representation for the latter. We also give a natural deduction formulation of ETTC, which might be convenient.

1 Introduction

The best known examples of categorial grammars are Lambek grammars, which are based on Lambek calculus (LC) [5], i.e. noncommutative intuitionistic linear logic (for background on linear logic see [2],[3]). These, however, have somewhat limited expressive power, and a lot of extensions/variations have been proposed, using discontinuous tuples of strings and λ-terms, commutative and non-commutative logical operations, modalities etc, let us mention displacement grammars [9], abstract categorial grammars (also called λ-grammars and linear grammars) [1],[10],[11] and hybrid type logical categorial grammars [4].

It has been known for a while (starting from the seminal work [8]) that different grammatical formalisms, such as those just mentioned, can all be represented using simple and familiar commutative system of first order multiplicative intuitionistic linear logic (MILL1) [6],[7]. In fact, not the whole of MILL1 is used: it can be noted that translations of different categorial grammars usually fit into some small fragment, which, therefore, can be given linguistic interpretation. Unfortunately, we do not have any deductive system intrinsic to this fragment. Typically, when deriving the MILL1 translation of an LC sequent in sequent calculus or natural deduction, one might have to use at intermediate steps sequents and derivations which are not translations of anything and have no linguistic meaning at all (as it seems).
Extended tensor type calculus (ETTC), a system extending propositional (classical) multiplicative linear logic (MLL), recently proposed by the author [13] (elaborating on [12]), was directly designed for linguistic interpretation in terms of bipartite graphs whose edges are labeled with words. It has been shown that tensor grammars based on ETTC include both Lambek grammars and abstract categorial grammars as conservative fragments, with representation very similar to that in MILL1. However, unlike the case of MILL1, in ETTC derivations each rule corresponds to a concrete operation on strings, so that formal language generation is decomposed into elementary steps, which, moreover, can be conveniently visualized in the pictorial setting of labeled graphs.

In this work we identify a fragment (we call it strictly balanced fragment) of first order linear logic, which is sufficient for linguistic constructions of [6], [7], and show that there are mutually inverse translations to and from ETTC. Thus ETTC turns out to be an alternative syntax for the relevant fragment of MILL1, equipped with an intrinsic deductive system and intuitive pictorial representation.

We also introduce a natural deduction formulation for ETTC in this work, which might be more convenient in some situations than the sequent calculus formulation of [13].

2 Background: systems of linear logic

The language of multiplicative intuitionistic logic (MILL1) is summarized in Figure 1a. We assume that we are given a set Pred of predicate symbols with assigned arities, a countable set Var of individual variables and a set Cons of constants. The set of first order atomic formulas is denoted as At, and the set of all first order linear intuitionistic formulas is denoted as Fm. The binary connectives ⊗, ⊸ are called respectively tensor (also times) and linear implication. A context Γ is a finite multiset of formulas; as usual, we denote formulas with Latin letters and contexts with Greek letters. The set of free variables in the context Γ is denoted as FV(Γ). The sequent calculus for MILL1 is shown in Figure 1b. We will use notation Γ ⊢ MILL1 C to indicate that the sequent Γ ⊢ C is derivable in MILL1 and a similar convention for other systems considered in this work.

The language of classical first order multiplicative linear logic (MLL1) is summarized in Figure 1c. We assume that we are given a set Pred+ of positive predicate symbols with assigned arities, and sets Var and Cons of variables and constants. Then the set Pred− of negative predicate symbols and the set Pred of all predicate symbols are defined. Atomic formulas are defined same as for MILL1. Formulas are built from the binary connectives ⊗, ⊸ and quantifiers, the connective ⊴ is called cotensor (also par). Linear negation (.) is not a connective, but is definable by induction on formula construction, as in Figure 1c. Note that, somewhat non-traditionally, we follow the convention that negation flips tensor/cotensor factors, typical for non-commutative systems. This does not change the logic (the formulas A ⊗ B and B ⊗ A are provably equivalent), but is more consistent with our intended geometric interpretation.

Contexts are defined same as for MILL1. The sequent calculus for MILL1 is shown
Figure 1: Systems of linear logic
in Figure 1d. Translation in Figure 1e identifies **MILL1** as a conservative fragment of **MLL1**.

The language and sequent calculus formulation of **Lambek calculus** (LC) are summarised in Figure 1f. Formulas are built from a set **Prop** of propositional symbols, and contexts are sequences, rather than multisets, of formulas.

Given two variables or constants $l, r$, the first order translation $||F||^{(l,r)}$ of an LC formula $F$ parameterized by $l, r$ is shown in Figure 1g (LC propositional symbols are treated as binary predicate symbols). This embeds LC into **MILL1** as a conservative fragment [8].

### 3 **MILL1** grammars and strictly balanced fragment

Translation from LC suggests defining **MILL1** grammars similar to Lambek grammars.

Let a finite alphabet $T$ of terminal symbols be given. Assume also that our first order language contains all integer constants and two special constants $l, r$.

Let us say that a **MILL1** lexical entry is a pair $(w, A)$, where $w \in T^*$ is nonempty, and $A$ is a **MILL1** formula with one occurrence of $l$ and one occurrence of $r$ and no other constants or free variables. For the formula $A$ occurring in a simple lexical entry we will write $A[l; r] = A[l := x_1, r := y_1]$ (so that $A = A[l; r]$).

