Extended Lambek Calculi and First-Order Linear Logic

Richard Moot

CNRS, LaBRI, University of Bordeaux
Richard.Moot@labri.fr

1 Introduction

The Syntactic Calculus [27] — often simply called the Lambek calculus, L, — is a beautiful system in many ways: Lambek grammars give a satisfactory syntactic analysis for the (context-free) core of natural language and, in addition, it provides a simple and elegant syntax-semantics interface.

However, since Lambek grammars generate only context-free languages [49], there are some well-know linguistic phenomena (Dutch verb clusters, at least if we want to get the semantics right [21], Swiss-German verb clusters [54], etc.) which cannot be treated by Lambek grammars.

In addition, though the syntax-semantics interface works for many of the standard examples, the Lambek calculus does not allow a non-peripheral quantifier to take wide scope (as we would need for sentence (1) below if we want the existential quantifier to have wide scope, the so-called "de re" reading) or non-peripheral extraction (as illustrated by sentence (2) below); see [34, Section 2.3] for discussion.

(1) John believes someone left.

(2) John will pick up the package which Mary left here yesterday.

To deal with these problems, several extensions of the Lambek calculus have been proposed. Though this is not the time and place to review them — I recommend [33,34] and the references cited therein for an up-to-date overview of the most prominent extensions; they include multimodal categorial grammar (MMCG,[31]), the Lambek-Grishin calculus (LG,[32]) and the Displacement calculus (D,[46]) — I will begin by listing a number of properties which I consider desirable for such an extension. In essence, these desiderata are all ways of keeping as many of good points of the Lambek calculus as possible while at the same time dealing with the inadequacies sketched above.[1]

[1] To the reader who is justifiably skeptical of any author who writes down a list of desiderata, followed by an argument by this same author arguing how well he scores on his own list, I say only that, in my opinion, this list is uncontroversial and at least implicitly shared by most of the work on extensions of the Lambek calculus and that the list still allows for a considerable debate as to *how well* each extension responds to each desideratum as well as discussion about the relative importance of the different items.

C. Casadio et al. (Eds.): Lambek Festschrift, LNCS 8222, pp. 297–330, 2014.

1. The logic should have a simple proof theory,
2. generate the mildly context-sensitive languages,
3. have a simple syntax-semantics interface giving a correct and simple account of medial scope for quantifiers and of medial extraction,
4. have a reasonable computational complexity.

None of these desiderata is absolute: there are matters of degree for each of them. First of all, it is often hard to distinguish familiarity from simplicity, but I think that having multiple equivalent proof systems for a single calculus is a sign that the calculus is a natural one: the Lambek calculus has a sequent calculus, natural deduction, proof nets, etc. and we would like its extensions to have as many of these as possible, each formulated in the simplest possible way.

The mildly context-sensitive languages [22] are a family of languages which extend the context-free language in a limited way, and opinions vary as to which of the members of this family is the most appropriate for the description of natural language. Throughout this article, I will only make the (rather conservative and uncontroversial) claim that any extension of the Lambek calculus should at least generate the tree adjoining languages, the multiple context-free languages [53] (the well-nested 2-MCFLs [24] are weakly equivalent to the tree adjoining languages) or the simple, positive range concatenation grammars (sRCG, weakly equivalent to MCFG, [7]).

With respect to the semantics, it generally takes the form of a simple homomorphism from proofs in the source logic to proofs in the Lambek-van Benthem calculus LP (which is multiplicative intuitionistic linear logic, MILL, for the linear logicians), though somewhat more elaborate continuation-based mappings [5,35] have been used as well.

Finally, what counts as reasonable computational complexity is open to discussion as well: since theorem-proving for the Lambek calculus is NP complete [50], I will consider NP-complete to be "reasonable", though polynomial parsing is generally considered a requirement for mildly context-sensitive formalisms [22]. Since the complexity of the logic used corresponds to the *universal* recognition problem in formal language theory, NP completeness is not as bad as it may seem, as it corresponds to the complexity of the universal recognition problem for multiple context-free grammars, which is a prototypical mildly context-sensitive formalism (NP completeness holds when we fix the maximum number of string tuples a non-terminal is allowed to have, non-deleting MCFGs without this restriction are PSPACE complete [23]). In addition, many NP hard grammar formalisms such as LFG and HPSG have very efficient parsers [8,30]. Little is known about fragments of extended Lambek calculi with better complexity (though some partial results can be found in [37,39]). Parsing the Lambek calculus itself is known to be polynomial when we fix the order of formulas [51].

Table 1 gives an overview of the Lambek calculus as well as several of its prominent extensions with respect to the complexity of the universal recognition problem, the class of languages generated and the facilities in the formalism for

Table 1. The Lambek calculus and several of its variants/extensions, together with the complexity of the universal recognition problem, classes of languages generated and the appropriateness of the formalism for handling medial quantifier scope and medial extraction

Calculus	Complexity	Languages	Scope	Extraction
L	NP complete	CFL	–	–
MMCG	PSPACE complete	CSL	+	+
LG	NP complete	$\supseteq$ MCFL	+	–
D	NP complete	$\supseteq$ MCFL	+	+
MILL1	NP complete	$\supseteq$ MCFL	+	+

handling medial quantifier scope and medial extraction. Note that few exact upper bounds for language classes are known.

In this paper, I will present an alternative extension of the Lambek calculus: first-order multiplicative intuitionistic linear logic (MILL1) [15,40]. It generates the right class of languages (MCFG are a subset of the Horn clause fragment, as shown in Section 3.3), and embeds the Displacement calculus (D, as shown in Section 4 and 5). As can be seen in Table 1, it has the lowest complexity class among the different extensions, generates (at least) the right class of languages, but also handles medial scope and medial extraction in a very simple way (as shown already in [40]). In addition, as we will see in Section 2, MILL1 has a very simple proof theory, essentially a resource-conscious version of first-order logic, with a proof net calculus which is a simple extension of the proof nets of multiplicative linear logic [10,15]. Finally, the homomorphism from MILL1 to MILL for semantics consists simply of dropping the first-order quantifiers.

I will also look at the (deterministic, unit-free) Displacement calculus from the perspective of MILL1 and give a translation of D into MILL1, indirectly solving two open problems from [43] by providing a proof net calculus for D and showing that D is NP complete. In addition it is also worth mentioning briefly that the simpler proof theory of MILL1 (ie. proof nets) greatly simplifies the cut elimination proofs of D [46,57]: as for the multiplicative case, cut elimination for MILL1 consists of simple, local conversions with only three distinct cases to verify (axiom, tensor/par and existential/universal).

The remainder of this paper is structured as follows. In the next section, I will briefly introduce MILL1 and its proof theory, including a novel correctness condition for first-order proof nets, which is a simple extension of the contraction criterion from Danos [9]. Section 3 will introduce the Displacement calculus, D, using a presentation of the calculus from [46] which emphasizes the operations on string tuples and, equivalently, on string positions. Section 4 will present a translation from D to MILL1, with a correctness proof in Section 5. Section 6 will briefly mention some other possible applications of MILL1, which include agreement, non-associativity and island constraints, and quantifier scope. Finally, I will reflect on the implications of the results in this paper and give some interesting open problems.

2 MILL1

First-order multiplicative intuitionistic linear logic (MILL1) extends (multiplicative) intuitionistic linear logic with the first-order quantifiers $\exists$ and $\forall$. The first-order multiplicative fragment shares many of the good properties of the propositional fragment: the decision problem is NP complete [28] and it has a simple proof net calculus which is an extension of the proof net calculus for multiplicative linear logic.

Table 2 presents the natural deduction calculus for MILL1, which is without surprises, though readers familiar with intuitionistic logic should note that the $\otimes E$, $\multimap I$ and $\exists E$ rule discharge exactly one occurrence of each of the hypotheses with which it is coindexed.

Table 2. Natural deduction rules for MILL1

$$\frac{A \otimes B \qquad \begin{array}{c}[A]^i[B]^i\\ \vdots\\ C\end{array}}{C}\otimes E_i \qquad \frac{A \quad B}{A \otimes B}\otimes I$$

$$\frac{A \quad A \multimap B}{B} \multimap E \qquad \frac{\begin{array}{c}[A]^i\\ \vdots\\ B\end{array}}{A \multimap B} \multimap I$$

$$\frac{\exists x.A \qquad \begin{array}{c}[A]^i\\ \vdots\\ C\end{array}}{C}\exists E_i^* \qquad \frac{A[x := t]}{\exists x.A}\exists I$$

$$\frac{\forall x.A}{A[x := t]}\forall E \qquad \frac{A}{\forall x.A}\forall I^*$$

* no free occurrences of x in any of the free hypotheses

The presentation of proof nets is (out of necessity) somewhat terse. A more gentle introduction to proof nets can be found, for example in [16,42]. I will present the proof net calculus in three steps, which also form a basic proof search procedure: for a given statement $\Gamma \vdash C$ (with C a formula and Γ a multiset of formulas) we form a *proof frame* by unfolding the formulas according to the logical links shown in the bottom two rows of Table 3, using the negative unfolding for the formulas in Γ and the positive unfolding for the formula C. We then connect the atomic formulas using the axiom link (shown on the top left of the table) until we have found a complete matching of the atomic formulas, forming a *proof structure*. Finally, we check if the resulting proof structure is a

proof net (ie. we verify if $\Gamma \vdash C$ is derivable) by verifying it satisfies a correctness condition.

As is usual, I will use the following conventions, which will make formulating the proof net calculus simpler.

- dotted binary links are called *par* links, solid binary links are called *tensor* links,
- dotted unary links are called *universal* links, solid unary links are called *existential* links, the bound variables of these links are called universally bound and existentially bound respectively.
- each occurrence of a quantifier link uses a distinct bound variable,
- the variable of a positive $\forall$ and a negative $\exists$ link (ie. the universal links and universally quantified variables) are called its *eigenvariable*,
- following [4], I require eigenvariables of existential links to be used *strictly*, meaning that replacing the eigenvariable throughout a proof with a special, unused constant will *not* result in a proof (in other words, we never unnecessarily instantiate an existentially quantified variable with the eigenvariable of a universal link).

The fact that both par links and universal links are drawn with dotted lines is not a notational accident: one of the fundamental insights of focusing proofs and ludics [2,17] is that these two types of links naturally group together, as do the existential and tensor links, both drawn with solid lines. This property is also what makes the correctness proof of Section 5 work. When it is convenient to refer to the par and universal links together, I will call them *asynchronous* links, similarly I will refer to the existential and tensor links as *synchronous* links (following Andreoli [2]).

In Table 3, the formulas drawn below the link are its conclusions (the axiom link, on the top left of the table, is the only multiple conclusion link, the cut link, on the top right, does not have a conclusion, all logical links have a single conclusion), the formulas drawn above the link are its premisses.

