Partial Orders, Residuation, and First-Order Linear Logic

Richard Moot

To cite this version:

Richard Moot. Partial Orders, Residuation, and First-Order Linear Logic. Natural Language Processing in Artificial Intelligence, 939, Springer, pp.37-67, 2021, Studies in Computational Intelligence, 978-3-030-63786-6. lirmm-02916101

HAL Id: lirmm-02916101
https://hal-lirmm.ccsd.cnrs.fr/lirmm-02916101v1

Submitted on 17 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

arXiv:2008.06351v1 [cs.LO] 14 Aug 2020

Chapter 1
Partial Orders, Residuation, and First-Order Linear Logic

Richard Moot

Abstract We will investigate proof-theoretic and linguistic aspects of first-order linear logic. We will show that adding partial order constraints in such a way that each sequent defines a unique linear order on the antecedent formulas of a sequent allows us to define many useful logical operators. In addition, the partial order constraints improve the efficiency of proof search.

1.1 Introduction

Residuation is a standard principle which holds for the Lambek calculus and many of its variants. However, even though first-order linear logic can embed the Lambek calculus and some of its variants, linear logic formulas need not be part of a residuated triple (or pair). In this paper, we will present conditions on first-order linear logic in the form of partial order constraints which allow it to satisfy the residuation principle. We investigate the number of connectives definable this way and compare these connectives to the connectives definable in other type-logical grammars. We conclude by investigating some of the applications of these results, both in terms of linguistic modelling and in terms of improving upon the efficiency of proof search.

Richard Moot
Université de Montpellier, LIRMM, CNRS
161 rue Ada 34095 Montpellier Cedex 5
France
Tel.: +33-4-67418585
Fax: +33-4-67418500
e-mail: `firstname.lastname@institute.country`

1.2 Categorial Grammars and Residuation

Lambek introduced his syntactic calculus first as a calculus based on residuation (Lambek 1958, Section 7, with a sequent calculus in Section 8). The principle of residuation is shown as Equation 1.1.

$$A \to C \,/\, B \quad \Longleftrightarrow \quad A \bullet B \to C \quad \Longleftrightarrow \quad B \to A \,\backslash\, C \tag{1.1}$$

The Lambek calculus is then defined using just the principle of residuation together together with reflexivity and transitivity of the derivation arrow and associativity of the product '$\bullet$'. Table 1.1 lists the full set of rules of the residuation-based representation of the Lambek calculus.

Identity

$$\frac{}{A \to A}\ \textit{Refl} \qquad\qquad \frac{A \to B \quad B \to C}{A \to C}\ \textit{Trans}$$

Residuation

$$\frac{A \bullet B \to C}{A \to C \,/\, B}\ \textit{Res}_{\bullet,/} \qquad\qquad \frac{A \bullet B \to C}{B \to A \,\backslash\, C}\ \textit{Res}_{\bullet,\backslash}$$

$$\frac{A \to C \,/\, B}{A \bullet B \to C}\ \textit{Res}_{/,\bullet} \qquad\qquad \frac{B \to A \,\backslash\, C}{A \bullet B \to C}\ \textit{Res}_{\backslash,\bullet}$$

Associativity

$$\frac{}{A \bullet (B \bullet C) \to (A \bullet B) \bullet C}\ \textit{Ass}_1 \qquad \frac{}{(A \bullet B) \bullet C \to A \bullet (B \bullet C)}\ \textit{Ass}_2$$

Table 1.1 Residuation-based presentation of the Lambek calculus

In the Lambek calculus, the standard interpretation of the product '$\bullet$' is as a type of concatenation, with the implications '$\backslash$' and '$/$' its residuals. Using the residuation calculus, we can derive standard cancellation schemes such as the following.

$$\frac{\dfrac{}{C \,/\, B \to C \,/\, B}\ \textit{Refl}}{(C \,/\, B) \bullet B \to C}\ \textit{Res}_{/,\bullet} \qquad\qquad \frac{\dfrac{}{A \,\backslash\, C \to A \,\backslash\, C}\ \textit{Refl}}{A \bullet (A \,\backslash\, C) \to C}\ \textit{Res}_{\backslash,\bullet}$$

Showing us that when we compose $C \,/\, B$ with a B to its right, we produce a C, and that when we compose $A \,\backslash\, C$ with an A to its left, we produce a C.

Figure 1.1 shows a standard visual representation of the residuation principle in the form of a triangle, with the each of the vertices of the triangle corresponding to one of the Lambek calculus connectives.

We can 'read off' many of the principles from this triangle, for example, the three different ways of concatenating the elements of a residuated triple are:

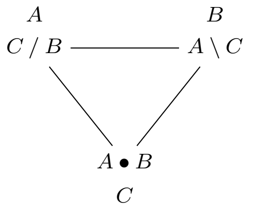


Fig. 1.1 Visual representation of residuation

1. composing A and B to produce $A \bullet B$,
2. composing C / B and B to produce C,
3. composing A and $A \setminus C$ to produce C.

The residuation presentation of the Lambek calculus naturally forms a category. This not only gives the Lambek calculus a category theoretic foundation — something Girard (2011) argues is an important, deeper level of meaning for logics — but it can also play the role of an alternative type of natural language semantics for the Lambek calculus (Coecke, Grefenstette, and Sadrzadeh 2013; Lambek 1988), to be contrasted with the more standard semantics for type-logical grammars in the tradition of Montague (1974).

Identity

$$\frac{}{A \rightarrow A}\ \mathit{Refl} \qquad \frac{A \rightarrow B \quad B \rightarrow C}{A \rightarrow C}\ \mathit{Trans}$$

Application

$$\frac{}{A \bullet (A \setminus B) \rightarrow B}\ \mathit{Appl}\backslash \qquad \frac{}{(B / A) \bullet A \rightarrow B}\ \mathit{Appl}/$$

Co-Application

$$\frac{}{A \rightarrow B \setminus (B \bullet A)}\ \mathit{Coappl}\backslash \qquad \frac{}{A \rightarrow (A \bullet B) / B}\ \mathit{Coappl}/$$

Monotonicity

$$\frac{A \rightarrow B \quad C \rightarrow D}{B \setminus C \rightarrow A \setminus D}\ \mathit{Mon}_{\backslash} \quad \frac{A \rightarrow B \quad C \rightarrow D}{A \bullet C \rightarrow B \bullet D}\ \mathit{Mon}_{\bullet} \quad \frac{A \rightarrow B \quad C \rightarrow D}{C / B \rightarrow D / A}\ \mathit{Mon}_{/}$$

Associativity

$$\frac{}{A \bullet (B \bullet C) \rightarrow (A \bullet B) \bullet C}\ \mathit{Ass}_1 \quad \frac{}{(A \bullet B) \bullet C \rightarrow A \bullet (B \bullet C)}\ \mathit{Ass}_2$$

Table 1.2 Došen's presentation of the Lambek calculus

An alternative combinatorial representation of residuation is found in Table 1.2. This presentation uses the two application principles we have derived above as axioms, and adds two additional principles of co-application, easily obtained from the identity on the product formulas together with a residuation step.

$$\dfrac{\overline{A \bullet B \to A \bullet B}\ \textit{Refl}}{A \to (A \bullet B) \,/\, B}\ \textit{Res}_{\bullet,/} \qquad \dfrac{\overline{B \bullet A \to B \bullet A}\ \textit{Refl}}{A \to B \,\backslash\, (B \bullet A)}\ \textit{Res}_{\bullet,\backslash}$$

The advantage of this presentation is that, besides transitivity, the only recursive rules are the monotonicity principles for the three connectives. This makes this presentation especially convenient for inductive proofs. For example, the completeness proofs of Došen (1992) use this presentation.

1.2.1 Residuation in Extended Lambek Calculi

Many of the extensions and variants of the Lambek calculus which have been proposed keep the principle of residuation central. For example, the multimodal Lambek calculus simply uses multiple families of residuated connectives $\{/_i, \bullet_i, \backslash_i\}$ for members i of a fixed, small set I of modes. Similarly, the unary connectives '$\Diamond$' and '$\Box$' connectives are a residuated pair (Kurtonina and Moortgat 1997; Moortgat 1996; Oehrle 2011).

However, some other formalisms do not use residuation as their central tool for defining connectives. These formalisms either add connectives corresponding to alternative algebraic principles, or abandon residuation altogether.

Formalisms in the former group take residuation for some of its connectives and *add* additional principles such as dual residuation, Galois connections, and dual Galois connections for other connectives (Areces, Bernardi, and Moortgat 2004; Bernardi and Moortgat 2010).

Formalisms in the latter group abandon residuation as a key principle (without replacing it with another algebraic principle), or only preserve it for some of their connectives. These formalisms include lambda grammars (Oehrle 1994), hybrid type-logical grammars (Kubota and Levine 2012, 2020) and first-order linear logic (Moot 2014; Moot and Piazza 2001).

1.2.2 Residuation and First-Order Linear Logic

The main theme of this paper will be to investigate what types of connectives are definable in first-order linear logic when we restrict ourselves to residuated connectives. We will look at generalised forms of concatenation and their residuals and see how we can define these in first-order linear logic.

Some of these definable connectives require us to explicitly specify partial order constraints on some of the positions to preserve the required information. The resulting grammar system then has two components: for a sentence to be grammatical, a logical statement has to be derivable (as is standard for type-logical grammars) but also a corresponding partial order definition must be consistent. This gives us a mechanism to specify the relative order of grammatical constituents (logical formulas in type-logical grammars). The property we want to preserve locally in each statement is that the strings corresponding to the antecedent formulas can be linearly ordered in a unique way.

1.3 First-Order Linear Logic

A *sequent* or a *statement* is an expression of the form $A_1, \ldots, A_n \vdash C$ (for some $n \geq 0$), which we will often shorten to $\Gamma \vdash C$. We call Γ the *antecedent*, formulas A_i in Γ *antecedent formulas*, and C the *succedent* of the statement. We assume the sequent comma is both associative and commutative and treat statements which differ only with respect to the order of the antecedent formulas to be equal. Table 1.3 shows the sequent calculus rules for first-order multiplicative intuitionistic linear logic. The $R\forall$ and $L\exists$ rule have the standard side condition that there are no free occurrences of x in Γ and C.

$$
\begin{array}{cc}
\dfrac{}{A \vdash A}\ Ax & \dfrac{\Gamma \vdash A \quad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}\ Cut \\[2ex]
\dfrac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ L\otimes & \dfrac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ R\otimes \\[2ex]
\dfrac{\Delta \vdash A \quad \Gamma, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C}\ L \multimap & \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ R \multimap \\[2ex]
\dfrac{\Gamma, A \vdash C}{\Gamma, \exists x.A \vdash C}\ L\exists^{*} & \dfrac{\Gamma \vdash A[x := t]}{\Gamma \vdash \exists x.A}\ R\exists \\[2ex]
\dfrac{\Gamma, A[x := t] \vdash C}{\Gamma, \forall x.A \vdash C}\ L\forall & \dfrac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ R\forall^{*}
\end{array}
$$

Table 1.3 The sequent calculus for first-order intuitionistic multiplicative linear logic.