A **MILL1** grammar $G$ is a pair $(\text{Lex}, S)$, where Lex is a finite set of **MILL1** lexical entries, and $S$ is a binary predicate symbol. The language $L(G)$ generated by $G$ or, simply, the language of $G$ is defined as

$$L(G) = \{ w_1 \ldots w_n | (w_1, A_1), \ldots, (w_n, A_n) \in \text{Lex}, A_1[0; 1], \ldots, A_n[n-1; n] \vdash_{\text{MILL1}} S(0, n) \}.$$ 

It seems clear that, under such a definition, Lambek grammars translate to **MILL1** grammars generating the same language.

It has been shown [6], [7] that **MILL1** allows representing more complex systems such as displacement calculus, abstract categorial grammars, hybrid type-logical grammars. This suggests that the above definition should be generalized to allow more complex lexical entries, corresponding to word tuples rather than just words. On the other hand, it seems evident that in translations from grammatical formalisms we may restrict to smaller fragments of **MILL1**.

### 3.1 Balanced sequents

We equip the first order language with an extra structure. We say that the language is balanced if each $n$-ary predicate symbol $p$ is equipped with a valency, which is a pair $(k, m)$ of nonnegative integers with $k + m = n$. We indicate it in notation by writing corresponding atomic formulas as $p(x_1, \ldots, x_k; y_1, \ldots, y_m)$, with first $k$ entries separated by a semicolon. In this setting, we say that $x_1, \ldots, x_k$ are left occurrences, or have left polarity, notation $\text{sgn}(x_i) = -1$, and $y_1, \ldots, y_m$ are right occurrences, $\text{sgn}(y_i) = +1$. We extend the notion of occurrence polarity to compound formulas and sequents by induction.
In the intuitionistic language the definition is as follows. For an occurrence \( x \) in an immediate subformula \( A \) of a compound formula \( F \), the polarity \( sgn_F(x) \) of \( x \) in \( F \) is defined by \( sgn_F(x) = -sgn_A(x) \) if \( F = A \to B \) and \( sgn_F(x) = sgn_A(x) \) otherwise. For an occurrence \( x \) in a formula \( F \) of a sequent \( \Gamma \vdash C \), the polarity \( sgn_{\Gamma;C}(x) \) of \( x \) in the sequent is defined by \( sgn_{\Gamma;C}(x) = -sgn_F(x) \), if \( F \in \Gamma \), and \( sgn_{\Gamma;C}(x) = sgn_F(x) \) if \( F = C \).

In the case of \( \text{MLL}_1 \) we require that valencies of dual predicate symbols be consistent, so that \( p(x_1, \ldots , x_k; y_1, \ldots , y_m) = \overline{p}(y_m, \ldots , y_1; x_k, \ldots , x_1) \). Then the polarity of an occurrence in a formula or a sequent is just the same as its polarity in the corresponding atomic formula. Obviously, the embedding of \( \text{MILL}_1 \) to \( \text{MLL}_1 \) preserves polarities.

We say that a first order formula (context, sequent) is balanced if every quantifier binds exactly one left and one right variable occurrence. A balanced formula (context, sequent) is strictly balanced if, furthermore, it has at most one left and at most one right occurrence of any free variable or constant. It is immediate that if a strictly balanced sequent is derivable (in \( \text{MLL}_1 \) or \( \text{MLL}_1 \)) then every free variable or constant occurring in it has exactly one occurrence of each polarity. We will say that a \( \text{MLL}_1 \) grammar is balanced if the formula in every lexical entry is strictly balanced, with constants \( l \) and \( r \) occurring with left and right polarity respectively.

It is immediate that if we treat \( \text{LC} \) propositional symbols as predicate symbols of valency \((1,1)\), then the translation in Figure 1g indeed uses only the strictly balanced fragment. Similar observations apply to translations in [6], [7].

### 3.2 Occurrence nets

Below we are discussing derivations of balanced and strictly balanced sequents in \( \text{MLL}_1 \). The discussion can be adapted to the intuitionistic case using the standard embedding. In the following we will say simply occurrence meaning occurrence of a variable or a constant.

An occurrence net of a balanced \( \text{MLL}_1 \) sequent \( \vdash \Gamma \) is a perfect matching \( \sigma \) between left and right free occurrences in \( \Gamma \), such that each pair (link) in \( \sigma \) consists of occurrences of the same variable (constant). Note that for a strictly balanced sequent there is only one occurrence net possible. Basically, occurrence nets are rudiments of proof-nets. To each cut-free derivation \( \pi \) of \( \vdash \Gamma \) we assign by induction an occurrence net \( \sigma(\pi) \).

For an axiom \( \vdash \overline{X}, X \), where \( X = p(e_1, \ldots, e_n) \), the net is defined by matching an occurrence \( e_i \) in \( X \) with the occurrence \( e_j \) in \( \overline{X} = \overline{p}(e_n, \ldots, e_1) \). For \( \pi \) obtained from a derivation \( \pi' \) by the (3) rule, we put \( \sigma(\pi) = \sigma(\pi') \). For \( \pi \) obtained from derivations \( \pi_1, \pi_2 \) by the (⊗) rule, \( \sigma(\pi) = \sigma(\pi_1) \cup \sigma(\pi_2) \). If \( \pi \) is obtained from some \( \pi' \) by the (V) rule introducing a formula \( \forall x A \), then the variable \( x \) has no free occurrences in the
might be desirable.

quent calculus is not very natural for such a system, and some other representation

strictly balanced fragment. It seems though that the usual syntax of first order se-

variable (constant)

Γ

Let

Proposition 3.1

A strictly balanced sequent

derivations

and the (∃) rule does the same gluing as (3); only vertex labels are changed.

We will say that derivations of strictly balanced sequents using rules of MLL1

Lemma 3.2

A strictly balanced sequent ⊢ Θ derivable in MLL1 is derivable with a

strictly balanced derivation.