Definition 1. *A* proof structure *is a set of polarized formulas connected by instances of the links shown in Table 3 such that each formula is at most once the premiss of a link and exactly once the conclusion of a link. Formulas which are not the premiss of any link are called the* conclusions *of the proof structure. We say a proof structure with negative conclusions Γ and positive conclusions Δ is a proof structure of the statement $\Gamma \vdash \Delta$.*

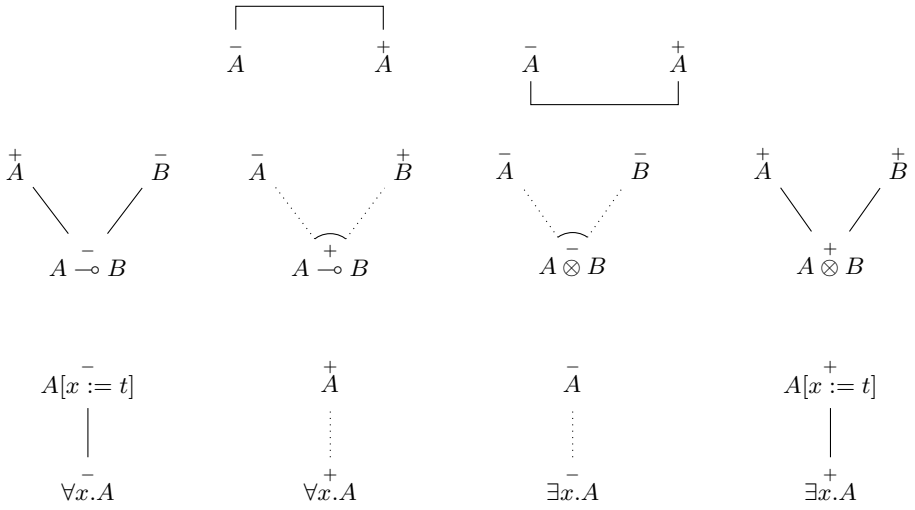
Definition 2. *Given a proof structure Π a* switching *is*

- *for each of the par links a choice of one of its two premisses,*
- *for each of the universal links a choice either of a formula containing the eigenvariable of the link or of the premiss of the link.*

Definition 3. *Given a proof structure Π and a switching s we obtain a* correction graph *G by*

- *replacing each par link by an edge connecting the conclusion of the link to the premiss selected by s*
- *replacing each universal link by an edge connecting the conclusion of the link to the formula selected by s*

Table 3. Logical links for MILL1 proof structures



Whereas a proof structure is a graph with some additional structure (paired edges, draw as connected dotted lines for the par links, and "universal" edges, draw as dotted lines) a correction graph is a plain graph as used in graph theory: both types of special edges are replaced by normal edges according to the switching s.

Definition 4. *A proof structure is a* proof net *iff for all switchings s the corresponding correction graph G is acyclic and connected.*

Remarkably, the proof nets correspond exactly to the provable statements in MILL1 [15].

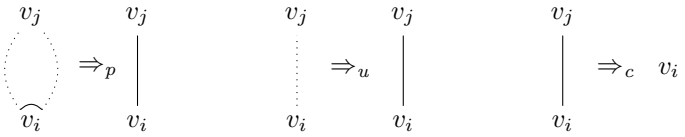
The basic idea of [40] is very simple: instead of using the well-known translation of Lambek calculus formulas into first-order logic (used for model-theory, see e.g. [11]), we use this same translation to obtain formulas of first-order multiplicative *linear* logic. In this paper, I extend this result to the discontinuous Lambek calculus D, while at the same time sketching some novel applications of the system which correspond more closely to analyses in multimodal categorial grammars.

2.1 A Danos-Style Correctness Condition

Though the correctness condition is conceptually simple, a proof structure has a number of correction graphs which is exponential in the number of asynchronous links, making the correctness condition hard to verify directly (though linear-time algorithms for checking the correctness condition exist in the quantifier-free case, eg. [20,47]).

Here, I present an extension of the correctness condition of [9] to the first-order case, which avoids this exponential complexity. Let G be a proof

structure, where each vertex of the proof structure is a assigned the set of eigenvariables which occur in the corresponding formula. Then we have the following contractions.



There is one contraction for the par links (p), one contraction for the universal links (u) and a final contraction which contracts components (connected subgraphs consisting only of synchronous, axiom and cut links) to a single vertex (c). The u contraction has the condition that there are no occurrences of the eigenvariable of the universal variable corresponding to the link outside of v_j. The c contraction has as condition that $i \neq j$; it contracts the vertex connecting i and j and the set of eigenvariables of v_i on the right hand side of the contraction corresponds to the set union of the eigenvariables of v_i and v_j on the left hand side of the contraction.

The following proposition is easy to prove using induction on the number of asynchronous links in the proof structure, using a variant of the "splitting par" sequentialization proof of Danos [9]:

Proposition 1. *A proof structure is a proof net iff it contracts to a single vertex using the contractions p, u and c.*

It is also easy to verify that the contractions are confluent, and can therefore be applied in any desired order.

To give an idea of how these contractions are applied, Figure 1 shows (on the left) a proof structure for the underivable statement $\forall x \exists y.f(x,y) \vdash \exists v \forall w.f(w,v)$. In the middle of the figure, we see the proof structure with each formula replaced by the set of its free variables and before any contractions, with the eigenvariables shown next to their universal links. On the right, we see the structure after all c contractions have been applied. It is clear that we cannot apply the u contraction for y, since y occurs at a vertex other than the top vertex. Similarly, we cannot apply the u contraction for w either, meaning the proof structure is not contractible and therefore not a proof net.

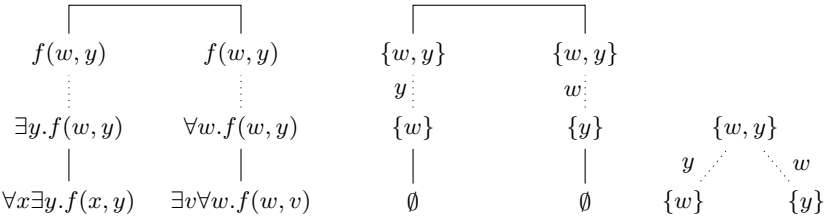


Fig. 1. Proof structure and partial contraction sequence for the underivable statement $\forall x \exists y.f(x,y) \vdash \exists v \forall w.f(w,v)$

Figure 2 shows the proof structure and part of the contraction sequence for the derivable statement $\exists x \forall y.f(x,y) \vdash \forall v \exists w.f(w,v)$. In this case, the structure on the right *does* allow us to perform the u contractions (in any order), producing a single vertex and thereby showing the proof structure is a proof net.

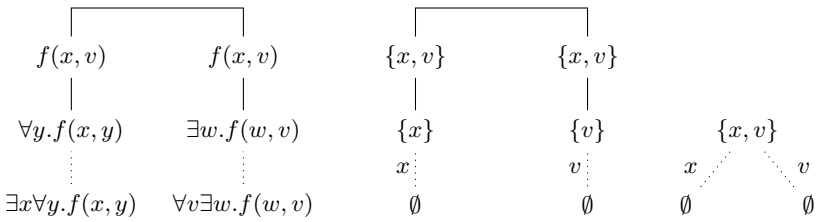


Fig. 2. Proof net and partial contraction sequence for the derivable statement $\exists x \forall y.f(x,y) \vdash \forall v \exists w.f(w,v)$

2.2 Eager Application of the Contractions

Though the contraction condition can be efficiently implemented, when verifying whether or not a given statement is a proof net it is often possible to disqualify partial proof structures (that is, proof structures where only some of the axiom links have been performed). Since the number of potential axiom links is enormous ($n!$ in the worst case), efficient methods for limiting the combinatorial explosion as much as possible are a prerequisite for performing proof search on realistic examples.

The contractions allow us to give a compact representation of the search space by reducing the partial proof structure produced so far. When each vertex is assigned a multiset of literals (in addition to the set of eigenvariables already required for the contractions), the axiom rule corresponds to selecting, if necessary, while unifying the existentially quantified variables, two conjugate literals $+A$ and $-A$ from two different vertices (since the axiom rule corresponds to an application of the c contraction), identifying the two vertices and taking the multiset union of the remaining literals from the two vertices, in addition to taking the set union of the eigenvariables of the vertices. When the input consists of (curried) Horn clauses, each vertex will correspond to a Horn clause; therefore this partial proof structure approach generalizes resolution theorem proving. However, it allows for a lot of freedom in the strategy of literal selection, so we can apply "smart backtracking" strategies such as selecting the literal which has the smallest number of conjugates [36,38]. The contraction condition immediately suggest the following.

– never connect a literal to a descendant or an ancestor (generalizes "formulas from different vertices" for the Horn clause case); failure to respect this constraint will result in a cyclic proof structure,

- if the premiss of an asynchronous link is a leaf with the empty set of literals, then we must be able to contract it immediately ; failure to respect this constraint will result in a disconnected proof structure.
- similarly, if an isolated vertex which is not the only vertex in the graph has the empty set of literals, then the proof structure is disconnected.

3 The Displacement Calculus

The Displacement calculus [46] is an extension of the Lambek calculus using tuples of strings as their basic units. Unless otherwise noted, I will restrict myself Displacement calculus without the identity elements, the synthetic connectives (though see the discussion in Section 4.2 on how some of the synthetic connectives can be included) or the non-deterministic connectives.

3.1 String Tuples

Whereas the Lambek calculus is the logic of strings, several formalisms are using *tuples* of strings as their basic units (eg. MCFGs, RCGs).

In what follows I use s, $s_0, s_1, \ldots, s', s'', \ldots$ to refer to *simple* strings (ie. the 1-tuples) with the constant ϵ for the empty string. The letters t, u, v etc. refer to i-tuples of strings for $i \geq 1$. I will write a i-tuple of strings as $s_1, \ldots, s_i$, but also (if $i \geq 2$) as s_1, t or t', s_i where t is understood to be the string tuple $s_2, \ldots, s_i$ and t' the string tuple $s_1, \ldots, s_{i-1}$, both $(i-1)$-tuples.

The basic operation for simple strings is concatenation. How does this operation extend to string tuples? For our current purposes, the natural extension of concatenation to string tuples is the following

$$(s_1, \ldots, s_m) \circ (s'_1, \ldots, s'_n) = s_1, \ldots, s_m s'_1, \ldots, s'_n$$

where $s_m s'_1$ is the string concatenation of the two simple strings s_m and s'_1. In other words, the result of concatenating an m-tuple t and an n-tuple u is the $n+m-1$ tuple obtained by first taking the first $m-1$ elements of t, then the simple string concatenation of the last element of t with the first element of u and finally the last $n-1$ elements of u. When both t and u are simple strings, then their concatenation is the string concatenation of their single element.[2] In what follows, I will simply write tu for the concatenation of two string tuples t and u and $u[t]$ to abbreviate $u_1 t u_2$.

3.2 Position Pairs

As is common in computational linguistics, it is sometimes more convenient to represent a simple string as a pair of string positions, the first element of the pair representing the leftmost string position and the second element its rightmost

[2] Another natural way to define concatenation is as point-wise concatenation of the different elements of two (equal-sized) tuples, as done by Stabler [56].

position. These positions are commonly represented as integers (to make the implicit linear precedence relation more easily visible). Likewise, we can represent an n-tuple of strings as a $2n$ tuple of string positions. This representation has the advantage that it makes string concatenation trivial: if x_0, x_1 is a string starting at position x_0 and ending at position x_1 and x_1, x_2 is a string starting at position x_1 and ending at position x_2 then the concatenation of these two strings is simply x_0, x_2 (this is the familiar difference list concatenation from Prolog [52]).