The sequent calculus is decidable (the decision problem is NP complete (Lincoln 1995)) and sequent proof search can be used as a practical decision procedure (Lincoln and Shankar 1994). Decidability presupposes both cut elimination (which, as usual, is a simple enough proof even though it consists of many rule permutation cases to verify) and a restriction on the choice of t for the $L\forall$ and $R\exists$ rules. A standard solution is to use unification for this purpose, effectively delaying the choice of t to the most general term required by the axioms in backward chaining cut-free proof search. This of course requires us to verify the eigenvariable conditions for the $R\forall$ and $L\exists$ rules are still satisfied after unification. We can see this in action in the following failed attempt to prove $\forall y[a \otimes b(y)] \vdash a \otimes \forall x.b(x)$ (the reader can easily verify all other proof attempts fail as well).

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a \vdash a}\,Ax \qquad \dfrac{\dfrac{Y = x}{b(Y) \vdash b(x)}\,Ax}{b(Y) \vdash \forall x.b(x)}\,\forall R*}{a, b(Y) \vdash a \otimes \forall x.b(x)}\,R\otimes}{a \otimes b(Y) \vdash a \otimes \forall x.b(x)}\,L\otimes}{\forall y.[a \otimes b(y)] \vdash a \otimes \forall x.b(x)}\,L\forall$$

Tracing the proof from the endsequent upwards to the axioms, we start by replacing y by a fresh metavariable Y to be unified later, then follow the proof upwards to the axioms. For the b predicates, we compute the most general unifier of x and Y, which is x. But then, the antecedent of the $\forall R$ rule becomes $b(x)$, which fails to respect the eigenvariable condition for x. We can improve on the sequent proof procedure for first-order linear logic, even exploiting some of the rule permutabilities (Lincoln and Shankar 1994). However, in Section 1.3.2 we will present a proof net calculus for first order linear logic, following Girard (1991), which *intrinsically* avoids the efficiency problems caused by rule permutations.

Before we do so, however, we will briefly recall how we can use first-order linear logic for modelling natural languages.

1.3.1 First-Order Linear Logic and Natural Language Grammars

For type-logical grammars, a *lexicon* is a mapping from words to formulas in the corresponding logic. In first-order linear logic, this mapping is parametric for two position variables L and R, corresponding respectively to the left and right position of the string segment corresponding to the word. In general, for a sentence with n words, we assign the formula of word w_i (for $1 \leq i \leq n$) the string positions $i-1$ and i. This simply follows the fairly standard convention in the parsing literature to represent substrings of the input string by pairs of integers.

As noted by Moot and Piazza (2001), we can translate Lambek calculus formulas to first-order linear logic formulas as follows.

$$\|p\|^{x,y} = p(x,y) \tag{1.2}$$

$$\|A \bullet B\|^{x,z} = \exists y.\|A\|^{x,y} \otimes \|B\|^{y,z} \tag{1.3}$$

$$\|A \setminus C\|^{y,z} = \forall x.\|A\|^{x,y} \multimap \|C\|^{x,z} \tag{1.4}$$

$$\|C \,/\, B\|^{x,y} = \forall z.\|B\|^{y,z} \multimap \|C\|^{x,z} \tag{1.5}$$

Equation 1.5 states that when C/B is a formula spanning string x, y (that is, having x as its left edge and y as its right edge), that means combining it with a formula B having y as its left edge and any z as its right edge.

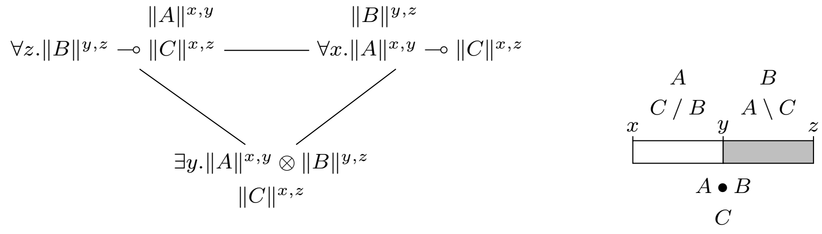

$\|A\|^{x,y}$
$\forall z.\|B\|^{y,z} \multimap \|C\|^{x,z}$
$\|B\|^{y,z}$
$\forall x.\|A\|^{x,y} \multimap \|C\|^{x,z}$
$\exists y.\|A\|^{x,y} \otimes \|B\|^{y,z}$
$\|C\|^{x,z}$
A
$C \,/\, B$
B
$A \setminus C$
x
y
z
$A \bullet B$
C

Fig. 1.2 Figure 1.1 with the corresponding translations in first-order logic

Figure 1.2 shows how this translation forms a residuated triple[1]. Note how combining (the translations of) A and B to $A \bullet B$, A and $A \setminus C$ to B, and $C \,/\, B$ and B to C all correspond to the concatenation of an x, y segment to an y, z segment to form x, z segment.

1.3.2 Proof Nets

Multiplicative linear logic has an attractive, graph-based representation of proofs called proof nets. It is relatively simple to add the first-order quantifiers to proof nets (Bellin and van de Wiele 1995; Girard 1991).

The choice for intuitionism is justified by our interest in natural language semantics: the Curry-Howard isomorphism between proofs in multiplicative intuitionistic linear logic and linear lambda terms gives us a simple and principled way of defining the syntax-semantics interface, thereby connecting our grammatical analyses to formal linguistic semantics in the tradition of Montague (1974).

Proof nets can be defined in two different ways.

1. We can define them *inductively* as instructions of how to build proof nets from simpler ones.
2. We can define proof nets as instances of a more general class of objects called *proof structures*.

Even though the inductive definition of proof nets is useful for proving all proof nets have certain properties, it is not immediately obvious how to

[1] To show this in full detail would require us to do the simple but tedious job of proving that this definition satisfies the monotonicity and Application/Co-Application principles of Table 1.2.

determine whether something is or is not a proof net, since its inductive structure is not immediately visible (unlike, say, for sequent proofs). But to distinguish proof nets we only care about the final graph structure, the inductive structure is irrelevant[2].

The second way of producing proof nets starts from proof structures. Given a sequent, there is a very direct procedure to enumerate its proof structures. Not all these proof structures will be proof nets (that is, correspond to the inductive definition of proof nets, or, equivalently, to provable sequents). A correctness condition allows us to distinguish the proof nets from other structures.

Proof structures are built from the links shown in Table 1.4. The formulas drawn above the links are called its premisses and the formulas drawn below it are called its conclusions. Each connective is assigned two links: one where it occurs as a premiss (the left link, corresponding to the left rule for the connective in the sequent calculus) and one where it occurs as a conclusion (corresponding to the right rule in the sequent calculus).

We call the formula occurrence containing the main connective of a link its *main* formula and all other formula occurrences its *active* formulas.

The logical links are divided into four groups:

1. the *tensor links* are the binary rules drawn with solid lines (the negative link for '$\multimap$' and the positive link for '$\otimes$'),
2. the *par links* are the binary rules drawn with dashed lines (the negative link for '$\otimes$' and the positive link for '$\multimap$'; *par* is the name for the multiplicative, classical disjunction of linear logic, '$\parr$'),
3. the *existential links* are the unary rules drawn with solid lines (the negative link for '$\forall$' and the positive link for '$\exists$'),
4. the *universal links* are the unary rules drawn with dashed lines and labeled with the corresponding eigenvariable (the negative link for '$\exists$' and the positive link for '$\forall$').

Definition 1 A *proof structure* is a tuple $\mathcal{S} = \langle F, L \rangle$ where F is a set of formula occurrences and L is a set of the links connecting these formula occurrences such that each local subgraph is an instantiation one of the links in Table 1.4 (for some A, B, x, t), and such that

- each formula is at most once the premiss of a link,
- each formula is at most once the conclusion of a link.

Finally, the quantifiers links and eigenvariables have the following additional conditions.

- each quantifier link uses a distinct bound variable,
- all conclusions and hypotheses of $\mathcal{S}$ are closed,

[2] Another way of thinking about this is that different ways of producing the same proof net correspond to rule permutations in the sequent calculus.

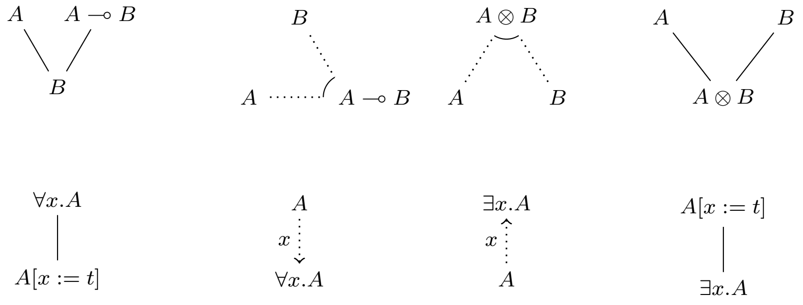


Table 1.4 Logical links for MILL1 proof structures

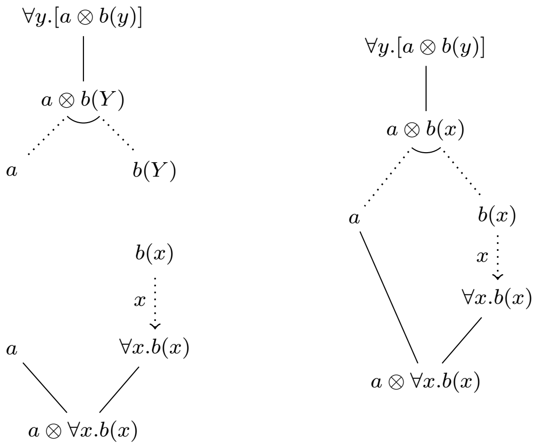
- all eigenvariables of links in $\mathcal{S}$ are used strictly, meaning that we cannot substitute a constant c_x for any set of occurrences of an eigenvariable x and obtain a proof structure with the same conclusions and hypotheses.

The formulas which are not the premisses of any link in a proof structure with hypotheses are the *conclusions* of the structure. The formulas which are not the conclusions of any link are the *hypotheses* of the structure.