Proof by induction on a cut-free derivation. If ⊢ Θ is the conclusion of the (3) rule

applied to non-strictly balanced premise, then, using Proposition 3.1 we replace the

premise with a strictly balanced sequent and obtain ⊢ Θ by the (3) rule.

Thus, adding the (3) rule, we obtain a kind of intrinsic deductive system for the

strictly balanced fragment. It seems though that the usual syntax of first order sequen-
calculus is not very natural for such a system, and some other representation might be desirable.

The following is easily proved by induction on derivation.

Proposition 3.1 Let π be a cut-free derivation of a balanced sequent ⊢ Γ, and assume

that (x_l, x_r) ∈ σ(π). Let e be a fresh variable or constant and Γ′ = Γ|e_l, e_r := e.

Then ⊢ Γ′ is derivable in MLL1 with a derivation π′ of the same size as π and

σ(π′) = σ(π) \{ (e_l, e_r) \} ∪ \{ (e_l′, e_r′) \}.

3.3 Strictly balanced derivations

Note that for all sequent rules except (3), if the conclusion is strictly balanced so are

the premises. We will supply an admissible rule (3′), alternative to (3), which applies
to strictly balanced sequents only and, therefore, satisfies the same property.

So assume that we have a derivable strictly balanced sequent ⊢ Θ, where Θ = Γ, A. Assume that A has a left free occurrence s_l and a right free occurrence t_r of a
variable (constant) s, t respectively, with s ≠ t. Let e be a fresh variable or constant.

Let Θ′ = Θ|s := e. By Proposition 3.1 the sequent ⊢ Θ′ is derivable.

Now, let us write Θ′ = Γ′A′, where A′ is the image of A in Θ′. Let e_l, e_r be the
occurrences of e in A′ that replace s_l and t_r respectively. Then the strictly balanced
sequent ⊢ Θ′′, where Θ′′ = Γ′, ∃xA′|e_l, e_r := x, is derivable from ⊢ Θ′ by the (3) rule.

We say that ⊢ Θ′′ is obtained from ⊢ Θ by the (3′) rule. Note that, on the level of occurrence nets, seen as bipartite graphs, the (3′) rule does the same gluing as (3); only vertex labels are changed.

We will say that derivations of strictly balanced sequents using rules of MLL1

and the (3′) rule and involving only strictly balanced sequents are strictly balanced
derivations.
4 Tensor type calculus

4.1 Tensor terms

We assume that we are given an infinite set \( \text{Ind} \) of indices. They will be used in all syntactic objects (terms, types, typing judgements) that we consider.

Let \( T \) be an alphabet of terminal symbols. The set \( \overline{TmEx} \) of provisional tensor term expressions is the free commutative monoid generated by the set

\[
\{[w]_i^j \mid i, j \in \text{Ind}, w \in T^* \} \cup \{[w] \mid w \in T^* \},
\]

and the set \( \overline{Tm} \) of provisional tensor terms is the quotient of \( \overline{TmEx} \) by the relations

\[
[w]_i^j \cdot [v]_k^j = [uv]_i^k, \quad [w]_i^i = [w], \quad [a_1 \ldots a_n] = [a_n a_1 \ldots a_{n-1}] \quad \text{for } a_1, \ldots, a_n \in T
\]

(we write the monoid operation multiplicatively and the monoid unit as 1). A tensor term expression is a provisional tensor term expression in which any index has at most one occurrence as an upper one and at most one occurrence as a lower one. The set \( Tm \) of tensor terms is the image of the set \( TmEx \) of term expressions in \( \overline{Tm} \) under the quotient map. (The adjective “tensor” will often be omitted in the following.)

Elements of generating set \((1)\) are elementary term expressions. Elementary expressions of the form \([w]_i^j\) with \( i \neq j \) are regular term expressions. A tensor term is regular if it is the image of the product of regular term expression. Otherwise the term is singular.

We think of regular terms as bipartite graphs whose vertices are labeled with indices and edges with words, and direction of edges is from lower indices to upper ones. Singular terms such as \([w]_i^i\) correspond to closed loops (with no vertices) labeled with cyclic words (this explains the last relation in \((2)\)). Thus, a regular term expression \([w]_i^j, i \neq j\), corresponds to a single edge from \( i \) to \( j \) labeled with \( w \), the product of two term expressions without common indices is the disjoint union of the corresponding graphs, and a term expression with repeated indices corresponds to a graph obtained by gluing edges along matching vertices. The monoidal unit 1 corresponds to the empty graph. As for singular terms, they arise when edges are glued cyclically.

We say that an index occurring in a term expression \( t \) is free in \( t \) if it occurs in \( t \) once. Otherwise we say that the index is bound. We say that a term expression is normal if it has no bound indices. The sets of free upper and lower indices of a term expression are invariant under the quotient map to \( Tm \), so they are well-defined for terms as well. We denote the set of free upper, respectively lower, indices of a tensor term \( t \) as \( FSup(t) \), respectively \( FSub(t) \).

The geometric representation makes especially obvious that any tensor term \( t \) is the image of a normal term expression and can be written as the product \( t = \text{reg}(t) \cdot t' \), where \( \text{reg}(t) \) is regular and \( t' \) is 1 or the product of elementary singular expressions. We say that \( \text{reg}(t) \) is the regular part of \( t \).