Definition 5. *We say a grammar is* simple in the input string *if for each input string $w_1, \ldots, w_n$ we have that w_i spans positions $i, i+1$.*

Much of the work in parsing presupposes grammars are simple in the input string [48], since it makes the definition of the standard parsing algorithms much neater. However, the original construction of Bar-Hillel et al. [3] on which it is based is much more general: it computes the intersection of a context-free grammar and a finite-state automaton (FSA), where each non-terminal is assigned an input state and an output state of the FSA. For grammars which are not simple in the input string, this FSA can have self-loops and complex cycles, whereas the input string for a simple grammar is an FSA with a simple, deterministic linear path as shown in the example below. With the exception of Section 4.2, where I discusses the possibility of abandoning this constraint, the grammars I use will be simple in the input string.

$$1 \xrightarrow{\text{Jan}} 2 \xrightarrow{\text{Henk}} 3 \xrightarrow{\text{Cecilia}} 4 \xrightarrow{\text{de}} 5 \xrightarrow{\text{nijlpaarden}} 6 \xrightarrow{\text{zag}} 7 \xrightarrow{\text{helpen}} 8 \xrightarrow{\text{voeren}} 9$$

Suppose "nijlpaarden" (hippos) above is assigned the category n, for noun. Incorporating its string positions produces $n(5,6)$. It gets more interesting with the determiner "de" (the): we assign it the formula $\forall x.n(5,x) \multimap np(4,x)$, which says that whenever it finds an n to its immediate right (starting at position 5 and ending at any position x) it will return an np from position 4 to this same x (this is the MILL1 translation of np/n at position $4,5$). In a chart parser, we would indicate this by adding an np arc from 4 to 6. There is an important difference with a standard chart parser though: since we are operating in a resource-conscious logic, we know that in a correct proof each rule is used exactly once (though their order is only partially determined).

Figure 3 shows the three elementary string operations of the Displacement calculus both in the form of operations of string tuples and in the form of operations of string position pairs.

Concatenation takes an i-tuple t (shown in the top of the figure as the white blocks, with corresponding string positions $x_0, \ldots, x_n$ for $n = 2i - 1$) and a j-tuple u (shown in the top of the figure as the gray blocks, with corresponding string positions $x_n, \ldots, x_{n+m}$ for $m = 2j - 1$) and the resulting concatenation tu (with the last element of t concatenated to the first element of u, indicated as the gray-white block x_{n-1}, x_{n+1}; x_n is not a string position in the resulting $i + j - 1$-tuple tu, which consists of the string positions $x_0, \ldots, x_{n-1}, x_{n+1}, \ldots, x_{n+m}$.

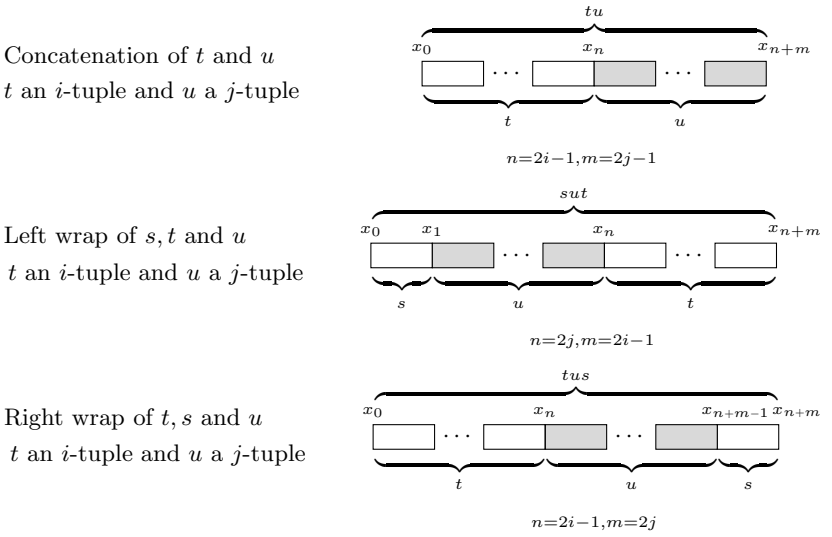


Fig. 3. String operations and their equivalent string positions operations

Left wrap takes an $i+1$-tuple s,t (with s a simple string and t an i-tuple, with string positions $x_0, x_1, x_n, \ldots, x_{n+m}$) and a j-tuple u (with string positions $x_1, \ldots, x_n$) and wraps s,t around u producing and $i+j-1$-tuple sut with string positions $x_0, x_2, \ldots, x_{n-1}, x_{n+1}, \ldots, x_{n+m}$, with positions x_1 and x_n removed because of the two concatenations.

Symmetrically, *right wrap* takes an $i+1$-tuple t,s (with s a simple string and t an i-tuple) and a j-tuple u and wraps t,s around u producing tus.

Given these operations, the proof rules for D are simple to state. I give a notational variant of the natural deduction calculus of [46]. As usual, natural deduction proofs start with a hypothesis $t : A$ (for $t : A$ an entry the lexicon of the grammar, in which case t is a lexical constant, or for a hypothesis discharged by the product elimination and implication introduction rules, in which case t is an appropriate n-tuple). In each case the string tuple t is unique in the proof.

For a given (sub-)proof, the *active* hypotheses are all hypotheses which have not been discharged by a product elimination of implication introduction rule in this (sub-)proof.

For the logical rules, we can see that the different families of connectives correspond to the three basic string tuple operations: with concatenation for $/$, $\bullet$ and $\backslash$ (the rules are shown in Figure 4 and Figure 7 with the corresponding string tuples), left wrap for $\uparrow_>$, $\odot_>$ and $\downarrow_>$ (shown in Figure 5 and Figure 8) and right wrap for $\uparrow_<$, $\odot_<$ and $\downarrow_<$ (shown in Figure 6 and Figure 9).

In the discontinuous Lambek calculus, we define the *sort* of a formula F, written $s(F)$ as the number of items in its string tuple minus 1. Given sorts for the atomic formulas, we compute the sort of a complex formula as shown

$$\frac{t : A \quad u : A \setminus C}{tu : C}\ \setminus E \qquad \frac{\begin{array}{c}[t : A]^i \\ \vdots \\ tu : C\end{array}}{u : A \setminus C}\ \setminus I_i$$

$$\frac{t : C \,/\, B \quad u : B}{tu : C}\ /E \qquad \frac{\begin{array}{c}[u : B]^i \\ \vdots \\ tu : C\end{array}}{t : C \,/\, B}\ /I_i$$

$$\frac{t : A \bullet B \qquad \begin{array}{c}[t_1 : A]^i \quad [t_2 : B]^i \\ \vdots \\ u[t_1 t_2] : C\end{array}}{u[t] : C}\ \bullet E_i \qquad \frac{t : A \quad u : B}{tu : A \bullet B}\ \bullet I$$

Fig. 4. Proof rules – Lambek calculus

$$\frac{s,t : A \quad u : A \downarrow_> C}{sut : C}\ \downarrow_> E \qquad \frac{\begin{array}{c}[s,t : A]^i \\ \vdots \\ sut : C\end{array}}{u : A \downarrow_> C}\ \downarrow_> I_i$$

$$\frac{s,t : C \uparrow_> B \quad u : B}{sut : C}\ \uparrow_> E \qquad \frac{\begin{array}{c}[u : B]^i \\ \vdots \\ sut : C\end{array}}{s,t : C \uparrow_> B}\ \uparrow_> I_i$$

$$\frac{t : A \odot_> B \qquad \begin{array}{c}[s, t_2 : A]^i \quad [t_1 : B]^i \\ \vdots \\ u[st_1 t_2] : C\end{array}}{u[t] : C}\ \odot_> E_i \qquad \frac{s,t : A \quad u : B}{sut : A \odot_> B}\ \odot_> I$$

Fig. 5. Proof rules — leftmost infixation,extraction

in Table 4 (the distinction between the left wrap and right wrap connectives is irrelevant for the sorts).

3.3 MILL1 and Multiple Context-Free Grammars

It is fairly easy to see that MILL1 generates (at least) the multiple context-free languages (or equivalently, the languages generated by simple, positive range concatenation grammars [7]) by using a lexicalized form of the grammars as defined below.

Definition 6. *A grammar is* lexicalized *if each grammar rule uses exactly one non-terminal symbol.*

Table 4. Computing the sort of a complex formula given the sort of its immediate subformulas

$$
\begin{aligned}
s(A \bullet B) &= s(A) + s(B) \\
s(A \setminus C) &= s(C) - s(A) \\
s(C / B) &= s(C) - s(B)
\end{aligned}
$$

$$
\begin{aligned}
s(A \odot B) &= s(A) + s(B) - 1 \\
s(A \downarrow C) &= s(C) + 1 - s(A) \\
s(C \uparrow B) &= s(C) + 1 - s(B)
\end{aligned}
$$

$$\frac{t, s : A \quad u : A \downarrow_< C}{tus : C} \downarrow_< E \qquad \frac{\begin{array}{c}[t, s : A]^i \\ \vdots \\ tus : C\end{array}}{u : A \downarrow_< C} \downarrow_< I_i$$

$$\frac{t, s : C \uparrow_< B \quad u : B}{tus : C} \uparrow_< E \qquad \frac{\begin{array}{c}[u : B]^i \\ \vdots \\ tus : C\end{array}}{t, s : C \uparrow_< B} \uparrow_< I_i$$

$$\frac{t : A \odot_< B \qquad \begin{array}{c}[t_1, s : A]^i \quad [t_2 : B]^i \\ \vdots \\ u[t_1 t_2 s] : C\end{array}}{u[t] : C} \odot_< E_i \qquad \frac{t, s : A \quad u : B}{tus : A \odot_< B} \odot_< I$$

Fig. 6. Proof rules — rightmost infixation,extraction

Lexicalization is one of the principal differences between traditional phrase structure grammars and categorial grammars: categorial grammars generally require a form of lexicalization, whereas phrase structure grammars do not. The most well-known lexicalized form is the Greibach normal form for context-free grammars [19]. Wijnholds [58] shows that any (ϵ-free) simple, positive range concatenation grammar has a lexicalized grammar generating the same language (see also [55])[3]. Since ranges are simply pairs of non-negative integers (see Section 3.2 and [7]) these translate directly to Horn clauses in MILL1. The following rules are therefore both a notational variant of a (lexicalized) MCFG/sRCG and an MILL1 lexicon (corresponding to the verbs of the example on page 306).

[3] Wijnholds [58] and Sorokin [55] also given translations between MCFG/sRCG and D, to which we will return in Section 4.2.

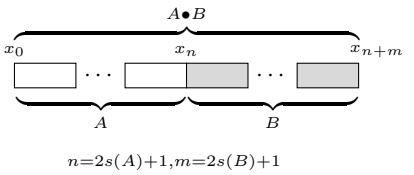
$$\forall x_0x_1x_2x_3.np(x_0,x_1)\otimes np(x_1,x_2)\otimes \mathit{inf}(x_2,6,7,x_3)\multimap s(x_0,x_3) \quad \textit{zag}$$
$$\forall x_0x_1x_2x_3.np(x_0,x_1)\otimes \mathit{inf}(x_1,x_2,8,x_3)\multimap \mathit{inf}(x_0,x_2,7,x_3) \quad \textit{helpen}$$
$$\forall x_0x_1.np(x_0,x_1)\multimap \mathit{inf}(x_0,x_1,8,9) \quad \textit{voeren}$$

In Section 4.2, we will see how we can obtain (the Curried versions of) these formulas via a translation of D and provide the corresponding MILL1 proof net.