Formulas which are both the premiss and the conclusion of a link in a proof structure are its *internal* formulas. All other formulas (that is, formulas which are either hypotheses or conclusions of the proof structure) are its *external* formulas. ◆

This definition essentially follows Girard (1991), incorporating the notion of strictly used eigenvariables from Bellin and van de Wiele (1995) and the proof structures with hypotheses of Danos (1990). The requirement that eigenvariables are used strictly avoids the case where, for example, a subproof $\forall x.a(x) \vdash \exists y.a(y)$ instantiates x and y to the eigenvariable z of a universal link elsewhere in the proof. Given that, by definition, we can replace such occurrences by a new constant c_z this is a minor technicality to facilitate the verification of the correctness of the universal links in a proof net.

Figure 1.3 shows, on the left hand side, the formula unfolding for the underivable sequent $\forall y[a \otimes b(y)] \vdash a \otimes \forall x.b(x)$. We want derivable sequents $A_1, \ldots, A_n \vdash C$ to correspond to proof structures (and proof *nets*) with exactly the A_i as hypotheses and C as a conclusion. The proof structure on the left hand side of Figure 1.3 has a and $b(Y)$ as additional conclusions and a and $b(x)$ as additional hypotheses. By identifying these formulas (and substituting x for Y) we obtain the proof structure shown on the right hand side of Figure 1.3. In the current case, this is the unique identification of atomic formulas producing a proof structure such that the only hypothesis is $\forall y[a \otimes b(y)]$ and the only conclusion is $a \otimes \forall x.b(x)$. In the general case, there can be many ways of identifying atomic formulas and this is the central problem for proof search using proof nets.



$\forall y.[a \otimes b(y)]$
$a \otimes b(Y)$
a
$b(Y)$
$b(x)$
x
a
$\forall x.b(x)$
$a \otimes \forall x.b(x)$
$\forall y.[a \otimes b(y)]$
$a \otimes b(x)$
a
$b(x)$
x
$\forall x.b(x)$
$a \otimes \forall x.b(x)$

Fig. 1.3 Two proof structures for the sequent $\forall y[a \otimes b(y)] \vdash a \otimes \forall x.b(x)$.

Underivability in the sequent calculus follows from the fact that there is no proof where the $\forall$ right rule is performed below the $\forall$ left rule (the intuitionistic version of this sequent $\forall y[a \wedge b(y)] \vdash a \wedge \forall x.b(x)$ *is* derivable, but it requires us to use the antecedent formula $\forall y[a \wedge b(y)]$ twice, which produces the correct order between the $\forall$ left and right rules). We will see below why the proof structure on the right of Figure 1.3 is not a proof net.

Definition 2 Given a proof structure $\mathcal{P}$ a *component* is a maximal, connected substructure containing only tensor and existential links.

We obtain the components of a proof structure by first removing the par and universal links, then taking each (maximal) connected substructure. Components can be single formulas. The components of the proof structure on the right of Figure 1.3 correspond to the induced substructures of $\{\forall y.[a \otimes b(y)], a \otimes b(x)\}$, $\{a, \forall x.b(x), a \otimes \forall x.b(x)\}$, and $\{b(x)\}$. For the first and last of these structures, the occurrences of x (all of them free) will be replaced by c_x. The second substructure contains the universal link for x (and only bound occurrences of x) and its formulas will therefore be unchanged. The corresponding sequents are given in Equations 1.6 to 1.8.

$$\forall y.[a \otimes b(y)] \vdash a \otimes b(c_x) \tag{1.6}$$

$$a, \forall x.b(x) \vdash a \otimes \forall x.b(x) \tag{1.7}$$

$$b(c_x) \vdash b(c_x) \tag{1.8}$$

The reader can verify that all of these are derivable (though we cannot combine these three proofs into a single proof of the required endsequent). Before we turn to the correctness condition, we need another auxiliary notion from Bellin and van de Wiele (1995).

Definition 3 Given a proof structure $\mathcal{P}$ and the eigenvariable x of a link l in P, the *existential frontier* of x in $\mathcal{P}$ is the set of formula occurrences $A_1, \ldots, A_n$ such that each A_i is the main formula of an existential link l_i where x occurs free in the active formula of l_i but not in its main formula A_i.

In Figure 1.3, the formula $\forall y.[a \otimes b(y)]$ is the only formula in the existential frontier of x.

To decide whether a proof structure is a proof net in linear logic, we need a correctness condition on the proof structure. Given that the two universal links correspond to sequent calculus rules with side conditions on the use of their eigenvariable, it should come as no surprise that we need to keep track of free occurrences of eigenvariables for deciding correctness. Typical correctness conditions involve graph switchings and graph contractions. Girard (1991), and Bellin and van de Wiele (1995) extend the switching condition of Danos and Regnier (1989) for first-order linear logic. Here we will extend the contraction condition of Danos (1990) to the first-order case.

Definition 4 An abstract proof structure $\mathcal{A} = \langle V, L\rangle$ is obtained from a proof structure $\mathcal{P} = \langle F, L\rangle$ by replacing each formula $A \in F$ by the set of eigenvariables freely occurring in A, plus the eigenvariable x in case A is on the existential frontier of a universal link of $\mathcal{P}$.

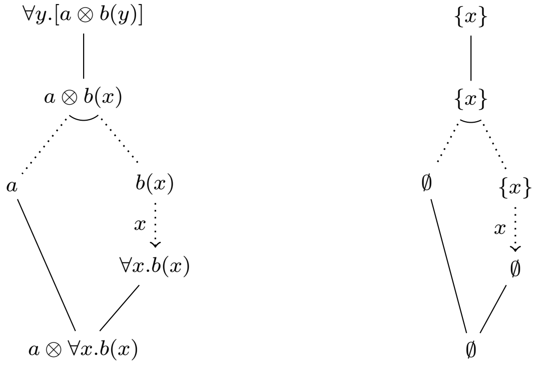


Fig. 1.4 Proof structure (left) and abstract proof structure (right) for the sequent $\forall y[a \otimes b(y)] \vdash a \otimes \forall x.b(x)$.

Figure 1.4 shows the proof structure and corresponding abstract proof structure of the proof structure we've seen before on the right of Figure 1.3. We have simply erased the formula information and kept only the information of the free variables at each node. The top node and only hypothesis of the structure, which corresponds to a closed formula (the formula $\forall y.[a \otimes b(y)]$), is on the existential frontier of x (there is an occurrence of x in the active formula of the link) and therefore has the singleton set $\{x\}$ assigned to it.

Table 1.5 shows the contractions for first-order linear logic. Each contraction is an edge contraction on the abstract proof structure, deleting an edge or a joined pair of edges, and identifying the two incident vertices v_i and v_j. The resulting vertex is incident both to all nodes incident to v_i (except v_j) and to all nodes incident to v_j (except v_i). The eigenvariables assigned to the resulting vertex are the set union of the eigenvariables assigned to v_i and v_j. For the universal contraction u the eigenvariable corresponding to the eigenvariable x of the link is removed. The contraction p verifies that the two premisses of a single par link can be joined in a single point. The contraction u verifies that all free occurrences of the eigenvariable of a universal link (and its existential border) can be found at the vertex corresponding to the premiss of the link. The contraction c contracts a component.

All contractions remove one edge (or, in the case of the par contraction p, a linked pair of edges) and keep all other edges the same, reducing the length of the paths which passed through the contracted edge by one. Contractions can produce self-loops and multiple edges between two nodes, but can never remove self-loops.

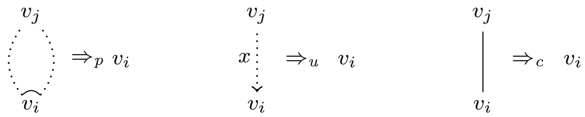


Table 1.5 Contractions for first-order linear logic. Conditions: $v_i \neq v_j$ and, for the u contraction, all occurrences of x are at v_j.

Definition 5 A proof structure is a *proof net* iff its abstract proof structure contracts to a single vertex using the contractions of Table 1.5.

The contraction system as presented is not confluent. For the critical cases, when a pair of vertices v_1 and v_2 is connected by two or more links of different types (par, universal or component), we can contract any of these multiple links connective v_1 and v_2 and produce a self-loop for all others. An easy solution to ensure confluence is to treat all self-loops as equivalent[3]. Figure 1.5 shows how the abstract proof structure of Figure 1.3 fails to contract to a single vertex. The final structure shown on the right of the figure cannot be further contracted: the par (p) contraction requires the two edges of the par link to end in the same vertex, whereas the universal (u) contraction requires all occurrences of x to be at the vertex from which the x edge is leaving.

Lemma 1 *$\Gamma \vdash C$ is derivable if and only if there is a proof net of $\Gamma \vdash C$*

See Bellin and van de Wiele (1995) for a proof, which adapts trivially to the current context.

[3] A more elegant solution for ensuring confluence would replace the right-hand side of the p and u contractions by the left-hand side of the c contraction.

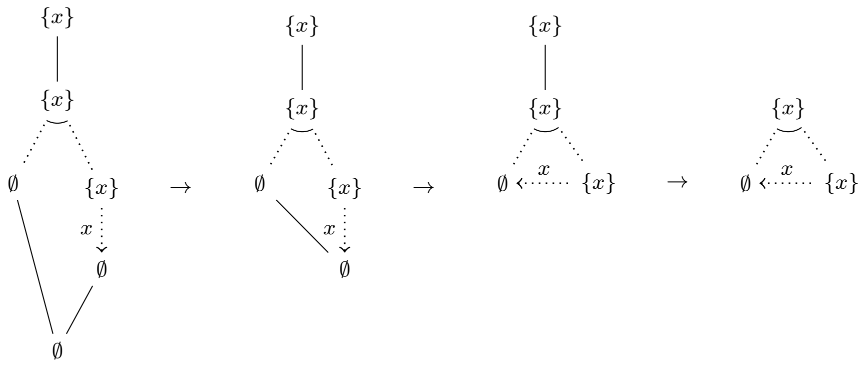


Fig. 1.5 Failed contraction sequence for the abstract proof structure on the right of Figure 1.3

1.4 Residuation and Partial Orders

So far, we have discussed proof-theoretic properties of first-order linear logic while only hinting at its applications as a formalism for natural language processing. In this section, I will suggest some principles for writing grammars using first-order linear logic, essentially in the form of constraints on the formulas. These constraints apply only to constants and variables used as string positions and not to other applications of first-order variables (such as grammatical case, island constraints and scoping constraints). The principles presented here should not be taken in a dogmatic way. It may turn out that a larger class of grammars has significant applications or better mathematical properties. The goal is merely to provide some terra firma for exploring both linguistic applications and mathematical properties. Indeed, some known classes of type-logical grammars are outside the residuated fragment investigated in this paper (Oehrle 1994), even though it is possible to follow Kubota and Levine (2020) and *combine* residuated connectives with non-residuated ones in the more general framework proposed here.

