Powerful and NP-Complete: Hypergraph Lambek Grammars

Tikhon Pshenitsyn[0000−0003−4779−3143][⋆]
ptihon@yandex.ru

Department of Mathematical Logic and Theory of Algorithms
Faculty of Mathematics and Mechanics
Lomonosov Moscow State University
GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation

Abstract. We consider two approaches to generating formal string languages: context-free grammars and Lambek grammars, which are based on the Lambek calculus. They are equivalent in the sense that they generate the same set of languages (disregarding the empty word). It is well known that context-free grammars can be generalized to hyperedge replacement grammars (HRGs) preserving their main principles and properties. In this paper, we study a generalization of the Lambek grammars to hypergraphs and investigate the recognizing power of the new formalism. We show how to define the hypergraph Lambek calculus (HL), and then introduce *hypergraph Lambek grammars* based on HL. It turns out that such grammars recognize all isolated-node bounded languages generated by HRGs. However, they are more powerful than HRGs: they recognize at least finite intersections of such languages. Thus the Pentus theorem along with the pumping lemma and the Parikh theorem have no place for hypergraph Lambek grammars. Besides, it can be shown that hypergraph Lambek grammars are NP-complete, so they constitute an attractive alternative to HRGs, which are also NP-complete.

1 Introduction

Formal grammars include two large classes opposed to each other: context-free grammars and categorial grammars. The former generate languages by means of productions: one starts with a single symbol and then applies productions to it, hence generating a string. On contrary, categorial grammars work in a "deductive way": they start with a whole string, assign types to strings and then check whether the resulting sequence of types is correct w.r.t. some uniform laws. There are many options of choosing such uniform laws, which lead to different formalisms. The laws are often provided by a logical calculus; in such cases, grammars are called type-logical.

⋆ The study was funded by RFBR, project number 20-01-00670 and by the Interdisciplinary Scientific and Educational School of Moscow University "Brain, Cognitive Systems, Artificial Intelligence". The author is a Scholarship holder of "BASIS" Foundation. The work was presented at ICGT 2021, and it was awarded by EATCS.

One of important classes of type-logical grammars is Lambek grammars. They are based on the Lambek calculus introduced in [4] (further we denote it as L). This calculus deals with types and sequents. Types are built from primitive types $Pr = \{p_1, p_2, \dots\}$ (which we further denote by small Latin letters $p, q, \dots$) using operations $\backslash$, $\cdot$, $/$; for example, $(p \cdot q)/(p\backslash q)$ is a type. A sequent is a structure of the form $A_1, \dots, A_n \to A$ where $n > 0$ and A_i, A are types. The Lambek calculus in the Gentzen style we consider in this paper explains how to derive sequents. There is one axiom and six inference rules of L:

$$\frac{}{A \to A}\ (\mathrm{Ax})$$

$$\frac{\Pi \to A \quad \Gamma, B, \Delta \to C}{\Gamma, \Pi, A\backslash B, \Delta \to C}\ (\backslash \to) \qquad \frac{A, \Pi \to B}{\Pi \to A\backslash B}\ (\to \backslash) \qquad \frac{\Gamma, A, B, \Delta \to C}{\Gamma, A \cdot B, \Delta \to C}\ (\cdot \to)$$

$$\frac{\Pi \to A \quad \Gamma, B, \Delta \to C}{\Gamma, B/A, \Pi, \Delta \to C}\ (/ \to) \qquad \frac{\Pi, A \to B}{\Pi \to B/A}\ (\to /) \qquad \frac{\Pi \to A \quad \Psi \to B}{\Pi, \Psi \to A \cdot B}\ (\to \cdot)$$

Here capital Latin letters denote types, and capital Greek letters denote sequences of types (besides, Π, Ψ are nonempty).

Example 1.1. Below an example of a derivation is presented:

$$\dfrac{\dfrac{\dfrac{s \to s \quad q \to q}{s/q, q \to p}\ (/ \to) \qquad s \to s}{s/q, s, s\backslash q \to s}\ (\backslash \to) \qquad q \to q}{s/q, s/q, q, s\backslash q \to s}\ (/ \to)$$

The Lambek calculus has algebraic and logical nature (e.g. it can be considered as a substructural logic of intuitionistic logic). Nicely, it can be used to describe formal and natural languages, which can be done using so-called Lambek grammars. A Lambek grammar consists of an alphabet Σ, of a finite binary relation $\triangleright$ between symbols of Σ and types of the Lambek calculus and of a distinguished type S. The grammar recognizes (synonym for *generates* regarding categorial grammars) the language of all strings $a_1 \dots a_n$, for which there exist types $T_1, \dots, T_n$ such that $a_i \triangleright T_i$, and $T_1, \dots, T_n \to S$ is derivable in L.