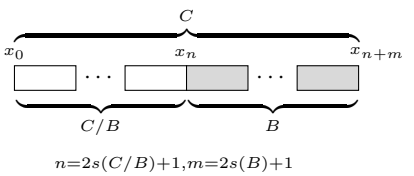
4 Translations

The proof rules and the corresponding string tuple operations (shown in Figures 7, 8 and 9) suggest the translation shown in Table 5 of D formulas into MILL1 formulas. It is an extension of the translation of Lambek calculus formulas of [40], while at the same time extending the translation of [44,12] for the simple displacement calculus (1-D, where all formulas are of sort ≤ 1). Fadda [12] also gives a proof net calculus for the simple displacement calculus, which is a special case of the one presented here.

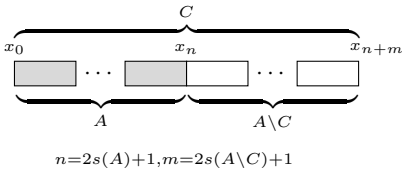
The reader intimidated by the number variable indices in Table 5 is invited to look at Figures 7, 8 and 9 for the correspondence between the string position numbers and the string components of the different formulas in the translation. Section 4.1 will illustrate the translation using some examples, whereas Section 5



$$\frac{t:A\bullet B \qquad \begin{array}{c}[t_1:A]^i \quad [t_2:B]^i\\ \vdots\\ u[t_1t_2]:C\end{array}}{u[t]:C}\ \bullet E_i \qquad \frac{t:A \quad u:B}{tu:A\bullet B}\ \bullet I$$

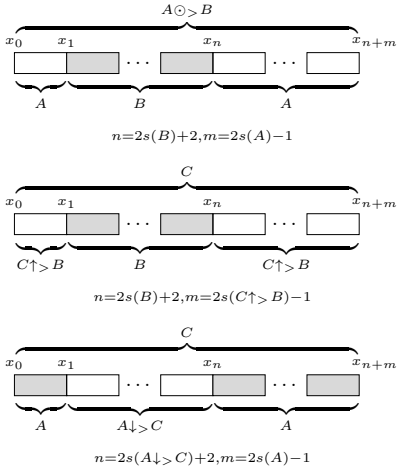


$$\frac{t:C\,/\,B \quad u:B}{tu:C}\ /E \qquad \frac{\begin{array}{c}[u:B]^i\\ \vdots\\ tu:C\end{array}}{t:C\,/\,B}\ /I_i$$



$$\frac{t:A \quad u:A\setminus C}{tu:C}\ \setminus E \qquad \frac{\begin{array}{c}[t:A]^i\\ \vdots\\ tu:C\end{array}}{u:A\setminus C}\ \setminus I_i$$

Fig. 7. String positions – Lambek calculus

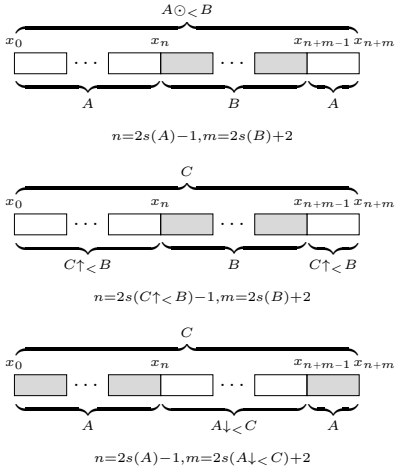


$$\frac{t : A \odot_> B \qquad \begin{array}{c}[s, t_2 : A]^i \quad [t_1 : B]^i \\ \vdots \\ u[st_1t_2] : C\end{array}}{u[t] : C} \odot_> E_i \qquad \frac{s, t : A \quad u : B}{sut : A \odot_> B} \odot_> I$$

$$\frac{s, t : C \uparrow_> B \quad u : B}{sut : C} \uparrow_> E \qquad \frac{\begin{array}{c}[u : B]^i \\ \vdots \\ sut : C\end{array}}{s, t : C \uparrow_> B} \uparrow_> I_i$$

$$\frac{s, t : A \quad u : A \downarrow_> C}{sut : C} \downarrow_> E \qquad \frac{\begin{array}{c}[s, t : A]^i \\ \vdots \\ sut : C\end{array}}{u : A \downarrow_> C} \downarrow_> I_i$$

Fig. 8. String positions – leftmost infix/extraction



$$\frac{t : A \odot_< B \qquad \begin{array}{c}[t_1, s : A]^i \quad [t_2 : B]^i \\ \vdots \\ u[t_1t_2s] : C\end{array}}{u[t] : C} \odot_< E_i \qquad \frac{t, s : A \quad u : B}{tus : A \odot_< B} \odot_< I$$

$$\frac{t, s : C \uparrow_< B \quad u : B}{tus : C} \uparrow_< E \qquad \frac{\begin{array}{c}[u : B]^i \\ \vdots \\ tus : C\end{array}}{t, s : C \uparrow_< B} \uparrow_< I_i$$

$$\frac{t, s : A \quad u : A \downarrow_< C}{tus : C} \downarrow_< E \qquad \frac{\begin{array}{c}[t, s : A]^i \\ \vdots \\ tus : C\end{array}}{u : A \downarrow_< C} \downarrow_< I_i$$

Fig. 9. String positions – rightmost infix/extraction

will make the correspondence with the rules from the Displacement calculus more precise.

Note: the sequence $x_i, \ldots, x_i$ is of course simply the unit sequence x_i whereas the sequence $x_i, \ldots, x_{i-1}$ is the empty sequence.

If there are at most two string tuples, both $C \uparrow_> B$ (Equation 5 with $n = 2$, $m = 1$, remembering that $x_2, \ldots, x_{n-1} \equiv x_2, \ldots, x_1$ which is equivalent to the empty sequence of string positions and the empty sequence of quantifier prefixes,

Table 5. Translation of D formulas to MILL1 formulas

$$(1)\quad \left.\begin{array}{r}\|A \bullet B\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}} = \\ \exists x_n \|A\|^{x_0,\ldots,x_n} \otimes \|B\|^{x_n,\ldots,x_{n+m}}\end{array}\right\}_{n=2s(A)+1,m=2s(B)+1}$$

$$(2)\quad \left.\begin{array}{r}\|C \,/\, B\|^{x_0,\ldots,x_n} = \\ \forall x_{n+1},\ldots,x_{n+m}\|B\|^{x_n,\ldots,x_{n+m}} \multimap \\ \|C\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}\end{array}\right\}_{n=2s(C/B)+1,m=2s(B)+1}$$

$$(3)\quad \left.\begin{array}{r}\|A \setminus C\|^{x_n,\ldots,x_{n+m}} = \\ \forall x_0,\ldots,x_{n-1}\|A\|^{x_0,\ldots,x_n} \multimap \\ \|C\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}\end{array}\right\}_{n=2s(A)+1,m=2s(A\setminus C)+1}$$

$$(4)\quad \left.\begin{array}{r}\|A \odot_> B\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}} = \\ \exists x_1,x_n\|A\|^{x_0,x_1,x_n,\ldots,x_{n+m}} \otimes \|B\|^{x_1,\ldots,x_n}\end{array}\right\}_{n=2s(B)+2,m=2s(A)-1}$$

$$(5)\quad \left.\begin{array}{r}\|C \uparrow_> B\|^{x_0,x_1,x_n,\ldots,x_{n+m}} = \\ \forall x_2,\ldots,x_{n-1}\|B\|^{x_1,\ldots,x_n} \multimap \\ \|C\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}\end{array}\right\}_{n=2s(B)+2,m=2s(C\uparrow_> B)-1}$$

$$(6)\quad \left.\begin{array}{r}\|A \downarrow_> C\|^{x_1,\ldots,x_n} = \\ \forall x_0,x_{n+1},\ldots,x_{n+m}\|A\|^{x_0,x_1,x_n,\ldots,x_{n+m}} \multimap \\ \|C\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}\end{array}\right\}_{n=2s(A\downarrow_> C)+2,m=2s(A)-1}$$

(7)

$$\left.\begin{array}{r}\|A \odot_< B\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m-2},x_{n+m}} = \\ \exists x_n,x_{n+m-1}\|A\|^{x_0,\ldots,x_n,x_{n+m-1},x_{n+m}} \otimes \|B\|^{x_n,\ldots,x_{n+m-1}}\end{array}\right\}_{n=2s(A)-1,m=2s(B)+2}$$

$$(8)\quad \left.\begin{array}{r}\|C \uparrow_< B\|^{x_0,\ldots,x_n,x_{n+m-1},x_{n+m}} = \\ \forall x_{n+1},\ldots,x_{n+m-2}\|B\|^{x_n,\ldots,x_{n+m-1}} \multimap \\ \|C\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m-2},x_{n+m}}\end{array}\right\}_{n=2s(C\uparrow_> B)-1,m=2s(B)+2}$$

$$(9)\quad \left.\begin{array}{r}\|A \downarrow_< C\|^{x_n,\ldots,x_{n+m-1}} = \\ \forall x_0,\ldots,x_{n-1},x_{n+m}\|A\|^{x_0,\ldots,x_n,x_{n+m-1},x_{n+m}} \multimap \\ \|C\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m-2},x_{n+m}}\end{array}\right\}_{n=2s(A)-1,m=2s(A\downarrow_> C)+2}$$

and that $x_{n+1}, \ldots, x_{n+m} \equiv x_3, \ldots, x_3 \equiv x_3$) and $C \uparrow_< B$ (Equation 8 with $n = 1$, $m = 2$) translate to the following

$$\|C \uparrow B\|^{x_0,x_1,x_2,x_3} = \|B\|^{x_1,x_2} \multimap \|C\|^{x_0,x_3}$$

Similarly, it is easy to verify that both $A \downarrow_> C$ (Equation 6 with $n = 2$, $m = 1$, remember that $x_2, \ldots x_{n-1} \equiv x_2, \ldots, x_1$ and therefore equal to the empty sequence and that $x_{n+1}, \ldots, x_{n+m} \equiv x_3, \ldots, x_3 \equiv x_3$) and $A \downarrow_< C$ (Equation 9 with $n = 1$, $m = 2$) produce the following translation for D formulas with at most two string tuples.

$$\|A \downarrow C\|^{x_1,x_2} = \forall x_0, x_3 \|A\|^{x_0,x_1,x_2,x_3} \multimap \|C\|^{x_0,x_3}$$

In the Lambek calculus, all sorts are zero, therefore instantiating Equation 3 with n=1, m=1 produces the following

$$\|A \setminus C\|^{x_1,x_2} = \forall x_0 \|A\|^{x_0,x_1} \multimap \|C\|^{x_0,x_2}$$

and therefore has the translation of [40] as a special case.

4.1 Examples

As an illustration, let's look at the formula unfolding of $((vp \uparrow vp)/vp)\backslash(vp \uparrow vp)$, which is the formula for "did" assigned to sentences like

(3) John slept before Mary did.

by [46]. This lexical entry for "did" is of sort 0 and therefore has two string positions (I use 4 and 5) to start off its translation. However, since both direct subformulas are of sort 1, these subformulas have four position variables each. Applying the translation for $\backslash$ shown in Equation 3 with $n = 3$ $(= 2s((vp \uparrow vp)/vp) + 1)$, $m = 1$ (the sort of the complete formula being 0) gives us the following partial translation.

$$\forall x_0 x_1 x_2 \|(vp \uparrow vp)/vp\|^{x_0,x_1,x_2,4} \multimap \|vp \uparrow vp\|^{x_0,x_1,x_2,5}$$

I first translate the leftmost subformula, which is of sort 1, and apply the / rule (Equation 2) with $n = 3$ $(= 2s((vp \uparrow vp)/vp) + 1)$ and $m = 1$ $(= 2s(vp) + 1)$ giving the following partial translation.

$$\forall x_0 x_1 x_2 [\forall x_3 [\|vp\|^{4,x_3} \multimap \|vp \uparrow vp\|^{x_0,x_1,x_2,x_3}] \multimap \|vp \uparrow vp\|^{x_0,x_1,x_2,5}]$$

Applying the translation rule for $C \uparrow B$ (Equation 5) twice produces.

$$\forall x_0 x_1 x_2 [\forall x_3 [\|vp\|^{4,x_3} \multimap \|vp\|^{x_1,x_2} \multimap \|vp\|^{x_0,x_3}] \multimap \|vp\|^{x_1,x_2} \multimap \|vp\|^{x_0,5}]$$

Figure 10 on the facing page shows a proof net for the complete sentence — slightly abbreviated, in that not all vp's have been expanded and that the existential links and the corresponding substitutions have not been included in the figure.