The main property we want our formulas to preserve is that we can always uniquely define a linear order on the string segments (pairs of position variables) used in the formulas of first-order linear logic. This is already somewhat of a shift with respect to standard first-order linear logic: an atomic formula $p(x_0, x_1, x_2, x_3)$ represents to string segments x_0, x_1 and x_2, x_3 without any claims about the relative order of these two segments. This gives us the freedom to build these two strings independently and let other lexical items in the grammar decide in which relative order these two segments will ultimately appear in the derived string. Adding the linear order requirement requires us to add an explicit relation between these two segments (either $x_1 \leq x_2$, for the linear order x_0, x_1, x_2, x_3, or $x_3 \leq x_0$ for the linear order x_2, x_3, x_0, x_1).

It is possible to define residuated connectives for string segments which are not linearly ordered. However, we would then be limited by the fact that any connective which linearises such segments (by ordering some of the previously unordered segments) would not be residuated. For example, suppose we want to define a connective combining two unordered string segments x_0, x_1 and x_2, x_3 by concatenating them (or 'wrapping' them around) a segment x_1, x_2 producing the complex segment x_0, x_1. This would entail the linear order to be x_0, x_1, x_2, x_3, and therefore the two segments x_0, x_1 and x_2, x_3 assigned to one of the residuals must be linearly ordered as well, simply because the alternative order x_2, x_3, x_0, x_1 has become incompatible with the linear order after concatenation. A restriction to residuated connective therefore sacrifices some flexibility for writing grammars in first-order linear logic. We will return briefly to this point in the discussion of Section 1.7.

1.4.1 Residuation for the Lambek Calculus Revisited

We have already looked at the Lambek calculus connectives and their translation into linear logic from the point of view of residuation. Figure 1.6 presents a simplified version of Figure 1.2. It focuses only on the position variables, which have been placed at the appropriate points in the triangle.

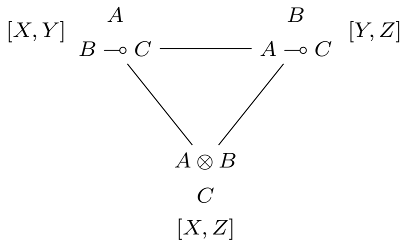


Fig. 1.6 Lambek calculus residuation translated into first-order linear logic.

Each variable occurs on exactly two of the tree points of the triangle. The place where a variable is absent determines the quantifier: '$\exists$' for '$\otimes$' (that is, the bottom node), and '$\forall$' for the two '$\multimap$' nodes (the two top nodes). Downwards movement — from A and B to $A \otimes B$, from A and $A \multimap C$ to C, and from $B \multimap C$ and B to C — corresponds to concatenation: we combine a first string with left position X and right position Y with a second string with left position Y and right position Z to form a new string starting at the left position X of the first and ending at the right position Z of the second.

A variable shared between the bottom position and one of the top positions of the figure must appear in both of these in either a left position or a right position (as, respectively, variables X and Z in Figure 1.6). A variable shared among the two top positions must appear in a right position in one and a left position in the other. Variable Y in the figure is in this case.

Seen from the point of view of string segments, the bottom element contains exactly the combination of the string segments of the left and right elements, with some of them (that is those positions occurring both left and right) concatenated.

1.4.2 Partial Orders

As a general principle, we want the left-to-right order of the position variables and constants to be globally coherent. This means that we do not want X to be left of Y at one place and to the right of it at another (at least not unless they are equal). Formally, this means that the variables in a formula and in a proof are partially ordered. More precisely, we have only argued for antisymmetry (that is $X \leq Y$ and $Y \leq X$ entail $X = Y$). To be a partial order, we also need reflexivity ($X \leq X$) and transitivity ($X \leq Y$ and $Y \leq Z$ entail $X \leq Z$, or, in our terms: if X occurs to the left of Y and Y occurs to the left of Z then X occurs to the left of Z).

We can add explicit partial order constraints to first-order linear logic, where a lexical entry specifies explicitly how some of its variables are ordered. In a system with explicit partial order constraints, a sequent is derivable if it is derivable in first-order linear logic (as before) but also satisfies all lexical constraints on the partial order. We will see in the next section how this can be useful.

Instead of using partial order constraints to obtain extra expressivity, we can also see it as a way of improving efficiency. For example, when we look at a sentence like.

1. John gave Mary flowers.

With formulas np, $((np\backslash s)/np)/np$, np, and np, we obtain the formula $np(0,1)$ for “John” and $\forall Z.np(Y,Z) \multimap \forall Y.np(2,Y) \multimap \forall X.np(X,1) \multimap s(X,Z)$ for “gave” (using the standard Lambek calculus translation). This produces the orders $0 < 1$ for “John” and $X \leq 1 < 2 \leq Y \leq Z$ for “gave”. Without any partial order constraints, it would be possible to identify $np(0,1)$ with $np(Y,Z)$. With the contraint, this would fail, since unifying Y with 0 would entail $2 \leq 0$ contradicting $0 < 2$. We will give a more detailed and interesting example in Section 1.5.3.

The residuation principle for generalised forms of concatenation requires use to be able to uniquely reconstruct the linear order of any of the three elements in a residuated triple based on the linear order of the two others. As we will see, for three position variables and two string segments, the Lambek calculus connectives are the only available residuated triple. But what happens when we increase the number of variables, and thereby the number of string positions?

Figure 1.7 shows two solutions with four position variables. The residuated triple at the top represents an infixation connective $A\backslash_{3a}C$ and a circumfixion connective $C/_{3a}B$. Note that since this last connective is represented by the pair of white rectangles, it positions itself 'around' the B formula. The infixation operation corresponds, at the string level, to the adjoining operation of tree adjoining grammars (Joshi and Schabes 1997) and to the simplest version of the discontinuous connectives of Morrill, Valentin, and Fadda (2011).

Given the concatenation operation, we can obtain its residuals by plugging them in the Application/Co-Application principles and adding the required quantifiers to make them derivable. However, the general principle is very simple and we can 'read off' the definitions directly (although the reader is invited to verify that all the Application/Co-Application principles hold). For the topmost residuated triple this gives the following definition (this connective is labeled $3a$ to indicate it is the first connective with 3 string positions).

$$\begin{aligned}
\|A \bullet_{3a} B\|^{x_0,x_3} &= \exists x_1, x_2.[\|A\|^{x_0,x_1,x_2,x_3} \otimes \|B\|^{x_1,x_2}] \\
\|A\backslash_{3a}C\|^{x_1,x_2} &= \forall x_0, x_3.[\|A\|^{x_0,x_1,x_2,x_3} \multimap \|C\|^{x_0,x_3}] \\
\|C/_{3a}B\|^{x_0,x_1,x_2,x_3} &= \|B\|^{x_1,x_2} \multimap \|C\|^{x_0,x_3}
\end{aligned}$$

We can see that the patterns are very similar to the translation of the Lambek calculus connectives: the variables shared between A and B (in the current case x_1 and x_2) are quantified existentially for the $A \otimes B$ case, the variables shared between A and C are quantified universally for the $A \multimap C$ case (x_1 and x_2 here), and the variables shared between B and C (none for this case) are quantified universally for the $B \multimap C$ case. In total each variable is quantified in exactly one of the translation cases.

The residuated triple at the bottom of Figure 1.7 assigns positions x_0, x_1 to its A formula and positions x_2, x_3 to its B formula. In this case, the positions assigned to $A \otimes B$ are underdetermined: we can say that nothing is known about the relation between x_1 and x_2, or between x_0 and x_3. This case therefore explicitly requires an additional partial order constraint to be a residuated triple. The recursive definitions are as follows.

$$\begin{aligned}
\|A \bullet_{3b} B\|^{x_0,x_1,x_2,x_3} &= \|A\|^{x_0,x_1} \otimes \|B\|^{x_2,x_3} \\
\|A\backslash_{3b}C\|^{x_2,x_3} &= \forall x_0, x_1.[\|A\|^{x_0,x_1} \multimap \|C\|^{x_0,x_1,x_2,x_3}] \\
\|C/_{3b}B\|^{x_0,x_1} &= \forall x_2, x_3.[\|B\|^{x_2,x_3} \multimap \|C\|^{x_0,x_1,x_2,x_3}]
\end{aligned}$$

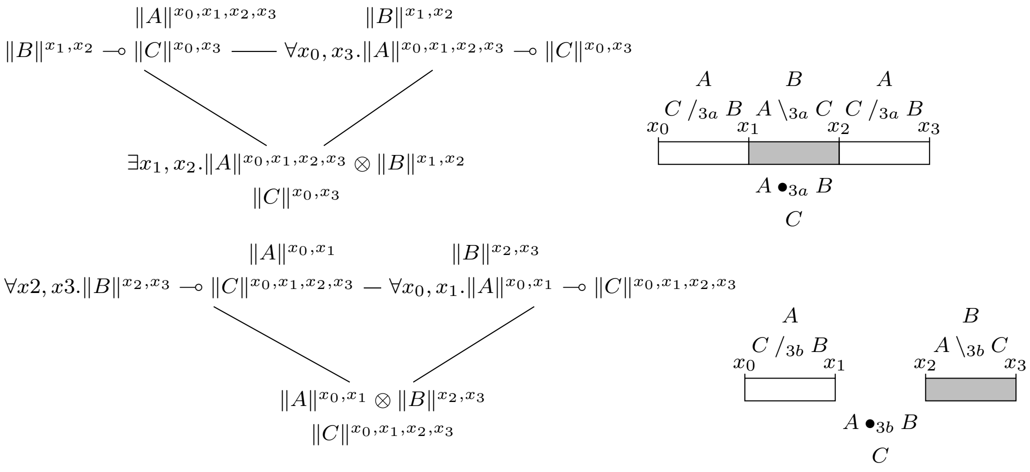

$\|A\|^{x_0,x_1,x_2,x_3}$
$\|B\|^{x_1,x_2}$
$\|B\|^{x_1,x_2} \multimap \|C\|^{x_0,x_3}$
$\forall x_0, x_3 . \|A\|^{x_0,x_1,x_2,x_3} \multimap \|C\|^{x_0,x_3}$
$\exists x_1, x_2 . \|A\|^{x_0,x_1,x_2,x_3} \otimes \|B\|^{x_1,x_2}$
$\|C\|^{x_0,x_3}$
A
B
A
$C /_{3a} B$
$A \backslash_{3a} C$
$C /_{3a} B$
x_0
x_1
x_2
x_3
$A \bullet_{3a} B$
C
$\|A\|^{x_0,x_1}$
$\|B\|^{x_2,x_3}$
$\forall x2, x3 . \|B\|^{x_2,x_3} \multimap \|C\|^{x_0,x_1,x_2,x_3}$
$\forall x_0, x_1 . \|A\|^{x_0,x_1} \multimap \|C\|^{x_0,x_1,x_2,x_3}$
$\|A\|^{x_0,x_1} \otimes \|B\|^{x_2,x_3}$
$\|C\|^{x_0,x_1,x_2,x_3}$
A
$C /_{3b} B$
x_0
x_1
B
$A \backslash_{3b} C$
x_2
x_3
$A \bullet_{3b} B$
C

Fig. 1.7 Two families of connectives with three segments and four position variables

The key case is $A \bullet_{3b} B$, where there would be a loss of information in the information passed to the two subformulas without the additional constraint the $x_1 \leq x_2$.