Especially important are terms of the form \( \delta_i^j = [\epsilon]_i^j \), where \( \epsilon \) denotes the empty word. We will also use the notation \( \delta_{i_1 \ldots i_n}^{j_1 \ldots j_n} = \delta_{i_1}^{j_1} \cdots \delta_{i_n}^{j_n} \).
We call terms of the above format Kronecker deltas. We will adopt the convention that capital Latin letters stand for sequences of indices and small Latin letters stand for individual indices. Typically, a Kronecker delta will be written concisely as $\delta^I_J$.

Multiplication by Kronecker deltas amounts to renaming indices. If $t$ is a tensor term expression with $i \in F_{\text{Sup}}(t)$, $j \in F_{\text{Sub}}(t)$, and $i', j' \notin \text{Ind}(t)$, then $\delta^i_{i'} \cdot t$ is the term $t$ with $i$ changed to $i'$, and $\delta^j_{j'} \cdot t$ is $t$ with $j$ changed to $j'$.

There is also a “divergent” delta $\delta = \delta^i_i = [e]$ that corresponds to a closed loop with no label.

### 4.2 Tensor types

Our goal is to assign types to tensor terms.

The set $\overline{Tp}$ of provisional tensor types is built according to the grammar in Figure 3a. We assume that we are given a set $\text{Lit}^+$ of positive atomic type symbols or positive literals, and every element $p \in \text{Lit}^+$ is assigned a valency $v(p) \in \mathbb{N}^2$. Then the set $\text{Lit}^-$ of negative atomic type symbols and the set $\text{Lit}$ of all atomic type symbols are defined. The convention for negative atomic symbols is that $v(\overline{p}) = (n, m)$ if $v(p) = (n, m)$. Duality $(\cdot)$ is not a connective or operator but is definable.

The symbols $\nabla, \Delta$ are binding operators. These bind indices exactly in the same way as quantifiers bind variables. The operator $Q \in \{\nabla, \Delta\}$ in front of an expression $Q^i_j A$ has $A$ as its scope and binds all lower, respectively, upper occurrences of $i$, respectively, $j$ in $A$ that are not already bound by some other operator. A tensor type is a provisional tensor type in which any index has at most one free occurrence (no matter, upper or lower) and every binding operator binds exactly one lower and one upper index occurrence. We will denote the set of upper, free upper, lower, free lower indices occurring in a type $A$ as $\text{Sup}(A)$, $F_{\text{Sup}}(A)$, $\text{Sub}(A)$, $F_{\text{Sub}}(A)$ respectively. We say that tensor types are $\alpha$-equivalent if they differ by renaming bound indices in a usual way.

We also define tensor type symbols as, basically, $\alpha$-equivalence classes of tensor types with all free indices erased. That is, tensor type symbol is an equivalence class for the smallest equivalence relation on tensor types that identifies any two $\alpha$-equivalent types and any two types obtained from each other by renaming a free index. Valency $v(a)$ of a type symbol $a$ is a pair $(n, m)$, where $n$ and $m$ are the numbers of, respectively, free upper and free lower indices in corresponding types.

Usually we will denote types with capital letters and type symbols with small letters. A tensor type, up to $\alpha$-equivalence, can be recovered from its symbol by specifying the sequences of its upper and lower free indices. Accordingly, we will write $A = a^i_j$ to indicate that $A$ is a type whose symbol is $a$, and sequences of free upper and lower indices are $I$ and $J$ respectively. Sometimes we will have enumerations of type symbols, such as $a^{(1)}, \ldots, a^{(n)}$. In such cases we will put brackets around subscripts in order to avoid confusion with tensor type indices.

A tensor type context is a finite set of tensor types whose elements have no common free indices. We extend the notation for sets of upper, free upper, lower and free lower indices from types to type contexts in the obvious way and write $\text{Sup}(\Gamma) = \bigcup_{A \in \Gamma} \text{Sup}(A)$ etc.
\[ \text{Lit}_t = \{ p | p \in \text{Lit}_t \}, \quad \overline{\text{p}} = p \text{ for } p \in \text{Lit}_t, \quad \text{Lit} = \text{Lit}_t \cup \text{Lit}_t_+ \]

\[ \overline{\text{At}}_t = \{ p_1^{i_1} \ldots p_m^{i_m} | p \in \text{Lit}, v(p) = (m, n), i_1, \ldots, i_m, j_1, \ldots, j_n \in \text{Ind} \}. \]

\[ \overline{T}p := \text{At}(\overline{T}p \otimes \overline{T}p) (\overline{T}p \otimes \overline{T}p) \overline{\text{V}}_i^j \overline{T}p \overline{\text{D}}_i^j \overline{T}p, \quad i, j \in \text{Ind}, \quad i \neq j. \]

\[ \overline{p}_j^i = \overline{p}_j^i \text{ for } p \in \text{Lit}_t, \text{ where } \overline{T} = (i_n, \ldots, i_1) \text{ for } I = (i_1, \ldots, i_n). \]

\[ \overline{A} \otimes \overline{B} = \overline{B} \otimes \overline{A}, \quad \overline{A} \otimes \overline{B} = \overline{B} \otimes \overline{A}, \quad \overline{\nabla}_i^j \overline{A} = \overline{\Delta}_i^j \overline{A}, \quad \overline{\Delta}_i^j \overline{A} = \overline{\nabla}_i^j \overline{A}. \]

(a) Language

\[ \delta t^\overline{T} \vdash p_j^i, \overline{p}_j^i, \quad p \in \text{Lit}_t (\text{Id}) \]

\[ t \vdash \Gamma, A, B \]

\[ t \vdash \Gamma, A \]

\[ s \vdash B, \Theta \]

\[ t \vdash \Gamma, A \otimes B, \Theta \]

\[ t \vdash \Gamma, s \vdash \Theta \]

\[ t \vdash \Gamma, A \]

\[ \delta^\alpha^\beta^\eta \cdot t \vdash \Gamma, A \]

\[ \delta^\alpha^\beta^\eta \cdot t \vdash \Gamma, A \]

\[ \delta^\alpha^\beta^\eta \cdot t \vdash \Gamma, A \]

\[ \delta^\alpha^\beta^\eta \cdot t \vdash \Gamma, A \]