The intelligent backtracking solution of Section 2.2 (and [38]) guarantees that at each step of the computation we can make a deterministic choice for literal selection, though the reader is invited to try and find a proof by hand to convince himself that this is by no means a trivial example!

As a slightly more complicated example translation, which depends on the distinction between left wrap and right wrap, Morrill et al. [46] give the following formula for an object reflexive:

$$((vp \uparrow_> np) \uparrow_< np) \downarrow_< (vp \uparrow_> np)$$

Translating the $\downarrow_<$ connective, with input positions 3 and 4 using Equation 9 with $n = 3$ (since $s((vp \uparrow_> np) \uparrow_< np) = 2$) and $m = 2$ gives the following partial translation.

$$\forall x_0, x_1, x_2, x_5 \| (vp \uparrow_> np) \uparrow_< np \|^{x_0, x_1, x_2, 3, 4, x_5} \multimap \| vp \uparrow_> np \|^{x_0, x_1, x_2, x_5}$$

Translating the $\uparrow_<$ connective using Equation 8 with $n = 3$ and $m = 2$ gives.

$$\forall x_0, x_1, x_2, x_5 [\| np \|^{3,4} \multimap \| vp \uparrow_> np \|^{x_0, x_1, x_2, x_5}] \multimap \| vp \uparrow_> np \|^{x_0, x_1, x_2, x_5}$$

Finally, unfolding the two $\uparrow_>$ connectives (using Equation 5) gives.

$$\forall x_0, x_1, x_2, x_5 [np(3, 4) \multimap np(x_1, x_2) \multimap \| vp \|^{x_0, x_5}] \multimap np(x_1, x_2) \multimap \| vp \|^{x_0, x_5}$$

Indicating that an object reflexive described by the given formula takes a ditransitive verb (with a first object argument spanning the input positions $3-4$ of the reflexive and a second without constraints on the position) to produce a transitive verb, a *vp* still missing an *np* spanning the positions of the second *np* of the ditransitive verb, which corresponds to the intuitive meaning of the lexical entry.

4.2 Synthetic Connectives

Morrill et al. [46] introduce the synthetic connectives[4] for the simple displacement calculus 1-D (with formulas of sort ≤ 1), whereas Valentín [57] presents

[4] Note that from the point of view of MILL1, *all* D-connectives are synthetic MILL1-connectives, that is, combinations of a series of quantifiers and a binary connective, as can already be seen from the translation in Table 5 on page 312; we will return to this point in Section 5. The synthetic connectives of D are combinations of a binary connective and an identity element. The idea for both is essentially the same: to treat a combination of rules as a single rule, which can be added to the logic as a conservative extension.

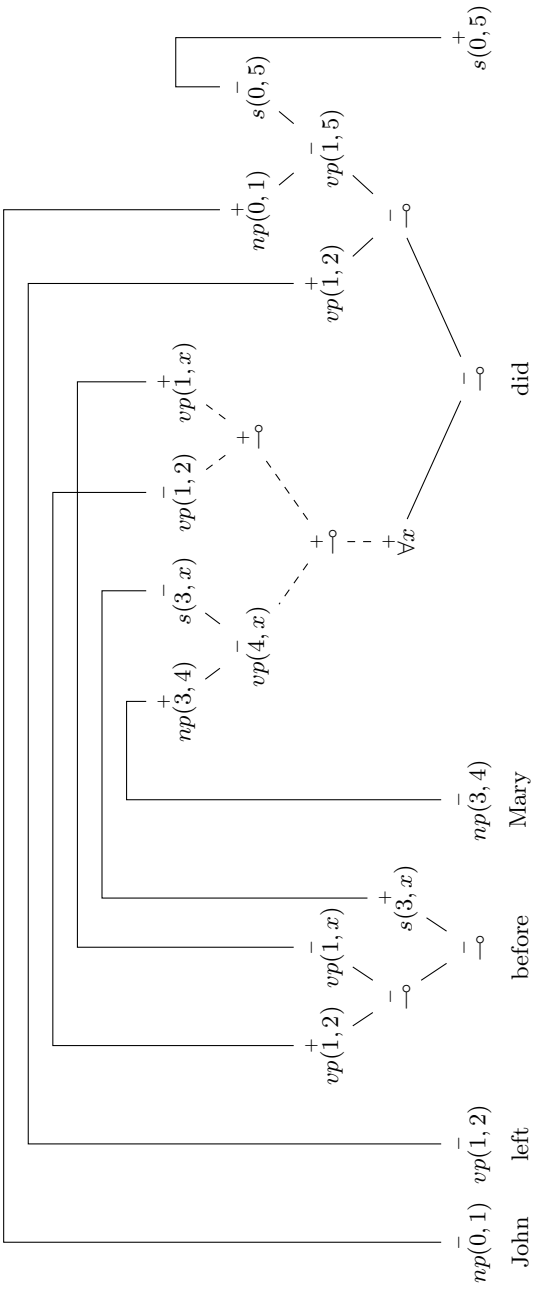


Fig. 10. Proof net for "John left before Mary did."

their natural extension to D (as well as non-deterministic versions of these connectives, which will not be treated here). The synthetic connectives can be seen as abbreviations of combinations of a connective and an identity element (I denoting the empty string ϵ and J denoting the pair of empty strings ϵ, ϵ) as shown in the list below.

$$\begin{aligned}
\breve{}A &=_{def} A \uparrow I && \text{Split}\\
\hat{}A &=_{def} A \odot I && \text{Bridge}\\
\triangleright^{-1}A &=_{def} J \setminus A && \text{Right projection}\\
\triangleright A &=_{def} J \bullet A && \text{Right injection}\\
\triangleleft^{-1}A &=_{def} A / J && \text{Left projection}\\
\triangleleft A &=_{def} A \bullet J && \text{Left injection}
\end{aligned}$$

Figures 11 and 12 show the proof rules for leftmost bridge/split and right projection/injection (the proof rules for left projection and injection as well as the proof rules for rightmost bridge and split are symmetric).

$$\frac{s,t:\breve{}A}{st:A}\ \breve{}E \qquad \frac{st:A}{s,t:\breve{}A}\ \breve{}I$$

$$\frac{t:\hat{}A \quad \begin{array}{c} s,t':A \\ \vdots \\ u[st']:C \end{array}}{u[t]:C}\ \hat{}E \qquad \frac{s,t:A}{st:\hat{}A}\ \hat{}I$$

Fig. 11. Proof rules — leftmost split, bridge

$$\frac{t:\triangleright^{-1}A}{\epsilon,t:A}\ \triangleright^{-1}E \qquad \frac{t:A}{\epsilon,t:\triangleright^{-1}A}\ \triangleright^{-1}I$$

$$\frac{v:\triangleright A \quad \begin{array}{c} t:A \\ \vdots \\ u,tu':C \end{array}}{uvu':C}\ \triangleright E \qquad \frac{t:A}{\epsilon,t:\triangleright A}\ \triangleright I$$

Fig. 12. Proof rules — right projection, injection

The synthetic connectives are translated as follows (only the leftmost split and wedge are shown, the rightmost versions are symmetric in the variables):

$$\|\breve{}A\|^{x_0,x_1,x_1,x_2,\ldots,x_n} = \|A\|^{x_0,x_2,\ldots,x_n} \tag{10}$$

$$\|\hat{}A\|^{x_0,x_2,\ldots,x_n} = \exists x_1.\|A\|^{x_0,x_1,x_1,x_2,\ldots,x_n} \tag{11}$$

$$\|\triangleright A\|^{x_0,x_0,x_1,\ldots,x_n} = \|A\|^{x_1,\ldots,x_n} \tag{12}$$

$$\|\triangleright^{-1} A\|^{x_1,\ldots,x_n} = \forall x_0.\|A\|^{x_0,x_0,x_1,\ldots,x_n} \tag{13}$$

$$\|\triangleleft A\|^{x_0,\ldots,x_n,x_{n+1},x_{n+1}} = \|A\|^{x_0,\ldots,x_n} \tag{14}$$

$$\|\triangleleft^{-1} A\|^{x_0,\ldots,x_n} = \forall x_{n+1}.\|A\|^{x_0,\ldots,x_n,x_{n+1},x_{n+1}} \tag{15}$$

In [46], the bridge connective appears exclusively in (positive) contexts $\hat{}(A \uparrow B)$ where it translates as.

$$\begin{aligned}\|\hat{}(A \uparrow B)\|^{x_0,x_2} &= \exists x_1.\|A \uparrow B\|^{x_0,x_1,x_1,x_2} \\ &= \exists x_1.[\|B\|^{x_1,x_1} \multimap \|A\|^{x_0,x_2}]\end{aligned}$$

The resulting formula indicates that it takes a B argument spanning the empty string (anywhere) to produce an A covering the original string position x_0 and x_2. Intuitively, this formalizes (in positive contexts) an A constituent with a B trace. The final translation is positive subformula of the extraction type used in [40].

The split connective ($\breve{}$, but also $\triangleleft$ and $\triangleright$) is more delicate, since it identifies string position variables. This can force the identification of variables, which means that direct application of the translation above can produce formulas which have "vacuous" quantifications, though this is not harmful (and these are easily removed in a post-processing step if desired). However, this identification of variables means that the grammars are no longer necessarily simple in the input string as discussed in Section 3.2. As an example, unfolding the formula below (which is patterned after the formula for "unfortunately" from [46]) with input variables x_i and x_j forces us to identify x_i and x_j as shown below, hence producing a self-loop in the input FSA.

$$\begin{aligned}\|\breve{}A \downarrow B\|^{x_i,x_i} &= \forall x_0, x_2.\|\breve{}A\|^{x_0,x_i,x_i,x_2} \multimap \|B\|^{x_0,x_2} \\ &= \forall x_0, x_2.\|A\|^{x_0,x_2} \multimap \|B\|^{x_0,x_2}\end{aligned}$$

Intuitively, this translation indicates that a formula of the form $\breve{}A \downarrow B$ takes its A argument at any span of the string and produces a B at the same position, with the complete formula spanning the empty string. It is, in essence, a translation of the (commutative) linear logic or LP implication into our current context. The MIX language can easily be generated using this property [45].