Now it may seem that this connective is just a formal curiosity. However, it is essentially this pattern, notably the $A \backslash_{3b} C$ connective, which figures in the analysis of the well-known crossed dependencies for Dutch verb clusters of Morrill, Valentin, and Fadda (ibid.).

1.5 The General Case

Given linear order of the string position variables, each additional string variable increases the number of possible connectives. We have seen the case for three position variables (the Lambek calculus connectives) and the two residuated triples for four position variables. Are these the only possibilities? And, more generally, how many residuated connectives exist for k position variables.

We want our residuated triples to combine two sequences of components, one containing elementary segments labeled a (corresponding to the left residual) and the other containing elementary segments labeled b (corresponding to the right residual) while allowing an 'empty' component between two other components (but not at the beginning or end of a generalised concatenation). Residuated triples can use the 'empty' segment **1**, which corresponds to a sort of placeholder or hole for another segment.

1. the first segment must be a (concatenations with b as first segment are obtained by left-right symmetry of the residuated triple),
2. there can be no consecutive a segments (that it, if two a segments have already been concatenated, we 'lose' the internal structure),
3. for the same reasons, there can be no consecutive b segments,
4. consecutive $\mathbf{1}$ segments do not increase expressivity and are therefore excluded,
5. there must be at least one b segment,
6. the last segment cannot be $\mathbf{1}$ (and, as a consequence of item 1, neither can the first segment).

The finite state automaton shown in Figure 1.8 generates all strings which satisfy these requirements. From the start state q_0, the only valid symbol is a. The condition that we cannot repeat the last symbol then ensures that the states where the last symbol was a (states q_1 and q_4) can only continue with a $\mathbf{1}$ or a b symbol. Similarly, the states where the last symbol was $\mathbf{1}$ (states q_2 and q_5) can only continue with an a or a b symbol, and the state where the last symbol was b (state q_3) can only continue with $\mathbf{1}$ or a. Finally, the states q_3, q_4 and q_5 denote the states where we have seen at least one b symbol. These are accepting states except for q_5 (because its last symbol is $\mathbf{1}$).

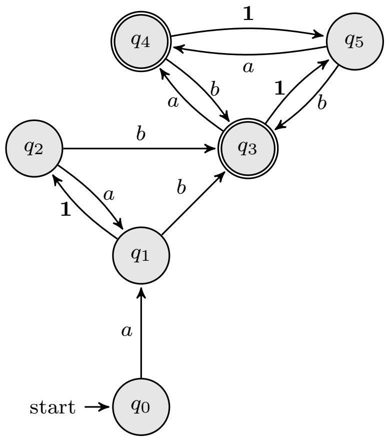


Fig. 1.8 Finite state automaton of concatenation-like operations.

We can now show that this machine generates only one two-symbol string ab (corresponding to three string positions and to the simple concatenation of a and b) and two three-symbol strings (with four string positions, namely $a\mathbf{1}b$ and aba).

Table 1.6 shows the concatenation-like operations definable with two, three, and four total string segments. The a segments correspond to empty rectangles, the b segments to filled rectangles and the $\mathbf{1}$ segments to empty

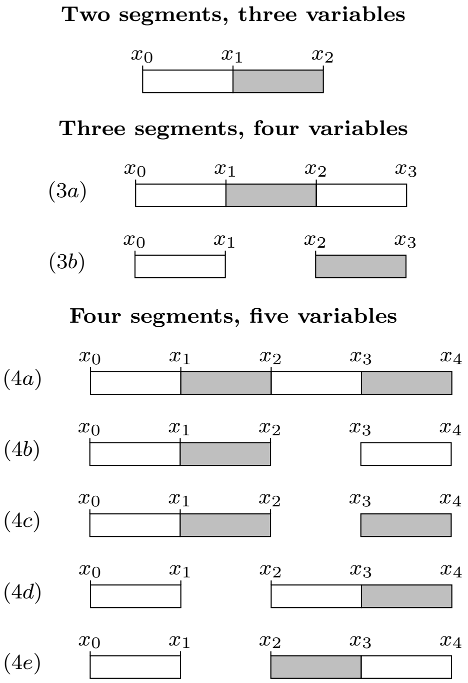


Table 1.6 Concatenation-like operations for two to four string segments

spaces between the other segments. We can read off the free variables and their linear order for each of the subformulas of a residuated triple.

For example, the A (and $B \multimap C$) segments of the first item with four segments corresponds to a formula with free variable x_0, x_1, x_2, x_3 (in that linear order) whereas the B (and $A \multimap C$) formula corresponds to a formula with free variables x_1, x_2, x_3, x_4. Finally, the result of the concatenation formula C (and $A \otimes B$) corresponds to variables x_0, x_4, with three separate concatenation operations. We concatenate $a\mathbf{1}a$ to $b\mathbf{1}b$ to produce $abab$. The number of variables shared by the left branch A and the right branch B corresponds to the number of concatenations of elementary segments. If we name this residuated triple $4a$, its recursive definition is as follows.

$$
\begin{aligned}
\|A \bullet_{4a} B\|^{x_0,x_4} &= \exists x_1, x_2, x_3.[\|A\|^{x_0,x_1,x_2,x_3} \otimes \|B\|^{x_1,x_2,x_3,x_4}] \\
\|A\backslash_{4a}C\|^{x_1,x_2,x_3,x_4} &= \qquad \forall x_0.[\|A\|^{x_0,x_1,x_2,x_3} \multimap \|C\|^{x_0,x_4}] \\
\|C/_{4a}B\|^{x_0,x_1,x_2,x_3} &= \qquad \forall x_4.[\|B\|^{x_1,x_2,x_3,x_4} \multimap \|C\|^{x_0,x_4}
\end{aligned}
$$

As another example, the fourth item with four segments (and five variables) assign the A (and $B \multimap C$) segments the sequence of variables x_0, x_1, x_2, x_3, the B and (and $A \multimap C$) formula the variables x_3, x_4, and the C (and $A \otimes B$ formula) the variables x_0, x_1, x_2, x_4. If we name this residuated triple $4d$, we obtain the following recursive definitions.

$$\begin{aligned}
\|A \bullet_{4d} B\|^{x_0,x_1,x_2,x_4} &= \exists x_3.[\|A\|^{x_0,x_1,x_2,x_3} \otimes \|B\|^{x_3,x_4}] \\
\|A\backslash_{4d}C\|^{x_3,x_4} &= \forall x_0, x_1, x_2.[\|A\|^{x_0,x_1,x_2,x_3} \multimap \|C\|^{x_0,x_1,x_2,x_4}] \\
\|C/_{4d}B\|^{x_0,x_1,x_2,x_3} &= \forall x_4.[\|B\|^{x_3,x_4} \multimap \|C\|^{x_0,x_1,x_2,x_4}]
\end{aligned}$$

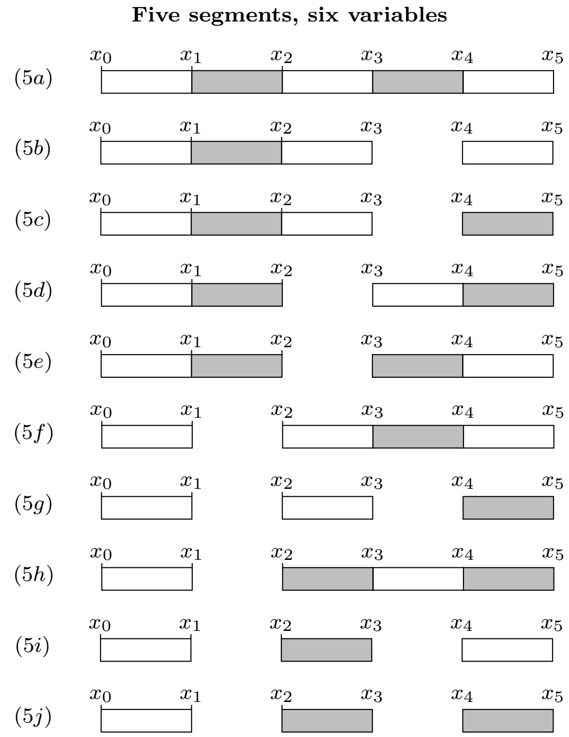


Table 1.7 Concatenation-like operations for five string segments

Table 1.7 shows the concatenation-like operations definable with five string segments. We give an example of only one of these, because it illustrates a new pattern. As we have seen, some concatenation-like operations require additional order constraints to uniquely define a linear order, for each subformula, on all variables occurring exactly once in this subformula. This was the case for the second possibility with three segments, where we could not infer the order between the A segment x_0, x_1 and the B segment x_2, x_3 without explicitly requiring $x_1 \leq x_2$.

The second item of Table 1.7, $5b$, shows a different type of underdetermination. When we give the translation of the table entry into a residuated triple $5b$, we obtain the following.

$$\begin{aligned}
\|A \bullet_{5b} B\|^{x_0,x_3,x_4,x_5} &= \exists x_1, x_2.[\|A\|^{x_0,x_1,x_2,x_3,x_4,x_5} \otimes \|B\|^{x_1,x_2}] \\
\|A\backslash_{5b} C\|^{x_1,x_2} &= \forall x_0, x_3, x_4, x_5.[\|A\|^{x_0,x_1,x_2,x_3,x_4,x_5} \multimap \|C\|^{x_0,x_3,x_4,x_5}] \\
\|C/_{5b} B\|^{x_0,x_1,x_2,x_3,x_4,x_5} &= \|B\|^{x_1,x_2} \multimap \|C\|^{x_0,x_3,x_4,x_5}
\end{aligned}$$

The problematic connective here is $C/_{5b}B$. The order information of its subformulas B and C does not allow us to unambiguously reconstruct the full order: it is compatible with an alternative linear order $x_0, x_3, x_4, x_1, x_2, x_5$, which is the sixth entry $5f$ in Table 1.7. The left residuals of $5b$ and $5f$ cannot be distinguished without an explicit constraint on the linear order for the left residual. In the case above, we need to explicitly state that $x_0 \leq x_1$ and $x_2 \leq x_3$ (technically, since x_0 is the leftmost element of the triple, the first constraint is superfluous).