(b) ETTC typing rules

\[ \Rightarrow [\text{xy}]^i_j \cdot [\text{ba}]^l_k \cdot [\delta]^r_s \]

\[ a_i \otimes b_j^l_i \]

(c) Picture for a typing judgement

(d) Picture for an axiom

(e) Picture for a cut

(f) Picture for a (\nabla) rule

(g) Picture for a (\Delta) rule

(h) Picture for \( a \rightarrow b \)

(i) \( b \rightarrow a \) inside \( a \rightarrow b \)

(j) \( a \backslash b \) inside \( a \rightarrow b \)

\[ a_j^l \rightarrow b_k^l = b_k^l \nabla j, \quad (b/a)^l_j = a^l_j (b_k^l \otimes j), \quad (a \backslash b)^l_j = \nabla^l_j (a_k^l \otimes b_j^l). \]

(k) Encoding different implications in ETTC

Figure 3: ETTC
4.3 Typing judgements and rules

A tensor typing judgement is a pair \((t, \Gamma)\), written as \(t \vdash \Gamma\), where \(\Gamma\) is a tensor type context, and \(t\) is a tensor term, such that \(F\text{Sup}(t) = F\text{Sub}(\Gamma), F\text{Sub}(t) = F\text{Sup}(\Gamma)\).

Typing judgements are derived using the rules of extended tensor type calculus (ETTC) [13] in Figure 3b. (The title “extended” refers in [13] to usage of binding operators, which extend plain types of MLL.)

It is implicit in the rules that all typing judgements are well defined, i.e. there are no index collisions. In the \((\triangledown)\) and \((\Delta)\) rules it is required that \(\alpha \in F\text{Sub}(A), \beta \in F\text{Sup}(A)\). Note that the term \(t\) in the premise of the \((\Delta)\) rule must have free occurrences of \(\alpha\) and \(\beta\) (as an upper and a lower index respectively). In the conclusion, the term \(\delta^\alpha_\beta \cdot t\) has these occurrences bound. Thus free indices to the left and to the right of the turnstile do match, and the typing judgement is well defined.

It is not hard to see that ETTC is cut-free [13]. Note also that the system does not use any terminal alphabet. Terminal symbols will appear in nonlogical axioms, i.e. lexical entries of formal grammars.

Let us define \(\alpha\)-equivalence of tensor typing judgements as the smallest equivalence relation identifying typing judgements that can be obtained from each other by renaming bound indices in types or, for a free index, by replacing both its occurrences to the right and to the left of the turnstile with a fresh one. Then, obviously, derivable typing judgements are closed under \(\alpha\)-equivalence. \(\alpha\)-Equivalent typing judgements are essentially “the same”; nothing is lost even if we identify them as in [13].

4.3.1 Geometric representation and meaning

While tensor terms have a natural interpretation as edge-labeled graphs, tensor typing judgements suggest some particular pictorial representation for such graphs. A type to the right of the turnstile can be seen as inducing an ordering on the set of free indices, for example, from left to right, from top to bottom, and this can be seen as an ordering of vertices in a picture. Thus, a typing judgement \(t \vdash F\) can be depicted as the graph representing the term \(t\) with vertices aligned, say, horizontally according to the ordering induced by \(F\). (Bound indices of \(F\) are not in the picture.) An example of a concrete typing judgement representation is given in Figure 3c. Note that such a representation identifies \(\alpha\)-equivalent judgements, and the indices usually become redundant, at least, in the absence of binding operators.

Some schematic pictures corresponding to rules of ETTC are shown in Figure 3. The \((\Delta)\) rule simply glues together two bound indices/vertices. On the other hand, the \((\triangledown)\) rule is applicable only in the case when the corresponding indices/vertices are connected with an edge carrying no label. Then this edge (together with its endpoints) is erased from the picture completely. The information about the erased edge is stored in the introduced type.

Thus, terms of type \(\nabla^\alpha_\beta A\) encode the subtype of \(A\) consisting of terms/graphs with that specific form: vertices corresponding to \(\alpha\) and \(\beta\) are connected with an edge, and the connecting edge carries no label. It can be observed that we have
the admissible rule $\frac{t \vdash \text{reg}(\vec{\alpha}, \vec{\beta})}{\delta_{\vec{\beta}}^a \cdot t \vdash \text{reg}(\vec{\alpha}, \vec{\beta})} \ (\forall E)$, so that “decoding” from $\text{reg}(\vec{\alpha}, \vec{\beta})A$ to $A$ is always possible.

Let $a, b$ be type symbols of valency $(1, 1)$, so that regular elements of the corresponding types are strings. The implication type symbol $a \to b = b \equiv \alpha b$ has valency $(2, 2)$, and elements of the corresponding type can be interpreted as pairs of strings, as Figure 3h suggests. Now, there are two subtypes consisting of elements of the forms respectively $[u]_{i}^{l} \cdot \delta^{j}_{k}$ and $[u]_{j}^{k} \cdot \delta^{i}_{l}$ corresponding to Figures 3i, 3j. It is easily computed that elements of the first form act (by means of the Cut rule) on elements of $a_{l}^{k}$ by multiplication (concatenation) on the left, and elements of the second form, by multiplication on the right. The two formats identify two subtypes of the implicational tensor type that correspond to two implicational types of Lambek calculus. Which suggests that the implicational types of Lambek calculus should be translated to ETTC as in Figure 3k (compare with Figure 1g).