It is easy to find complex formulas which, together with their arguments, produce a complex cycle. The following formula spans the empty string after it combines with its C argument.

$$\begin{aligned}\|(\check{}A \downarrow B)/C\|^{x_i,x_j} &= \forall x_1 \|C\|^{x_j,x_1} \multimap \|\check{}A \downarrow B\|^{x_i,x_1} \\ &= \forall x_1 [\|C\|^{x_j,x_1} \multimap \forall x_0, x_2.[\|\check{}A\|^{x_0,x_i,x_1,x_2} \multimap \|B\|^{x_0,x_2}]] \\ &= \forall x_1 [\|C\|^{x_j,x_i} \multimap \forall x_0, x_2.[\|A\|^{x_0,x_2} \multimap \|B\|^{x_0,x_2}]] \\ &= \|C\|^{x_j,x_i} \multimap \forall x_0, x_2.[\|A\|^{x_0,x_2} \multimap \|B\|^{x_0,x_2}]\end{aligned}$$

The final line in the equation simply removes the x_1 quantifier. Since there are no longer any occurrences of the x_1 variable in the rest of the formula, this produces the equivalent formula shown. The translation specifies that the formula, which spans positions x_i to x_j takes an *np* argument spanning positions x_j to x_i, ie. the rightmost position of the *np* argument is the leftmost position of the complete formula.

If we want a displacement grammar to be simple in the input string, we can restrict the synthetic connectives used for its lexical entries to $\hat{}$, $\triangleright^{-1}$ and $\triangleleft^{-1}$; in addition, no formulas contain the units I and J except where these occurrences are instances of the allowed synthetic connectives.[5] The only lexical entries proposed for D which are not simple in this sense are those of the MIX grammar and the type for "supposedly" discussed above.

The $\triangleright^{-1}$ and $\triangleleft^{-1}$ connectives, together with atomic formulas of sort greater than 0, allow us to encode MCFG-style analyses, as we have seen them in Section 3.3, into D [6]. As an example, let's look at the unfolding of "lezen" which is assigned formula $\triangleright^{-1} np \setminus (np \setminus si)$ and assume "lezen" occupies the string position 4,5.

$$\forall x_2 \|np \setminus (np \setminus si)\|^{x_2,x_2,4,5}$$

Given that $s(np) = 0$ this reduces further to.

$$\forall x_2 \forall x_1 \|np\|^{x_1,x_2} \multimap \|np \setminus si\|^{x_1,x_2,4,5}$$

If we combine this entry with "boeken" from positions 1,2 (ie. the formula $np(1,2)$, instantiating x_1 to 1 and x_2 to 2, this gives the following partial translation for "boeken lezen"

$$\|np \setminus si\|^{1,2,4,5}$$

Similarly, "kunnen" with formula $\triangleright^{-1}(np \setminus si) \downarrow (np \setminus si)$ reduces as follows when occupying string position 3,4.

$$\forall x_2 \|(np \setminus si) \downarrow (np \setminus si)\|^{x_2,x_2,3,4}$$

[5] Alternatively, we can allow the $\check{}$, $\triangleright$ and $\triangleleft$ connectives but restrict them to cases where there is strict identity of the two string positions (disallowing instantiation of variables to obtain identity). Note that this means that formulas of the form $\check{}A \downarrow B$ are valid only in contexts spanning the empty string.

[6] Wijnholds [58] and Sorokin [55] show that D of order 1 (ie. containing only the synchronous rules, but also the bridge and projection connectives) generates the well-nested multiple context-free languages.

Which unfolds further as.

$$\forall x_2 \forall x_0 \forall x_5 \|np \setminus si\|^{x_0,x_2,4,x_5} \multimap \|np \setminus si\|^{x_0,x_2,3,x_5}$$

This combines with the previous translation of "boeken lezen", instantiating x_0 to 1, x_2 to 2 and x_5 to 5, giving "boeken kunnen lezen" with translation $\|np \setminus si\|^{1,2,3,5}$.

Finally, the tensed verb "wil" with formula $(np \setminus si) \downarrow (np \setminus s)$ unfolds at position 2,3 as.

$$\forall x_0 \forall x_3 \|np \setminus si\|^{x_0,2,3,x_3} \multimap \|np \setminus s\|^{x_0,x_3}$$

Instantiating x_0 to 1 and x_3 to 5 and combining this with the previously computed translation of "boeken kunnen lezen" produces $\|np \setminus s\|^{1,5}$ for "boeken wil kunnen lezen". Figure 13 on the next page shows a proof net derivation of the slightly more complex "(dat) Jan Henk Cecilia de nijlpaarden zag helpen voeren". Note that the axiom linkings are again fully deterministic.

5 Correctness of the Translation

The basic idea of the correctness proof is again very simple: we use the property of focused proof search and of ludics that combinations of synchronous connectives can always be seen as instances of a synthetic synchronous connective, whereas the same holds for the asynchronous connectives. Since the translations either use a combination of $\exists$ and $\otimes$ (both synchronous) or a combination of $\forall$ and $\multimap$ (both asynchronous), it follows immediately that we can treat these combinations as synthetic connectives, giving a rule to (synthetic) rule translation.

We only prove the case for the binary continuous and discontinuous connectives. As noted in Section 4.2, the extension to the bridge and projection connectives is simple, whereas the split and injection are more complicated and will be left for future research. In addition, none of the non-deterministic connectives of [46,57] are considered: their obvious translation would use the additive connectives from linear logic, which would complicate the proof nets and increase the computational complexity [28,29][7].

Lemma 1. *For every proof of $t_1 : A_1, \ldots, t_n : A_n \vdash t : C$ in D, there is a proof of its translation in MILL1.*

Proof. Refer back to Figure 8 to see the correspondence between pairs of string positions and tuples of strings more clearly. The rules are simply the translation of the natural deduction rules of D, where the string tuples have been replaced by pairs of string positions.

For the case of $\setminus E$ we are in the following situation (let $i = \frac{1}{2}(n-1)$, $j = \frac{1}{2}(m-1)$, then $x_0, \ldots, x_n$ corresponds to a i-tuple t, $x_n, \ldots, x_{n+m}$ to a j-tuple u and $x_0, \ldots, x_{n-1}, x_{n+1}, \ldots, x_{n+m}$ to their concatenation tu). The translations

[7] Though if the non-deterministic connectives occur only in negative contexts, we can treat them by simply multiplying the lexical entries.

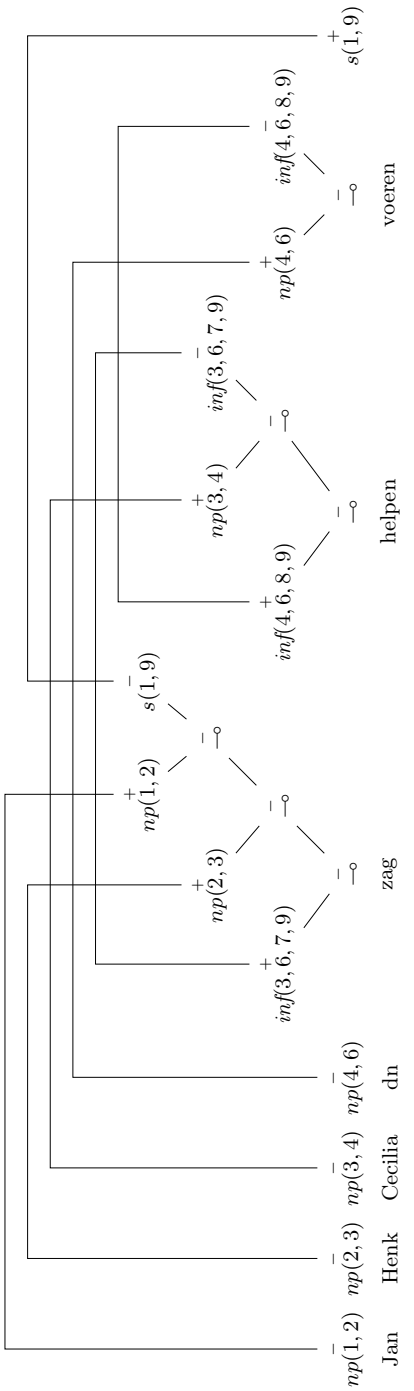


Fig. 13. Proof net for "(dat) Jan Henk Cecilia de nijlpaarden zag helpen voeren"

of A and $A \setminus C$ share point x_n and we can instantiate the universally quantified variables of the other points of A (x_0 to x_{n-1}) applying the $\forall E$ rule n times ($/$ is symmetric).

$$\dfrac{\|A\|^{x_0,\ldots,x_n} \qquad \dfrac{\dfrac{\|A \setminus C\|^{x_n,\ldots,x_{n+m}}}{\forall y_0,\ldots,y_{n-1}\|A\|^{y_0,\ldots,y_{n-1},x_n} \multimap \|C\|^{y_0,\ldots,y_{n-1},x_{n+1},\ldots,x_{n+m}}} =_{def}}{\|A\|^{x_0,\ldots,x_n} \multimap \|C\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}} \forall E \text{ (n times})}{\|C\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}} \multimap E$$

For the introduction rule, we again set i to $\frac{1}{2}(n-1)$ and j to $\frac{1}{2}(m-1)$, making $x_0,\ldots,x_n$ corresponds to a i-tuple t, $x_n,\ldots,x_{n+m}$ to a j-tuple u and $x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}$ to their concatenation tu. In this case, induction hypothesis gives us a MILL1 proof corresponding to $\Gamma, t : A \vdash tu : C$. To extend this proof to a MILL1 proof corresponding to $\Gamma \vdash u : A \backslash C$ ($/$ is again symmetric), we can continue the translated proof of $\Gamma, t : A \vdash tu : C$ as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\|A\|^{x_0,\ldots,x_n}]^i \;\ldots\; \Gamma}{\vdots}}{\|C\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}}}{\|A\|^{x_0,\ldots,x_n} \multimap \|C\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}} \multimap I_i}{\forall x_0,\ldots,x_{n-1}\|A\|^{x_0,\ldots,x_n} \multimap \|C\|^{x_0,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}} \forall I \text{ (n times})}{\|A \setminus C\|^{x_n,\ldots,x_{x+m}}} =_{def}$$

The cases for $\uparrow_>$ are shown below ($\downarrow_>$ is easily verified).

$$\dfrac{\dfrac{\dfrac{\|C \uparrow_> B\|^{x_0,x_1,x_n,\ldots,x_{n+m}}}{\forall y_2,\ldots,y_{n-1}\|B\|^{x_1,y_2,\ldots,y_{n-1},x_n} \multimap \|C\|^{x_0,y_2,\ldots,y_{n-1},x_{n+1},\ldots,x_{n+m}}} =_{def}}{\|B\|^{x_1,\ldots,x_n} \multimap \|C\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}} \forall E \text{ ($n-2$ times})} \quad \|B\|^{x_1,\ldots,x_n}}{\|C\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}} \multimap E$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[\|B\|^{x_1,\ldots,x_n}]^i \;\ldots\; \Gamma}{\vdots}}{\|C\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}}}{\|B\|^{x_1,\ldots,x_n} \multimap \|C\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}} \multimap I_i}{\forall x_2,\ldots,x_{n-1}\|B\|^{x_1,\ldots,x_n} \multimap \|C\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}} \forall I \text{ ($n-2$ times})}{\|C \uparrow_> B\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{x+m}}} =_{def}$$

Finally, the cases for $\odot_>$ are as follows.