1.5.1 How Many Residuated Connectives Are There for Concatenation-Like Operations?

Since the finite state automaton of Figure 1.8 is deterministic, each transition produces a symbol and it is therefore easy to use the automaton to enumerate the number of strings[4] of a certain length k.

We can also use the machine to directly compute the number of words, either by using a standard dynamic programming approach or by solving the linear recurrence specified by the automaton to produce a closed form. For example, there is a single length 1 path to q_1 (the path from the start state q_0). For paths of length greater than 1, the number of paths to q_1 of length k is equal to the number of paths of length $k-1$ to q_2. In general, the number of paths of length k to a state is the sum of the paths of length $k-1$ which can reach this state in one step. Writing out the full definition then gives the following set of linear recurrences, where $p[Q][K]$ denotes the number of paths of length K which reach state Q. In addition, $p[k]$ denotes the number of accepting paths of length k and it is the sum of the number of paths to the two accepting states q_3 and q_4.

[4] In the literature on finite state automata it is common to refer to sequences of symbols produced by such an automaton as "words". However, we reserve "words" to refer to elements in the lexicon of a type-logical grammar and exclusively use "string" for a sequence of symbols produced by a finite state automaton.

$$\begin{aligned}
p[q_1][1] &= 1\\
p[q_1][k] &= p[q_2][k-1]\\
p[q_2][k] &= p[q_1][k-1]\\
p[q_3][k] &= p[q_4][k-1] + p[q_5][k-1] + p[q_1][k-1] + p[q_2][k-1]\\
p[q_4][k] &= p[q_3][k-1] + p[q_5][k-1]\\
p[q_5][k] &= p[q_3][k-1] + p[q_4][k-1]\\
p[k] &= p[q_3][k] + p[q_4][k]
\end{aligned}$$

We can simplify these equations by observing that for each k there is exactly one path arriving at q in k steps from either q_2 (for $k-1$ even) or q_1 (for $k-1$ odd). So we can simplify $p[q_1][k-1] + p[q_2][k-1]$ to 1. In addition, because of the symmetries in the automaton, there are exactly as many paths reaching q_4 as there are reaching q_5 for any k, so we can replace $p[q_5][k]$ by $p[q_4][k]$ without changing the results. This simplifies the equations as follows.

$$\begin{aligned}
p[q_3][0] = p[q_3][1] &= 0\\
p[q_4][0] = p[q_4][1] = p[q_4][2] &= 0\\
p[q_3][k] &= 2 * p[q_4][k-1] + 1 && (k > 1)\\
p[q_4][k] &= p[q_3][k-1] + p[q_4][k-1] && (k > 2)\\
p[k] &= p[q_3][k] + p[q_4][k]
\end{aligned}$$

We can now show the following.

$$p[q_3][k] = p[q_4][k] \qquad (\textit{for } k \textit{ odd}) \tag{1.9}$$

$$p[q_3][k] = p[q_4][k] + 1 \qquad (\textit{for } k \textit{ even and} \geq 2) \tag{1.10}$$

This is an easy induction: it is trivially true for $k = 1$. Now assume Equations 1.9 and 1.10 hold for all $k' < k$.

If k is even, $k-1$ is odd, and induction hypothesis gives us $p[q_3][k-1] = p[q_4][k-1]$ and we need to show that $p[q_3][k] = p[q_4][k]+1$, given $k \geq 2$. Using $p[q_3][k-1] = p[q_4][k-1]$, we can simplify $p[q_4][k] = p[k_3][k-1] + p[p4][k-1]$ to $p[q_4][k] = 2 * p[q_4][k]$. But since $p[q_3][k] = 2 * p[q_4][k] + 1$ we have therefore shown that $p[q_3][k] = p[q_4][k] + 1$.

If k is odd, $k-1$ is even, and induction hypothesis gives us $p[q_3][k-1] = p[q_4][k-1] + 1$. We have already verified $k = 1$, so we only need to verify $k \geq 3$. Again, using $p[q_3][k-1] = p[q_4][k-1] + 1$ to substitute $p[q_4][k-1] + 1$ for $p[q_3][k-1]$ in the equation for $p[q_4][k]$ produces $p[q_4][k] = 2*p[q_4][k-1]+1$ and we have therefore shown that $p[q_3][k] = p[q_4][k]$ as required.

We can use Equations 1.9 and 1.10 to further simplify the machine equations and end up with the following.

For k odd, we have

$$\begin{aligned} p[q_4][k] &= 2 * p[q_4][k-1] + 1 \\ p[q_3][k] &= 2 * p[q_3][k-1] - 1 \end{aligned}$$

and therefore

$$\begin{aligned} p[k] &= 2 * p[q_4][k-1] + 1 + 2 * p[q_3][k-1] - 1 \\ &= 2 * p[k-1] \end{aligned}$$

For k even and ≥ 2, we have

$$\begin{aligned} p[q_4][k] &= 2 * p[q_4][k-1] \\ p[q_3][k] &= 2 * p[q_3][k-1] + 1 \end{aligned}$$

and therefore

$$\begin{aligned} p[k] &= 2 * p[q_4][k-1] + 2 * p[q_3][k-1] + 1 \\ &= 2 * p[k-1] + 1 \end{aligned}$$

The number of residuated connectives definable in first-order linear logic with partial order constraints therefore corresponds to sequence A000975 of the Online Encyclopedia of Integer Sequences (OEIS Foundation 1964). Giving us the sequence the following sequence of the number of residuated triples

$$0, 1, 2, 5, 10, 21, 42, 85, 170, 341, 682, \ldots$$

for $1, 2, 3, \ldots$ total string components and for $2, 3, 4, \ldots$ total string positions[5].

1.5.2 Well-Nestedness

One important property often imposed on linguistic formalisms is the property of well-nestedness (Kallmeyer 2010). In the current context, this means that with respect to the finite state automaton of Figure 1.8, we restrict ourselves to those paths where, whenever we encounter an a symbol after a b, there can be no further b symbols. In other words, the bs are sandwiched between the a, but not inversely. The simplest non-wellnested combination is $abab$.

We can write out the linear recurrences as before. The number of paths to q_3 and q_5 are easily established to be the following.

[5] A closed form solution for this recurrence is the following (OEIS Foundation 1964).

$$p[n] = \left\lceil \frac{2(2^n - 1)}{3} \right\rceil$$

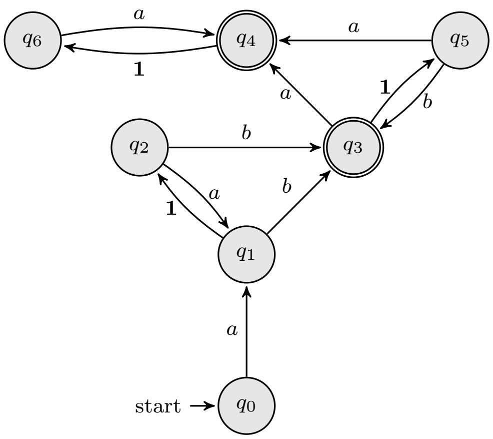


Fig. 1.9 Variant of the finite state automatic of Figure 1.8 for well-nested operations

$$\begin{aligned}
p[q_3][2k] &= k \\
p[q_3][2k+1] &= k \\
p[q_5][2k] &= k-1 \qquad\qquad k>0 \\
p[q_5][2k+1] &= k
\end{aligned}$$

Then given that $p[q_6][n] = p[q_4][n-1]$, we can establish the number of paths to q_4 as follows.

$$\begin{aligned}
p[q_4][2k] &= q[3][2k-1] + q[5][2k-1] + q[4][2(k-1)] \\
p[q_4][2k+1] &= q[3][2k] \qquad + q[5][2k] \qquad + q[4][2(k-1)+1]
\end{aligned}$$

Simplifying the above recurrence with the calculated values for q_3 and q_5 produces the following.

$$\begin{aligned}
p[q_4][2k] &= q[4][2(k-1)] \qquad + 2(k-1) &&= k(k-1) \\
p[q_4][2k+1] &= q[4][2(k-1)+1] + 2k &&= k^2
\end{aligned}$$

The number of paths to a final state of the automaton is then obtain by simply adding the number of paths to q_3 to those to q_4, which gives us the following solutions after some elementary arithmetic.

$$\begin{aligned}
p[2k] &= p[q_3][2k] + p[q_4][2k] & \\
&= k + k(k-1) & = k^2 \\
p[2k+1] &= p[q_3][2k+1] + p[q_4][2k+1] & \\
&= k + k^2 & = k(k+1)
\end{aligned}$$

An alternative way to state this same solution is the following.

$$p[n] = \lfloor (n/2) \rfloor * \lceil n/2 \rceil$$

Accordingly, the number of well-nested residuated connectives is the following

$$0, 1, 2, 4, 6, 9, 12, 16, 20, 25, 30, 36, \ldots$$

for $1, 2, 3, 4, 5, \ldots$ segments and $2, 3, 4, 5, 6, \ldots$ string position variables. This corresponds to sequence A002620 of the Online Encyclopedia of Integer Sequences (OEIS Foundation 1964).

As a sanity check, we can verify that 4 out of 5 of the four segment possibilities of Table 1.6 are well-nested (only $4a$ is not) whereas 6 out of 10 of the five segment possibilities of Table 1.7 are well-nested (the exceptions being $5a$, $5c$, $5d$, and $5h$).

1.5.3 Partial Order Constraints in Practice

As an example, we will give an analysis of the sentence ‘John left before Mary did’ based on the analysis of Morrill, Valentin, and Fadda (2011). We assign ‘John’ and ‘Mary’ the formulas $np(0,1)$ and $np(3,4)$ respectively (based on their positions in the string). We assign ‘left’ the formula $np\backslash s$, which at positions $1, 2$ translates to $\forall A.[np(A,1) \multimap s(A,2)]$. We assign the ‘before’ the formula $((np\backslash s)\backslash(np\backslash s))/s$ (that is, it selects a sentence to its right and a $vp = np\backslash s$ to its left to return a vp). This translates to the following formula.

$$\forall B.[s(3,B) \multimap \forall D.[\forall x.[np(x_0,D) \multimap s(x_0,2)] \multimap \forall C.[np(C,D) \multimap s(C,B)]]]$$

Finally, the complicated formula is assigned to ‘did’. In terms of the residuated connectives it is assigned to formula $((vp/_{3a}vp)/vp)\backslash_{4d}(vp/_{3a}vp)$. As a reminder, we restate the relevant translations of the connectives occurring in this formula.

$$\begin{aligned}
\|C/B\|^{x_0,x_1} &= \forall x_2.[\|B\|^{x_1,x_2} \multimap \|C\|^{x0,x_2}]\\
\|C/_{3a}B\|^{x_0,x_1,x_2,x_3} &= \|B\|^{x_1,x_2} \multimap \|C\|^{x0,x_3}\\
\|A\backslash_{4d}C\|^{x_3,x_4} &= \forall x_0,x_1,x_2.[\|A\|^{x_0,x_1,x_2,x_3} \multimap \|C\|^{x_0,x_1,x_2,x_4}]
\end{aligned}$$

Given these translations, we can translate this formula into first-order linear logic as follows.

$$\begin{aligned}
&\|((vp/_{3a}vp)/vp)\backslash_{4d}(vp/_{3a}vp)\|^{4,5}\\
&\forall F,I,J.\|(vp/_{3a}vp)/vp\|^{F,I,J,4} \multimap \|vp/_{3a}vp\|^{F,I,J,5}\\
&\forall F,I,J.[\forall x_1, \|vp\|^{4,x_1} \multimap \|vp/_{3a}vp\|^{F,I,J,x_1}] \multimap \|vp\|^{I,J} \multimap \|vp\|^{F,5}\\
&\forall F,I,J.[\forall x_1, \|vp\|^{4,x_1} \multimap \|vp\|^{I,J} \multimap \|vp\|^{F,x_1}] \multimap \|vp\|^{I,J} \multimap \|vp\|^{F,5}
\end{aligned}$$

We have left the final $vp = np\backslash s$ subformulas untranslated. We can see that aside for some fairly complicate manipulation with string positions, to which we will return shortly, the formula simply indicates it select a function of two vp's into a single vp to become a vp modifier.