4.4 Relation with first order logic

Given a balanced first order language, we identify predicate symbols with atomic type symbols of the same valencies. For an $\text{MLL}_1$ strictly balanced context $\Gamma$, let $X$ be the set variables and constants occurring in $\Gamma$ (not necessarily freely) and choose non-intersecting subsets $I, J \subset \text{Ind}$ together with a pair of bijections $\pi : X \rightarrow I, \rho : X \rightarrow J$. Translation of subformulas of $\Gamma$ to tensor types is given by

$$||a(x_1, \ldots, x_n; y_1, \ldots, y_m)||_{\rho \pi} = a_{\pi(x_1)}^{\rho(x_1)} \cdots a_{\pi(x_n)}^{\rho(x_n)}$$

for $a \in N$,

$$||\forall x A||_{\rho \pi} = \text{reg}(\vec{\alpha})_{\pi(x)} \ ||A||_{\rho \pi}$$

and

$$||\exists x A||_{\rho \pi} = \text{reg}(\vec{\alpha})_{\pi(x)} \ ||A||_{\rho \pi}$$

(with binary connectives obviously translating to themselves).

**Lemma 4.1** There is a translation assigning to any strictly balanced derivation of $\vdash \text{reg}(\vec{\alpha})_{\pi(x)} \ ||A||_{\rho \pi}$, where the regularization $\text{reg}(\vec{\alpha})_{\pi(x)}$ is, independent of a particular derivation, the product of all Kronecker deltas $\delta_{\rho \pi(x)}^{\pi(x)}$, where $x$ ranges over all free occurrences in $\Gamma$. Conversely, any derivable tensor typing judgement is obtained as a translation of a strictly balanced $\text{MLL}_1$ derivation.

**Proof** The translation is by induction on strictly balanced derivation. The regular part $\text{reg}(\vec{\alpha})_{\pi(x)}$ encodes the unique occurrence net of $\Gamma$. Axioms and propositional rules translate to themselves. The $(\forall)$ rule translates the $(\forall)$ rule, while the $(\exists)$ and $(\exists')$ rules translate as the $(\Delta)$ rule. An interesting case is that of the $(\exists)$ rule. When applied to a strictly balanced sequent, it erases a link from the occurrence net, just as the $(\forall)$ rule. As for the $(\Delta)$ rule in this case, it glues together the corresponding link endpoints, producing a closed loop, divergent $\delta$. The link is removed from the regular part, but not from the term entirely. The second claim of the lemma is proven by induction on ETTC derivations. □

We observe that, in general, the tensor translation does indeed depend on the derivation, only the regular part of the term is completely determined by the sequent.
5 Tensor natural deduction and grammars

In order to give a natural deduction formulation for ETTC we need to allow variables standing for tensor terms. Thus we introduce a countable set of tensor variable symbols of different valencies (where valency, as usual, is a pair of nonnegative integers) and the convention that if $x$ is a variable symbol of valency $(m, n)$ and $i_1, \ldots, i_m, j_1, \ldots, j_m$ are pairwise distinct indices, then $x^{i_1\ldots i_m}_{j_1\ldots j_m}$ is a tensor variable. Similarly to types and type symbols, we will use small letters for variable symbols and capital letters for variables; typically, we will write $X = x^{i_j}_{j_j}$.

Natural deduction tensor term expressions and terms are defined from provisional term expressions exactly as in Section 4.1, with the modification that we add to generators (1) of the provisional monoid $\tilde{T}mEx$ all tensor variables, and to defining relations (2) all possible equations

$$\delta^j_i x^j_{i_jj} = x^j_{i_jj}, \quad \delta^j_i x^{i_jj} = x^{i_jj}.$$ 

In the following we abbreviate the title “natural deduction” as “n.d.”.

An n. d. tensor typing judgement is an expression of the form $t : A$, where $t$ is a natural deduction term and $A$ is a tensor type, with $Fsup(t) = Fsub(A)$, $Fsub(t) = Fsup(A)$ (basically, we use a colon instead of a turnstile). A variable declaration is a typing judgement with a tensor variable to the left of the colon, and an n.d. typing context is a finite set of variable declarations which have no common variable symbols. An n.d. tensor sequent is an expression of the form $\Gamma \vdash \sigma$, where $\Gamma$ is an n.d. typing context and $\sigma$ is an n.d. typing judgement. Natural deduction system of ETTC is given by the rules in Figure 4.
Again, it is implicit that types and typing judgements in Figure 4a are well-defined. In particular, in the (\(\Gamma I\)) rule we need pairwise disjoint sequences \(I', J'\) of pairwise distinct fresh indices, in the (\(\Sigma E\)) rule the contexts \(\Gamma, \Theta\) have no common tensor variable symbols etc. The (\(\bigvee I\)) and (\(\bigtriangleup I\)) rules are same as in the sequent calculus and use the same convention: \(\alpha \in Fsup(A), \beta \in Fsub(A)\).

Let us say that a n.d. term is closed if it contains no tensor variables. A closed term is pure if it contains no terminal symbols. A closed term is lexical if, on the contrary, it cannot be written as a normal term expression of the form \(t = \delta_{I'}(a_{(1)})_{I'_{1}} \cdots (a_{(n)})_{I'_{n}}\). A typing judgement \(t : A\) is lexicalized if the term \(t\) is lexical.