$$\dfrac{\dfrac{\|A \odot_> B\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}}{\exists x_1 \exists x_n \|A\|^{x_0,\ldots,x_n} \otimes \|B\|^{x_n,\ldots,x_{n+m}}} =_{def} \quad \dfrac{[\|A\|^{x_0,x_1,x_n,\ldots,x_{n+m}} \otimes \|B\|^{x_1,\ldots,x_n}]^i \qquad \begin{array}{c}[\|A\|^{x_0,x_1,x_n,\ldots,x_{n+m}]^j}\ [\|B\|^{x_1,\ldots,x_n}]^j \\ \vdots \\ C\end{array}}{C} \otimes E_j}{C} \exists E_i \text{ twice}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\|A\|^{x_0,x_1,x_n,\ldots,x_{n+m}} \quad \|B\|^{x_1,\ldots,x_n}}{\|A\|^{x_0,x_1,x_n,\ldots,x_{n+m}} \otimes \|B\|^{x_1,\ldots,x_n}} \otimes I}{\exists x_n \|A\|^{x_0,x_1,x_n\ldots,x_{n+m}} \otimes \|B\|^{x_1,\ldots,x_n}} \exists I}{\exists x_1 \exists x_n \|A\|^{x_0,x_1,x_n\ldots,x_{n+m}} \otimes \|B\|^{x_1,\ldots,x_n}} \exists I}{\|A \odot_> B\|^{x_0,x_2,\ldots,x_{n-1},x_{n+1},\ldots,x_{n+m}}} =_{def}$$

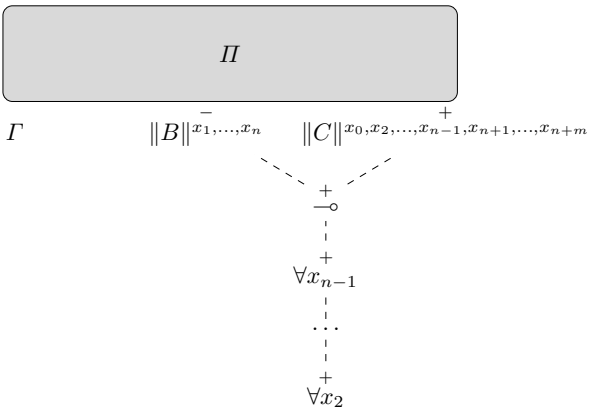
□

Lemma 2. *If the translation of a D sequent $t_1 : A_1,\ldots,t_n : A_n \vdash t : C$ is provable, then there is a D proof of $t_1 : A_1,\ldots,t_n : A_n \vdash t : C$.*

Proof. This is most easily shown using proof nets, using induction on the number of links while removing them in groups of synchronous or asynchronous links corresponding to a D connective.

If there are terminal asynchronous links, then we proceed by case analysis knowing that we are dealing the result of the translation of D formulas.

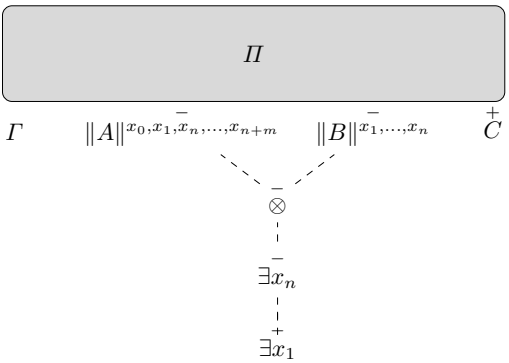
The case for $C \uparrow_> B$ looks as follows.



Given that removing the portrayed links produces a proof net Π of $\Gamma, B \vdash C$, we can apply the induction hypothesis, which gives a proof δ of $\Gamma, u : B \vdash sut : C$, which we can extend as follows.

$$\dfrac{\begin{array}{c}\Gamma \quad u : B \\ \vdots\ \delta \\ C : sut\end{array}}{s, t : C \uparrow_> B}\ \uparrow_> I$$

Similarly, the par case for $\odot_>$ looks as follows.

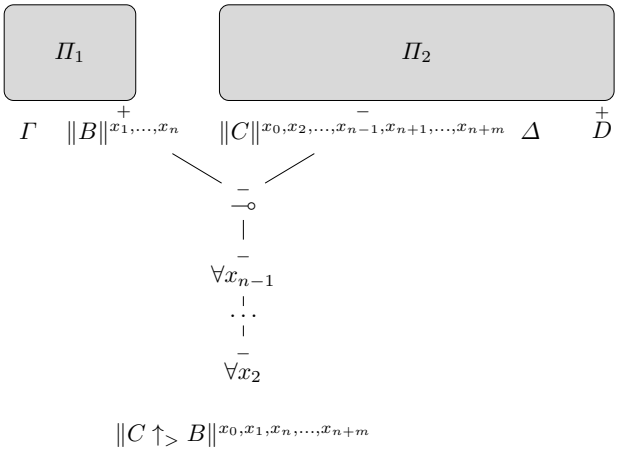


Again, we know by induction hypothesis that there is a proof δ of $\Gamma, s, t : A, u : B \vdash v[sut] : C$ and we need to show that there is a proof of $\Gamma, sut : A \odot_> B \vdash v[sut] : C$, which we do as follows.

$$\dfrac{\begin{array}{cc} \Gamma & \begin{array}{c}[s,t:A]^i \\ \vdots\ \delta \\ v[sut]:C\end{array} \quad [u:B]^i \\ sut : A \odot_> B & \end{array}}{v[sut] : C}\ \odot_> E^i$$

Suppose there are no terminal asynchronous links, then we know there must be a group of splitting synchronous links corresponding to a D connective (a series of universal links ended by a tensor link which splits the proof net into two subnets, though the synthetic connectives of Section 4.2 allow for a single universal link, which is splitting by definition, since after removal of the link, all premisses of the link are the conclusion of disjoint subnets), using the standard splitting tensor argument [14,10,4].

Suppose this group of splitting links is the translation of $\uparrow_>$, then the proof net is of the following form. Note that the translation of B corresponds to the string tuple u (with $i = \frac{1}{2}n$ components), the translation of $C \uparrow_> B$ to the string tuple sut and the translation of C to the string tuple s, t.

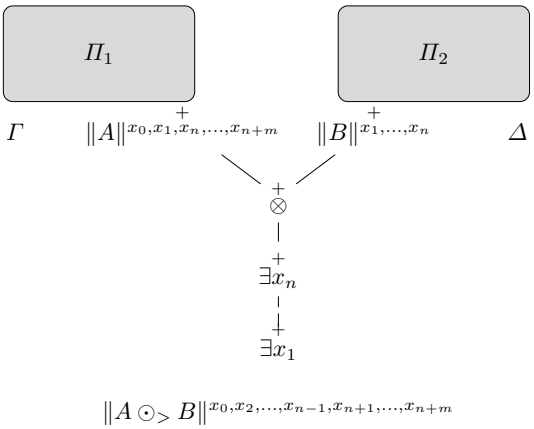


Therefore, we know by induction hypothesis that there is a proof δ_1 of $\Gamma \vdash u : B$ and a proof δ_2 of $\Delta, sut : C \vdash D$. We need to show that there is a proof $\Gamma, \Delta, s, t : C \uparrow_> B \vdash D$, which we can do as follows[8].

[8] The observant reader has surely noted that this step is just the standard sequentialization step followed by the translation of sequent calculus into natural deduction, see for example [18,42].

$$\Delta \quad \dfrac{\begin{array}{c}\Gamma \\ \vdots\ \delta_1 \\ u : B\end{array} \qquad s,t : C \uparrow_> B}{\begin{array}{c}sut : C \\ \vdots\ \delta_2 \\ D\end{array}} \uparrow_> E$$

In case the splitting tensor link and associated existential quantifiers are the translation of a $\odot_>$ formula, we are in the following case.



Induction hypothesis gives us a proof δ_1 of $\Gamma \vdash s,t : A$ and a proof δ_2 of $\Delta \vdash u : B$, which we combine as follows.

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots\ \delta_1 \\ s,t : A\end{array} \qquad \begin{array}{c}\Delta \\ \vdots\ \delta_2 \\ u : B\end{array}}{sut : A \odot_> B} \odot_> I$$

Theorem 1. *Derivability of a statement in D and derivability of the translation of this statement into MILL1 coincide.*

Proof. Immediate from Lemma 1 and 2.

The main theorem gives a simple solution to two of the main open problems from [43].

Corollary 1. *D is NP-complete.*

Proof. We have that the derivability of L, D and MILL1 are related as follows (given the translations of L and D into MILL1) $L \subset D \subset MILL1$. Therefore NP-completeness of L and MILL1 gives us NP-completeness of D.

Corollary 2. *MILL1 provides a proof net calculus for D.*

Proof. This is immediate from the fact that the D connectives correspond to synthetic MILL1 connectives. Therefore, adding these synthetic connectives to MILL1 provides a conservative extension of MILL1, which contains a proof net calculus of D. For the synchronous links, this possibility is already implicit in the proof nets of Figures 10 and 13, where the existential links are not portrayed; the combination of the asynchronous $\forall$ and the $\multimap$ link in Figure 10 can be similarly replaced by a single link and a switch to either one of the premisses of the $\multimap$ link or to one of the formulas containing the variable x.[9]

Corollary 3. *D satisfies cut elimination.*

Cut elimination for D, including the non-deterministic connectives and the units, is already proved directly in [46,57]. However, using the translation into MILL1 gives us a very easy cut elimination proof.

6 Agreement, Non-associativity and Scope Restrictions

Though I have focused only on using the first-order terms for representing string positions, I will sketch a number of other applications of the first-order terms which are orthogonal to their use for string positions, for which other extension of the Lambek calculus have introduced additional connectives and logical rules, such as the unary modalities of multimodal categorial grammar [26].

The most obvious of these applications is for the use of linguistic features, allowing us, for example, to distinguish between nominative and accusative noun phrases $np(nom)$ and $np(acc)$ but also allowing a lexical entry to fill either role by assigning it the formula $\forall x.np(x)$.

Until now, we have only seen variables and constants as arguments of predicate symbols. When we allow more complex terms, things get more interesting. Let's only consider complex terms of the form $s(T)$ — the well-known successor term from unary arithmetic not to be confused with the predicate symbol s for sentence — where T is itself a term (complex, a variable or a constant). These complex terms allow us to implement non-associativity when we need it, using the following translation (remember that the string positions are orthogonal and can be included if needed).

$$\begin{aligned} \|A \bullet B\|^x &= \|A\|^{s(x)} \otimes \|B\|^{s(x)} \\ \|C/B\|^{s(x)} &= \|B\|^{s(x)} \multimap \|C\|^x \\ \|A\backslash C\|^{s(x)} &= \|A\|^{s(x)} \multimap \|C\|^x \end{aligned}$$

The translation is parametric in a single variable x unique to the formula, which can get partially instantiated during the translation, producing a formula with a single free variable which is universally quantified to complete the translation. For example, a prototypical statement whose derivability presupposes associativity

[9] Another proof net calculus for D would be a direct adaptation of the results from Section 7 of [41]. However, this footnote is too small to contain it.

$$a/b, b/c \vdash a/c$$

translates as

$$\forall x[b(s(x)) \multimap a(x)], \forall y[c(s(y)) \multimap b(y)] \vdash \forall z[c(s(z)) \multimap a(z)]$$

which the reader can easily verify to be underivable. This translation generalizes both the translation of NL to MILL1 and the implementation of island constraints of [40].

In addition, we can handle scope restrictions in the same spirit as [6], by translating s_1 as $\forall x.s(x)$, s_2 as $\forall x.s(s(x))$ and s_3 as $\forall x.s(s(s(x)))$, which are easily verified to satisfy $s_i \vdash s_j$ for $i \leq j$ and $s_i \nvdash s_j$ for $i > j$.

Scope restrictions and island constraints are some of the iconic applications of the unary modalities of multimodal categorial grammars and I consider it an attractive feature of MILL1 they permit a transparent translation of these applications.