Given these translations, Figure 1.10 shows the formula unfolding for the sentence 'John left before Mary did'. Each node indicates the corresponding linear order on the variables occurring once in this subformula. The complex formula 'did' has many branchings but referring back to the position variables allows to to identify which node corresponds to which subformula in the translation. For example, the node labeled F, I, J, x_1 corresponds to (the leftmost occurrence of) the formula $vp/_{3a}vp$.

Table 1.8 shows the possible matchings between positive and negative atomic formulas. The rows of the table represent the choices for the positive formulas, whereas the columns represent the choices for the negative formulas. The positive $s(0,5)$ formula represents the conclusion, the other positive formulas are those which are premisses of their link. Each of the candidate proof structures for the goal sequent is one of the perfect matchings of the positive with the negative formulas. However, since there are $n!$ matchings, brute force search is to be avoided as much as possible. Just for the current example, there are $5! = 120$ choices for the np formulas and the same number of choices for the s formulas. Given that these choices are independent, this amounts to a total of 14.400 different possible proof structures.

Fortunately, there are quite a number of constraints on the possible connections in the proof structure. The partial order constraints are one of those. Figure 1.11 summarises the partial order constraints for the structure of Figure 1.10. The partial order constraints allow us to avoid connecting $s(3,B)$ to $s(A,2)$ since it fails both the $3 \leq B$ constraint (when unifying B to 2) and the $A \leq 1$ constraint (when unifying A to 3). A slightly less obvious connection which fails the constraint is the connection between $s(x_2,x_1)$ and $s(H,J)$.

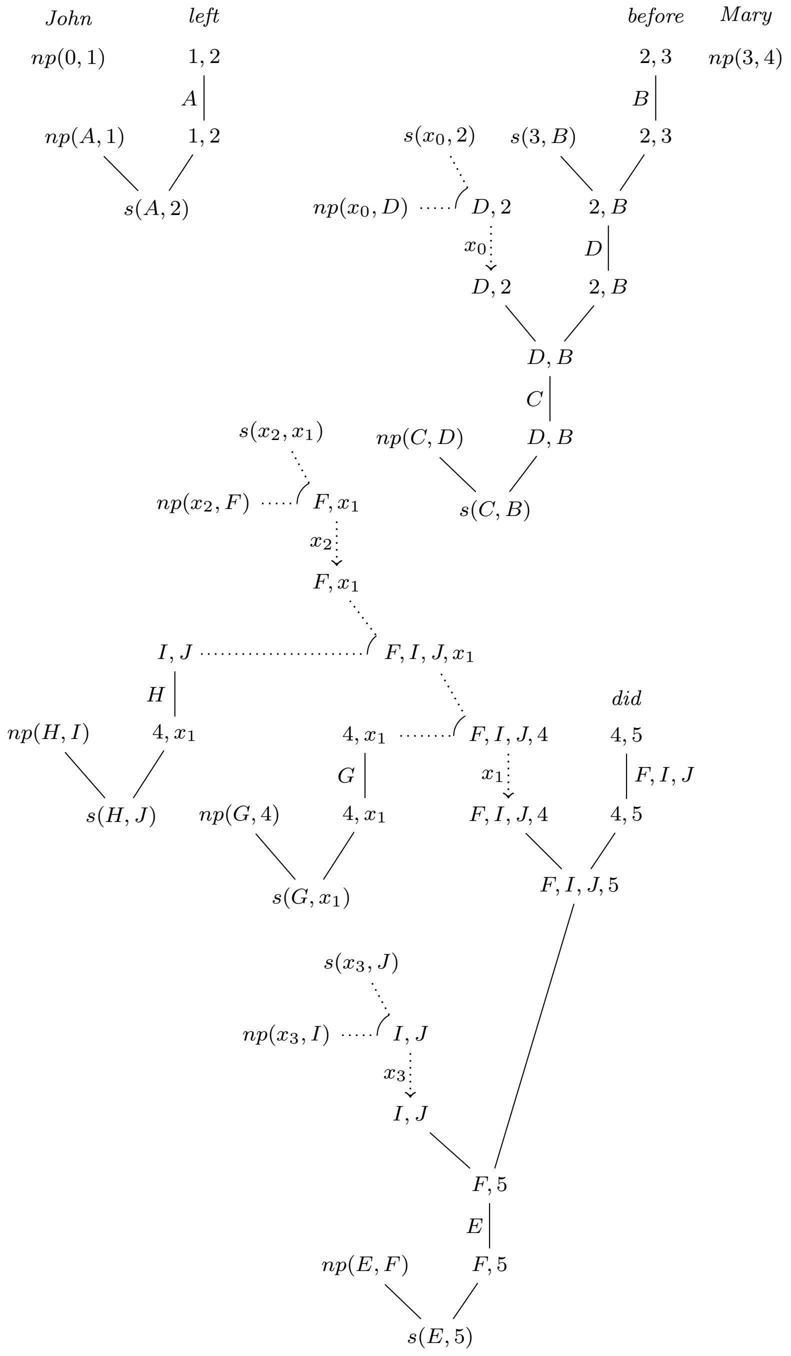


Fig. 1.10 Proof structure formed from the formula unfolding for 'John left before Mary did'

	$np(x_0, D)$	$np(0, 1)$	$np(3, 4)$	$np(x_2, F)$	$np(x_3, I)$
$np(C, D)$				2	
$np(A, 1)$					9
$np(E, F)$		4			
$np(G, 4)$			8		
$np(H, I)$	10				

	$s(A, 2)$	$s(E, 5)$	$s(H, J)$	$s(G, x_1)$	$s(C, B)$
$s(x_0, 2)$			6		
$s(3, B)$				7	
$s(x_3, J)$	5				
$s(x_2, x_1)$					1
$s(0, 5)$		3			

Table 1.8 Possible axiom connectives for the proof structure in Figure 1.10, with the columns representing the negative occurrences and the rows the positive ones.

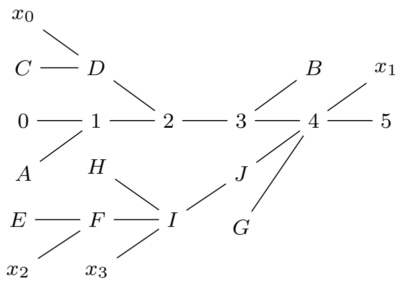


Fig. 1.11 The partial order constraints corresponding to the proof structure of Figure 1.10.

Here we have $J \leq 4$, but also $4 < x_1$. Unifying J to x_1 would therefore produce the contradicting $x_1 \leq 4$ and $4 < x_1$.

Many potential axioms connections are excluded by a simply failure of unification between the two atoms: the positive atom $s(x_2, x_1)$ cannot connect either to $s(A, 2)$ or to $s(E, 5)$ (since x_1 does not unify with either 2 or 5).

Finally, the contractability condition excludes many other connections. The metavariables F, I, and J have free occurrences at many nodes. This notably means none of them can unify with x_1, x_2 or x_3 without violating the contraction condition. Similarly, x_0 cannot unify with B, C, or D. In general, the eigenvariable of a universal link can never appear on the 'wrong' side of its link (the part to which the arrow points), since this would correspond to a violation of the eigenvariable condition in the sequent calculus.

Now, returning to our proof structure, we can see there is only a single possibility for the positive atomic formula $s(x_2, x_1)$. We have already seen that $s(A, 2)$ and $s(E, 5)$ do no unify and that $s(H, J)$ fails on the partial order constraint. This leaves only $s(C, B)$ and $s(G, x_1)$. However, $s(G, x_1)$

fails on the proof net condition: unifying G to x_2 produces an occurrence of x_2 on the $4, x_1$ node of the proof structure above the G link (since it is on the existential frontier of G). And a reduction of the par link requires an identification of this node with the F, I, J, x_1 node, thereby producing an occurrence of x_2 on the wrong side of its universal link. Therefore, the only possible connection for $s(x_2, x_1)$ is to $s(C, B)$, unifying $C = x_2$ and $B = x_1$. This fills in the first cell labeled 1 of Table 1.8. This unification then turns the positive $np(C, D)$ formula into $np(x_2, D)$ which can only unify with $np(x_2, F)$, filling cell 2 of the table.

We can now turn to the goal formula $s(0, 5)$. Since we have already connected the $s(C, B)$ formula to $s(x_2, x_1)$ this option is no longer available, and the $s(A, 2)$ and $s(G, x_1)$ options are excluded by failure of unification. Finally, $s(H, J)$ is excluded because the $J \leq 4$ partial order constraint would contradict unifying J to 5. This leaves only the $s(E, 5)$ possibility, unifying E to 0, as indicated by cell 3 of the table.

After these unifications the negative $np(E, F)$ has become $np(0, D)$ which only unifies with $np(0, 1)$, instantiating D to 1, and filling cell 4 of the table. We have now essentially solved the linking problem and the remaining s connections can only be made in a single way, filling cells 5 to 7 in the table. Following that, we can apply similar reasoning to the np connections and fill the remaining cells (cells 8 to 10).

What we have shown is that even a for a quite complex proof structure such as the one in Figure 1.10, the partial order constraints combined with the proof net conditions can allow us to produce the unique solution while avoiding all backtracking. Given the essentially non-deterministic natural of natural language parsing (sentences can have multiple readings and our parser should therefore produce as many proofs), we will in many cases be required to use some form of backtracking. But this examples gives an illustration of how powerful the combined constraints are.