It is easily seen that any sequent derivable in natural deduction of ETTC can be written in the form

\[
X_{(1)} : (a_{(1)})_{I'_{1}} \cdots X_{(n)} : (a_{(n)})_{I'_{n}} \vdash X_{(1)} \cdots X_{(n)} \cdot t : B, \tag{3}
\]

where the term \(t\) is pure. (In the case of the (Id) axiom we put \(t = 1\).) The following is completely standard.

**Lemma 5.1 (“Deduction theorem”)** Let \(\Xi\) be a finite set of lexicalized typing judgements, \(\Xi = \{\tau_{(1)} : (a_{(1)})_{I'_{1}}, \ldots, \tau_{(n)} : (a_{(n)})_{I'_{n}}\}\).

A natural deduction sequent of the form \(\vdash \sigma : B\) is derivable from \(\Xi\) using each element of \(\Xi\) exactly once iff there is a natural deduction sequent of the form (3) derivable without nonlogical axioms such that \(t \cdot \tau_{(1)} \cdots \tau_{(n)} = \sigma\). □

We define the *sequent calculus translation* of (3) as the tensor typing judgement

\[
\delta_{I'_{1} \cdots I'_{n}}^{I_{1} \cdots I_{n}} \cdots \delta_{I'_{1} \cdots I'_{n}}^{I_{1} \cdots I_{n}} : t \vdash B, (a_{(1)})_{I'_{1}} \cdots (a_{(n)})_{I'_{n}}, \tag{4}
\]

where \(I_{1}', \ldots, I'_{n}, J_{1}', \ldots, J_{n}\) are pairwise disjoint sequences of pairwise distinct indices, all disjoint from \(Fsup(B), Fsup(B)\). Translation (4) provides for equivalence of natural deduction and sequent calculus formulation of ETTC, which is proven in a routine way familiar from the case of, say, MILL1.

**Lemma 5.2** Sequent (3) is derivable in the natural deduction formulation of ETTC iff its translation (4) is derivable in the sequent calculus formulation. □

### 5.1 Geometric representation and example

Translating to the sequent calculus allows geometric representation of natural deduction derivations as operations on bipartite graphs with indices corresponding to vertices. It is natural to depict indices occurring to the left and to the right of the turnstile as vertices aligned on two parallel lines (continuously deforming the graph of translation (4), where all vertices are on one line). Also, it should be clear from formula (4) that different occurrences of the same index in a natural deduction sequent correspond to distinct indices in the translation and to distinct vertices in the picture. In Figures 4b, 4c, 5b we align vertices vertically. With such a convention, the geometric representation for the (\(\bigvee\)) rules can be shown schematically as in Figures...
Figure 5: Example for natural deduction

4b, 4c. The (\(\triangledown\)) rule does not change the graph, but changes its pictorial representation by a continuous deformation.

Figure 5a shows a derivation of a tensor sequent corresponding under the translation in Figure 3k to a sequent derivable in \(\text{LC}\). Figure 5b gives a geometric representation of the derivation.

5.2 Grammars

We give a more elaborate, linguistically motivated example of a derivation from non-logical axioms in Figure 5. For better readability, we systematically use notation of \(\text{LC}\) and \(\text{MILL}\) understood as in Figure 3k and, in concrete derivations, we omit free indices in types of valency (1, 1) (they are uniquely determined by free indices in terms). In Figure 6a we fix admissible rules corresponding to rules of \(\text{LC}\) and \(\text{MILL}\) that we will use. Figure 6b shows our axioms (the lexicon), which we assume closed under \(\alpha\)-equivalence of typing judgements, and in Figure 6c we derive the noun phrase “Mary who John loves madly”. It might be entertaining to reproduce the derivation in the geometric language.

Now let us define a tensor lexical entry as an \(\alpha\)-equivalence class of lexicalized
(a) Admissible rules

\[ \Gamma, x^J_i : a^J_i \vdash x^K_l : t : B \quad (\rightarrow I) \]
\[ \Gamma \vdash t : \delta_{iJ}^I \cdot a^J_i \rightarrow B \]
\[ \Gamma \vdash t : (b/a)^J_i \quad \Theta \vdash s : a^J_k \]
\[ \Gamma, \Theta \vdash ts : b^I_k \quad (\text{E}) \]

\[ \Gamma \vdash s : \Theta \vdash t : (a \backslash b)^J_i \]
\[ \Gamma, \Theta \vdash st : b^I_k \quad (\text{E}) \]

(b) Axioms

\[ \vdash \text{loves}_{iJ}^k : (n^p \backslash s) \rightarrow n^p \]  
\[ x^J_k : n^p \vdash x^J_k : n^p \]
\[ x^J_k : n^p \vdash \text{loves}_{iJ}^k : x^J_k : n^p \]
\[ x^J_k : n^p \vdash \text{loves}_{iJ}^k : x^J_k : [\text{madly}]_{iJ}^j : n^p \]
\[ \vdash \text{loves}_{iJ}^k : (n^p \backslash s) \rightarrow n^p \]

\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]

\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]

\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]

\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]

\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]
\[ \vdash \text{loves}_{iJ}^k : n^p \]

(c) Derivation

Figure 6: Example with nonlogical axioms
typing judgments and a tensor grammar as a pair \((\text{Lex}, s)\), where \(\text{Lex}\) is a finite set of lexical entries and \(s\) is an atomic type symbol of valency \((1,1)\). For a tensor grammar \(G\) we will right \(\vdash_G \sigma : A\) to indicate that the sequent \(\sigma : A\) is derivable in ETTC from elements of \(\text{Lex}\). We define two languages generated by \(G\). The regular language \(L(G)\) and the regularized language \(L_{reg}(G)\) of \(G\) are, respectively the sets

\[
L(G) = \{ w \vdash_G [w]_i^j : s \}, \quad L_{reg}(G) = \{ w \vdash_G [w]_i^j \cdot (\delta)^n : s \}.
\]

(Note that \(n\) in the definition of \(L_{reg}\) is not an index, but indicates the \(n\)-th power.)