The use of complex terms moves us rather close to the indexed grammars [1], where complex unary term symbols play the role of a stack of indices. The linear indexed grammars [13] would then correspond to the restriction of quantified variables to two occurrences of opposite polarity[10] (or a single occurrence of any polarity; for the string position variables, they occur twice: either once as a left (resp. right) position of a positive atomic formula and once as a left (resp. right) position of a negative atomic formula or once as a left position and once as a right position of atomic formulas of the same polarity). If we restrict variables to at most two occurrences of each variable, without any restriction on the polarities, we are closer to an extension of linear indexed grammars proposed by [25], which they call partially linear PATR, and thereby closer to unification-based grammars such as LFG and HPSG. This restriction on quantified variables seems very interesting and naturally encompasses the restriction on string position variables.

These are of course only suggestions, which need to be studied in more detail in future work.

7 Conclusions and Open Questions

First-order multiplicative intuitionistic linear logic includes several interesting subsystems: multiple context-free grammars, the Lambek calculus and the Displacement calculus. In spite of this, the computational complexity of MILL1 is the same as the complexity of the universal recognition problem for each of these individual systems. In addition, it gives a natural implementation of several additional linguistic phenomena, which would require further machinery in each of the other calculi.

MILL1 satisfies all conditions of extended Lambek calculi: it has a simple proof theory, which includes a proof net calculus, it generates the mildly context-free languages, it is NP-complete and the homomorphism for semantics consists

[10] The encoding of non-associativity above is a clear violation of this constraint, since the quantified variable will occur in all atomic subformulas.

of simply dropping the quantifiers to obtain an MILL proof — though it is conceivable to use the first-order quantifiers for *semantic* features which would have a reflection in the homomorphism.

Many important questions have been left open. Do MILL1 grammars without complex terms (simple in the input string or not) generate exactly the MCFLs or strictly more? Do MILL1 grammars *with* complex terms generate exactly the indexed languages and can we get interesting subclasses (eg. partially linear PATR) by restricting the variables to occur at most twice? Are there interesting fragments of MILL1 grammars which have a polynomial recognition problem? I hope these questions will receive definite answers in the future.

References

1. Aho, A.: Indexed grammars: An extension of context-free grammars. Journal of the ACM 15(4), 647–671 (1968)
2. Andreoli, J.-M.: Logic programming with focussing proofs in linear logic. Journal of Logic and Computation 2(3) (1992)
3. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. In: Bar-Hillel, Y. (ed.) Language and Information. Selected Essays on their Theory and Application, pp. 116–150. Addison-Wesley, New York (1964)
4. Bellin, G., van de Wiele, J.: Empires and kingdoms in MLL. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic, pp. 249–270. Cambridge University Press (1995)
5. Bernardi, R., Moortgat, M.: Continuation semantics for symmetric categorial grammar. In: Leivant, D., de Queiroz, R. (eds.) WoLLIC 2007. LNCS, vol. 4576, pp. 53–71. Springer, Heidelberg (2007)
6. Bernardi, R., Moot, R.: Generalized quantifiers in declarative and interrogative sentences. Logic Journal of the IGPL 11(4), 419–434 (2003)
7. Boullier, P.: Proposal for a natural language processing syntactic backbone. Technical Report 3342, INRIA, Rocquencourt (1998)
8. Boullier, P., Sagot, B.: Efficient and robust LFG parsing: SxLfg. In: International Workshop on Parsing Technologies (2005)
9. Danos, V.: La Logique Linéaire Appliquée à l'étude de Divers Processus de Normalisation (Principalement du λ-Calcul). PhD thesis, University of Paris VII (June 1990)
10. Danos, V., Regnier, L.: The structure of multiplicatives. Archive for Mathematical Logic 28, 181–203 (1989)
11. Došen, K.: A brief survey of frames for the Lambek calculus. Zeitschrift für Mathematische Logic und Grundlagen der Mathematik 38, 179–187 (1992)
12. Fadda, M.: Geometry of Grammar: Exercises in Lambek Style. PhD thesis, Universitat Politècnica de Catalunya (2010)
13. Gazdar, G.: Applicability of indexed grammars to natural languages. In: Reyle, U., Rohrer, C. (eds.) Natural Language Parsing and Linguistic Theories, pp. 69–94. D. Reidel, Dordrecht (1988)
14. Girard, J.-Y.: Linear logic. Theoretical Computer Science 50, 1–102 (1987)
15. Girard, J.-Y.: Quantifiers in linear logic II. In: Corsi, G., Sambin, G. (eds.) Nuovi Problemi Della Logica e Della Filosofia Della Scienza, Bologna, Italy, vol. II. CLUEB (1991). Proceedings of the conference with the same name, Viareggio, Italy (January 1990)

16. Girard, J.-Y.: Linear logic: Its syntax and semantics. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic, pp. 1–42. Cambridge University Press (1995)
17. Girard, J.-Y.: Locus solum: From the rules of logic to the logic of rules. Mathematical Structures in Computer Science 11, 301–506 (2001)
18. Girard, J.-Y., Lafont, Y., Taylor, P.: Proofs and Types. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press (1988)
19. Greibach, S.A.: A new normal-form theorem for context-free phrase structure grammars. Journal of the ACM 12(1), 42–52 (1965)
20. Guerrini, S.: Correctness of multiplicative proof nets is linear. In: Fourteenth Annual IEEE Symposium on Logic in Computer Science, pp. 454–263. IEEE Computer Science Society (1999)
21. Huybregts, R.: The weak inadequacy of context-free phrase structure grammars. In: de Haan, G., Trommelen, M., Zonneveld, W. (eds.) Van Periferie naar Kern. Foris, Dordrecht (1984)
22. Joshi, A.: Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions. In: Dowty, D., Karttunen, L., Zwicky, A. (eds.) Natural Language Processing: Theoretical, Computational, and Psychological Perspectives. Cambridge University Press (1985)
23. Kaji, Y., Nakanishi, R., Seki, H., Kasami, T.: The computational complexity of the universal recognition problem for parallel multiple context-free grammars. Computational Intelligence 10(4), 440–452 (1994)
24. Kanazawa, M.: The pumping lemma for well-nested multiple context-free languages. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 312–325. Springer, Heidelberg (2009)
25. Keller, B., Weir, D.: A tractable extension of linear indexed grammars. In: Proceedings of the Seventh Meeting of the European Chapter of the Association for Computational Linguistics, pp. 75–82 (1995)
26. Kurtonina, N., Moortgat, M.: Structural control. In: Blackburn, P., de Rijke, M. (eds.) Specifying Syntactic Structures, pp. 75–113. CSLI, Stanford (1997)
27. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
28. Lincoln, P.: Deciding provability of linear logic formulas. In: Girard, Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic, pp. 109–122. Cambridge University Press (1995)
29. Lincoln, P., Scedrov, A.: First order linear logic without modalities is NEXPTIME-hard. Theoretical Computer Science 135(1), 139–154 (1994)
30. Matsuzaki, T., Miyao, Y., Tsujii, J.: Efficient HPSG parsing with supertagging and CFG-filtering. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence, pp. 1671–1676 (2007)
31. Moortgat, M.: Multimodal linguistic inference. Journal of Logic, Language and Information 5(3-4), 349–385 (1996)
32. Moortgat, M.: Symmetries in natural language syntax and semantics: The Lambek-Grishin calculus. In: Leivant, D., de Queiroz, R. (eds.) WoLLIC 2007. LNCS, vol. 4576, pp. 264–284. Springer, Heidelberg (2007)
33. Moortgat, M.: Typelogical grammar. Stanford Encyclopedia of Philosophy Website (2010), `http://plato.stanford.edu/entries/typelogical-grammar/`

34. Moortgat, M.: Categorial type logics. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, ch. 2, pp. 95–179. Elsevier/MIT Press (2011)
35. Moortgat, M., Moot, R.: Proof nets for the lambek-grishin calculus. In: Grefenstette, E., Heunen, C., Sadrzadeh, M. (eds.) Compositional Methods in Physics and Linguistics, pp. 283–320. Oxford University Press (2013)
36. Moot, R.: Proof nets and labeling for categorial grammar logics. Master's thesis, Utrecht University, Utrecht (1996)
37. Moot, R.: Proof Nets for Linguistic Analysis. PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University (2002)
38. Moot, R.: Filtering axiom links for proof nets. In: Kallmeyer, L., Monachesi, P., Penn, G., Satta, G. (eds.) Proceedings of Formal Grammar 2007 (2007) (to appear with CSLI)
39. Moot, R.: Lambek grammars, tree adjoining grammars and hyperedge replacement grammars. In: Gardent, C., Sarkar, A. (eds.) Proceedings of TAG+9, The Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms, pp. 65–72 (2008)
40. Moot, R., Piazza, M.: Linguistic applications of first order multiplicative linear logic. Journal of Logic, Language and Information 10(2), 211–232 (2001)
41. Moot, R., Puite, Q.: Proof nets for the multimodal Lambek calculus. Studia Logica 71(3), 415–442 (2002)
42. Moot, R., Retoré, C.: The Logic of Categorial Grammars. LNCS, vol. 6850. Springer, Heidelberg (2012)
43. Morrill, G.: Categorial Grammar: Logical Syntax, Semantics, and Processing. Oxford University Press (2011)
44. Morrill, G., Fadda, M.: Proof nets for basic discontinuous Lambek calculus. Journal of Logic and Computation 18(2), 239–256 (2008)
45. Morrill, G., Valentín, O.: On calculus of displacement. In: Proceedings of TAG+Related Formalisms. University of Yale (2010)
46. Morrill, G., Valentín, O., Fadda, M.: The displacement calculus. Journal of Logic, Language and Information 20(1), 1–48 (2011)
47. Murawski, A.S., Ong, C.-H.L.: Dominator trees and fast verification of proof nets. In: Logic in Computer Science, pp. 181–191 (2000)
48. Nederhof, M.-J., Satta, G.: Theory of parsing. In: Clark, A., Fox, C., Lappin, S. (eds.) The Handbook of Computational Linguistics and Natural Language Processing, pp. 105–130. Wiley-Blackwell (2010)
49. Pentus, M.: Product-free Lambek calculus and context-free grammars. Journal of Symbolic Logic 62, 648–660 (1997)
50. Pentus, M.: Lambek calculus is NP-complete. Theoretical Computer Science 357(1), 186–201 (2006)
51. Pentus, M.: A polynomial-time algorithm for Lambek grammars of bounded order. Linguistic Analysis 36(1-4), 441–471 (2010)
52. Pereira, F., Shieber, S.: Prolog and Natural Language Analysis. CSLI, Stanford (1987)
53. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88, 191–229 (1991)
54. Shieber, S.: Evidence against the context-freeness of natural language. Linguistics & Philosophy 8, 333–343 (1985)
55. Sorokin, A.: Normal forms for multiple context-free languages and displacement Lambek grammars. In: Artemov, S., Nerode, A. (eds.) LFCS 2013. LNCS, vol. 7734, pp. 319–334. Springer, Heidelberg (2013)

56. Stabler, E.: Tupled pregroup grammars. Technical report, University of California, Los Angeles (2003)
57. Valentín, O.: Theory of Discontinuous Lambek Calculus. PhD thesis, Universitat Autònoma de Catalunya (2012)
58. Wijnholds, G.: Investigations into categorial grammar: Symmetric pregroup grammar and displacement calculus, Bachelor thesis, Utrecht University (2011)