1.6 The Empty String

Up until now, we have not explicitly allowed string segments to be empty. However, there are some well-know applications of empty string, notably the treatment of extraction in variants of the Lambek calculus. We can add a variant of extraction as a residuated pair as follows.

$$\begin{aligned} \|A \multimap C\|^{y,z} &= \forall x.[\|A\|^{x,x}] \multimap \|C\|^{y,z} \\ \|A \otimes B\|^{y,z} &= \forall x.[\|A\|^{x,x}] \otimes \|B\| \end{aligned}$$

Even though this works in many cases, there is a potential problem here: suppose the extracted element is a vp, that is the Lambek calculus formula $np\backslash s$, with the standard translation into first-order linear logic of

$$\forall x_0.[np(x_0,x_1) \multimap s(x_0,x_2)]$$

corresponding to a *vp* at positions x_1, x_2. When we plug this formula into the A argument of the implication selecting an empty argument, the result is the identification of x_1 and x_2, producing the formula

$$\forall x_1 \forall x_0.[np(x_0,x_1) \multimap s(x_0,x_1)]$$

for this extracted *vp*.

Compare this to an extracted formula corresponding to s/np. It would be translated into

$$\forall x_2.[np(x_1,x_2) \multimap s(x_0,x_2)]$$

at positions x_0, x_1. Turning this into the empty string identifies x_0 with x_1, producing the following

$$\forall x_1 \forall x_2.[np(x_1,x_2) \multimap s(x_1,x_2)]$$

The problem now is that this is equivalent to the formula for the extracted *vp* we computed before!

Though it would seem that there is not much of a difference between concatenating the empty string to the left or to the right of an *np* constituent, there should be a difference in behaviour between an $np\backslash s$ gap and a s/np gap: for example, the first, but not the second can be modified by an subject-oriented adverb of type $(np\backslash s)\backslash(np\backslash s)$. The naive first-order translation fails to make this distinction.

There is a solution, and it consists of moving the universal quantifier out. Instead of the universal quantifier having only the A formula as its scope, we turn it into an existential quantifier which has the entire $A \multimap C$ formula as its scope as follows.

$$\|A \multimap C\|^{y,z} = \exists x.[\|A\|^{x,x} \multimap \|C\|^{y,z}]$$

This allows us to correctly distinguish these two cases, but at the price of no longer having a residuated pair for the extraction phenomena[6].

1.7 Discussion

One obvious aspect of first-order linear logic which hasn't been mention thus far is that the Horn clause fragment corresponds to a lexicalised version of multiple context-free grammars (Moot 2014; Wijnholds 2011). Horn clauses

[6] This analysis also makes an unexpected empirical claim: the treatment of parasitic gapping in type-logical grammars using the linear logic exponential ! would require the exponential to have scope over the quantified variable representing the empty string. We therefore need to claim that parasitic gapping can only happen with atomic formulas.

for first-order linear logic are of the form $\forall x_0, \ldots, x_n[p_1 \otimes \ldots \otimes p_m \multimap q]$ for predicates p_i and q, or equivalently $\forall x_1, \ldots, x_n.(p_1 \multimap (\ldots \multimap (p_m \multimap q)$, and they code each segment of an MCFG by a pair of string positions. In the context of MCFG it is well-known that each additional segment increases the generative capacity. When the maximum arity is 2, each predicate has a single segment and we have context-free grammars allowing us to generate languages such as $a^n b^n$. When the maximum arity is 4, we can generate $a^n b^n c^n d^n$, with maximum arity 6 $a^n b^n c^n d^n e^n f^n$, and so on (Kallmeyer 2010).

It is unclear which of these classes best captures the properties we want with respect to the string languages needed for the analysis of natural languages. It is generally assumed that a reasonable minimum is 4 (that is, two string segments per predicate). For example the languages generated by tree adjoining grammars and several similar formalisms are strictly included in this class (more precisely, the tree adjoining languages have the additional contraint of well-nestedness, whereas the multiple context free languages in general do not (Seki et al. 1991)).

It is unclear to me which would be the right number of components to consider. Values between 4 and 6 components would seem to suffice for most applications, and it is unclear whether there are good linguistic reasons for abandoning well-nestnedness.

The well-nested, residuated connectives seem to be the same as those definable in the Displacement calculus. Indeed, I have elsewhere already implicitly assumed a linear order for all subproofs when relating the Displacment calculus to first-order linear logic (Moot 2014).

One interesting area of further investigation would be to relax the linear order constraint. For example, we let our sequent compute a unique partial order over the initial position variables (now no longer linearly ordered) and consider the sentence grammatical when the input string is a valid linearisation of this partial order. This would be potentially interesting for languages with relatively free word order.

1.8 Conclusions

This paper has discussed several aspect of adding partial order constraints to first-order linear logic. Although somewhat odd from the logical point of view, adding order constraints to the variables in first-order linear logic allows us to preserve the standard algebraic and category theoretic perspectives on type-logical grammars. In addition, some linguistically interesting operations can only be defined as part of a residuated triple when we impose partial order constraints on the string position variables.

We have also shown how partial order constraints can be use as a mechanism for improving proof search by filtering out choices inconsistent with this order.

References

Areces, Carlos, Raffaella Bernardi, and Michael Moortgat (2004). "Galois connections in categorial type logic". In: *Electronic Notes in Theoretical Computer Science* 53, pp. 3–20.

Bellin, Gianluigi and J. van de Wiele (1995). "Empires and Kingdoms in MLL". In: *Advances in Linear Logic*. Ed. by Jean-Yves Girard, Yves Lafont, and Laurent Regnier. Cambridge University Press, pp. 249–270.

Bernardi, Raffaella and Michael Moortgat (2010). "Continuation semantics for the Lambek–Grishin calculus". In: *Information and Computation* 208.5, pp. 397–416.

Coecke, Bob, Edward Grefenstette, and Mehrnoosh Sadrzadeh (2013). "Lambek vs. Lambek: Functorial vector space semantics and string diagrams for Lambek calculus". In: *Annals of pure and applied logic* 164.11, pp. 1079–1100.

Danos, Vincent (1990). "La Logique Linéaire Appliquée à l'étude de Divers Processus de Normalisation (Principalement du λ-Calcul)". PhD thesis. University of Paris VII.

Danos, Vincent and Laurent Regnier (1989). "The Structure of Multiplicatives". In: *Archive for Mathematical Logic* 28, pp. 181–203.

Došen, Kosta (1992). "A Brief Survey of Frames for the Lambek Calculus". In: *Zeitschrift für Mathematische Logic und Grundlagen der Mathematik* 38, pp. 179–187.

Girard, Jean-Yves (1991). "Quantifiers in Linear Logic II". In: *Nuovi problemi della logica e della filosofia della scienza*. Ed. by G. Corsi and G. Sambin. Vol. II. Proceedings of the conference with the same name, Viareggio, Italy, January 1990. Bologna, Italy: CLUEB.

– (2011). *The Blind Spot: Lectures on Logic*. European Mathematical Society.

Joshi, Aravind and Yves Schabes (1997). "Tree-adjoining Grammars". In: *Handbook of Formal Languages 3: Beyond Words*. Ed. by Grzegorz Rosenberg and Arto Salomaa. New York: Springer, pp. 69–123.

Kallmeyer, Laura (2010). *Parsing Beyond Context-Free Grammars*. Cognitive Technologies. Springer.

Kubota, Yusuke and Robert Levine (2012). "Gapping as Like-Category Coordination". In: *Logical Aspects of Computational Linguistics*. Ed. by Denis Béchet and Alexander Dikovsky. Vol. 7351. Lecture Notes in Computer Science. Nantes: Springer, pp. 135–150.

– (2020). *Type-Logical Syntax*. MIT Press.

Kurtonina, Natasha and Michael Moortgat (1997). "Structural Control". In: *Specifying Syntactic Structures*. Ed. by Patrick Blackburn and Maarten de Rijke. Stanford: CSLI, pp. 75–113.

Lambek, Joachim (1958). "The Mathematics of Sentence Structure". In: *American Mathematical Monthly* 65, pp. 154–170.

– (1988). “Categorial and Categorical Grammars”. In: *Categorial Grammars and Natural Language Structures*. Ed. by Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler. Vol. 32. Studies in Linguistics and Philosophy. Reidel, pp. 297–317.

Lincoln, Patrick (1995). “Deciding Provability of Linear Logic Formulas”. In: *Advances in Linear Logic*. Ed. by Jean-Yves Girard, Yves Lafont, and Laurent Regnier. Cambridge University Press, pp. 109–122.

Lincoln, Patrick and Natarajan Shankar (1994). “Proof Search in First-order Linear Logic and Other Cut-free Sequent Calculi”. In: *Proceedings of Logic in Computer Science (LICS'94)*. IEEE Computer Society Press, pp. 282–291.

Montague, Richard (1974). “The Proper Treatment of Quantification in Ordinary English”. In: *Formal Philosophy. Selected Papers of Richard Montague*. Ed. by R. Thomason. New Haven: Yale University Press.

Moortgat, Michael (1996). “Multimodal Linguistic Inference”. In: *Journal of Logic, Language and Information* 5.3–4, pp. 349–385.

Moot, Richard (2014). “Extended Lambek calculi and first-order linear logic”. In: *Categories and Types in Logic, Language, and Physics: Essays dedicated to Jim Lambek on the Occasion of this 90th Birthday*. Ed. by Claudia Casadio et al. Lecture Notes in Artificial Intelligence 8222. Springer, pp. 297–330.

Moot, Richard and Mario Piazza (2001). “Linguistic Applications of First Order Multiplicative Linear Logic”. In: *Journal of Logic, Language and Information* 10.2, pp. 211–232.

Morrill, Glyn, Oriol Valentin, and Mario Fadda (2011). “The Displacement Calculus”. In: *Journal of Logic, Language and Information* 20.1, pp. 1–48.

Oehrle, Richard T. (1994). “Term-Labeled Categorial Type Systems”. In: *Linguistics & Philosophy* 17.6, pp. 633–678.

– (2011). “Multi-modal type-logical grammar”. In: *Non-transformational Syntax: Formal and Explicit Models of Grammar*. Ed. by Robert Borsley and Kersti Börjars. Wiley-Blackwell. Chap. 6, pp. 225–267.

OEIS Foundation (1964). *On-Line Encyclopedia of Integer Sequences (OEIS)*. `http://oeis.org`. Accessed July 23 2020.

Seki, Hiroyuki et al. (1991). “On Multiple Context-free Grammars”. In: *Theoretical Computer Science* 88, pp. 191–229.

Wijnholds, Gijs (2011). “Investigations into Categorial Grammar: Symmetric Pregroup Grammar and Displacement Calculus”. MA thesis. Utrecht University.