Any balanced MILL1 grammar \(G\) translates to a tensor grammar \(\tilde{G}\) by writing for every lexical entry \((w, A)\) of \(G\) the tensor lexical entry \([w]_{\rho(l)}^\pi(l) : \alpha\), where \(\alpha\) is the type symbol of \(|A|_x^\rho\) (the choice of particular bijections \(\rho, \pi\) is not important because of closure under \(\alpha\)-equivalence of typing judgements).

**Lemma 5.3** In the above setting we have \(L(G) = L_{reg}(\tilde{G})\).

**Proof** Let \(w \in L_{reg}(\tilde{G})\). Then we have \(\vdash_G \tau : s\), where \(\tau = [w]_i^j \cdot (\delta)^n\).

Obviously \(w\) is the concatenation \(w = w_1 \ldots w_n\) of all words occurring in lexical entries \([w]_{\mu} : (a_{\mu})_{\mu}^j\), \(\mu = 1, \ldots, n\), used to derive \(\vdash \tau : s\). By Lemma 5.1 (“Deduction theorem”) it must be that the we have the derivable n.d. sequent

\[
(x(1))_{i_1}^{i_1} \cdot (a_{\mu})_{i_1}^{j_1}, \ldots, (x(n))_{i_1}^{j_n} \vdash \delta_{i_1}^{j_1} \cdot \delta_{i_{n-1}}^{j_{n-1}} \cdot (\delta)^n \cdot s_{i_n}^{j_n},
\]

which corresponds, by Lemma 5.2, to the derivable typing judgement

\[
\delta_{i_0}^{j_0} \cdot \delta_{i_{n+1}}^{j_{n+1}} \cdot (\delta)^n \vdash s_{i_{n+1}}^{j_{n+1}}, \quad (a_{\mu})_{i_1}^{j_1}, \ldots, (a_{\mu})_{i_1}^{j_1}.
\]

If we translate MILL1 to ETTC using the prescription \(\rho(\mu) = i_\mu, \pi(\mu) = i_{\mu+1}\), where \(\mu = 0, \ldots, n\), then, by Lemma 4.1, we have that \(5\) comes from a derivation of the sequent \(\vdash S[0; n], A_n[n; n-1], \ldots, A_1[1; 0]\). The latter is identified as the image of an MILL1 sequent expressing, by definition, that \(w \in L(G)\).

The opposite inclusion is similar, just easier. □

Geometric intuition might suggest that it is the regular, rather than, regularized, language of a tensor grammar that should be accepted as natural. In any case, it can be shown directly that in the case of tensor grammars obtained from Lambek grammars or \(\lambda\)-grammars there are no additional words generated “by regularization”: i.e. \(L(G) = L_{reg}(G)\).

**References**

[1] P. de Groote, “Towards Abstract Categorial Grammars”, in Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, ACL `01, pp.148-155, 2001.

[2] Jean-Yves Girard, “Linear logic”, Theoretical Computer Science, 50:1-102, 1987.
[3] Jean-Yves Girard, “Linear logic: its syntax and semantics”, in J.-Y. Girard, Y. Lafont and L. Regnier, eds. Advances in Linear Logic, 1-42, Cambridge University Press, 1995, Proc. of the Workshop on Linear Logic, Ithaca, New York, June, 1993.

[4] Y. Kubota, R. Levine, “Gapping as like-category coordination”, in D. Béchet & A. Dikovsky, eds, ‘Logical Aspects of Computational Linguistics’, Vol. 7351 of Lecture Notes in Computer Science, Springer, Nantes, pp. 135-150, 2012.

[5] J. Lambek, ”The mathematics of sentence structure”, Amer. Math. Monthly, 65: 154-170, 1958.

[6] R. Moot, “Extended Lambek calculi and first-order linear logic”. In C. Casadio, B. Coecke, M. Moortgat, and P. Scott, eds., Categories and Types in Logic, Language, and Physics: Essays dedicated to Jim Lambek on the Occasion of this 90th Birthday, No. 8222 in Lecture Notes in Artificial Intelligence, 2970330, Springer, 2014.

[7] R. Moot, “Hybrid type-logical grammars, 1rst-order linear logic and the descriptive inadequacy of lambda grammars”, https://hal.archives-ouvertes.fr/hal-00996724, 2014.

[8] R. Moot and M. Piazza, ”Linguistic applications of first order multiplicative linear logic”. Journal of Logic, Language and Information, 10(2):211-232, 2001.

[9] G. Morrill, O. Valentín, M. Fadda, “The displacement calculus”, Journal of Logic, Language and Information 20(1), 1-48, 2011.

[10] R. Muskens, ”Separating syntax and combinatorics in categorial grammar”. Research on Language and Computation 5(3), 267-285, 2007.

[11] V. Mihaliček, C. Pollard, “Distinguishing phenogrammar from tectogrammar simplifies the analysis of interrogatives.” In Proceedings of the 15th and 16th International Conference on Formal Grammar (FG’10/FG’11), 130-145. Springer 2012.

[12] S. Slavnov, “Cobordisms and commutative categorial grammars”, Journal of Cognitive Science, 22, 68-91, 2021.

[13] S. Slavnov, “On embedding Lambek calculus into commutative categorial grammars”, Journal of Logic and Computation, 2021.