Example 1.2. The Lambek grammar over the alphabet $\{a, b\}$ such that $S = s \in Pr$ and $a \triangleright s/q$, $b \triangleright s\backslash q$, $b \triangleright q$ generates the language $\{a^n b^n \mid n > 0\}$. E.g., given the string $aabb$, one transforms it into a sequent $s/q, s/q, q, s\backslash q \to s$ and derives it in L as is shown in Example 1.1.

From 1960-s until nowadays different extensions of L were proposed to capture linguistic phenomena of interest (e.g. parasitic gaps, wh-movements etc.; see [5]). Many mathematical properties of L and of Lambek grammars have been discovered. The one, which is of interest in this work, is the Pentus theorem (see [7]). It states that Lambek grammars recognize exactly the class of context-free languages (without the empty word). Therefore, in the string case Lambek grammars and context-free grammars are equivalent.

Moving away from categorial grammars, we consider another approach: *hyperedge replacement grammar* (HRG in short). These grammars were developed in order to generalize context-free grammars to hypergraphs. Namely, they generate hypergraphs by means of productions: a production allows one to replace a hyperedge of a hypergraph with another hypergraph. Hyperedge replacement grammars (HRGs) have a number of properties in common with context-free grammars such as the pumping lemma, the Parikh theorem, the Greibach normal form etc. An overview of HRGs can be found in [2]. In this work, we follow definitions and notation from this handbook chapter as far as possible.

Comparing two fields of research, we came up with a question: is it possible to generalize type-logical formalisms to hypergraphs in a natural way? A desired extension should satisfy two properties: there must be an embedding of string type-logical grammars in it, and it should be connected to hyperedge replacement grammars, as type-logical grammars are connected to context-free grammars. Moreover, one expects that the Lambek calculus underlying Lambek grammars can also be generalized to hypergraphs resulting in some kind of a graph logic.

Our first attempt was concerned with generalizing basic categorial grammars to hypergraphs. The result called hypergraph basic categorial grammars is introduced in [8]. The idea of such grammars is just inverting a derivation of HRGs and remodeling it from bottom to top. This idea is not new; it is, e.g. closely connected to the concept of abstract categorial grammars (see [3]). In [8], such inverting is done straightforwardly: there are types, which are complex terms made of hypergraphs; using them one labels hyperedges and then checks whether a graph labeled by types can be reduced to a single-edge hypergraph using a uniform reduction law.

In our preprint [9], we do the next step and introduce a generalization of the Lambek calculus to hypergraphs called the hypergraph Lambek calculus (HL in short). This calculus naturally generalizes L to hypergraphs, and, moreover, contains its different variants as fragments. On the other hand, the inference rules of this calculus are defined through hyperedge replacement, and thus HL is related to HRGs. In this paper, we introduce the definition of HL in Section 3, establish an embedding of L in HL, and then turn to defining and studying a grammar formalism that can be built on the basis of HL. Such formalism is called *hypergraph Lambek grammar* (HL-grammar for short); it is introduced in Section 4. In Section 5 we establish three main results: HL-grammars are not weaker than HRGs; the parsing problem for HL-grammars is NP-complete; HL-grammars can recognize finite intersections of hypergraph context-free languages. These three facts enable one to conclude that HL-grammar is an attractive alternative to HRG: both formalisms are NP-complete while HL-grammars are more powerful.

2 Preliminaries

$\mathbb{N}$ includes 0. Σ^* is the set of all strings over the alphabet Σ. $\Sigma^{\circledast}$ is the set of all strings consisting of distinct symbols. The length $|w|$ of the word w is the number of symbols in w. The set of all symbols contained in a word w is denoted by $[w]$. If $f : \Sigma \to \Delta$ is a function from one set to another, then it is naturally extended to a function $f : \Sigma^* \to \Delta^*$ ($f(\sigma_1 \dots \sigma_k) = f(\sigma_1) \dots f(\sigma_k)$).

Let C be some fixed set of labels for whom the function $type : C \to \mathbb{N}$ is considered (C is called a ranked alphabet).

Definition 2.1. A hypergraph G over C *is a tuple* $G = \langle V, E, att, lab, ext\rangle$ *where* V *is a set of nodes,* E *is a set of hyperedges,* $att : E \to V^{\circledast}$ *assigns a string (i.e. an ordered set) of attachment nodes to each hyperedge,* $lab : E \to C$ *labels each hyperedge by some element of* C *in such a way that* $type(lab(e)) = |att(e)|$ *whenever* $e \in E$*, and* $ext \in V^{\circledast}$ *is a string of external nodes.*

Components of a hypergraph G *are denoted by* $V_G, E_G, att_G, lab_G, ext_G$ *resp.*

In the remainder of the paper, hypergraphs are usually called just graphs, and hyperedges are called edges. The set of all graphs with labels from C is denoted by $\mathcal{H}(C)$. Graphs are usually named by letters G and H.

In drawings of graphs, black dots correspond to nodes, labeled squares correspond to edges, *att* is represented by numbered lines, and external nodes are depicted by numbers in brackets. If an edge has exactly two attachment nodes, it can be depicted by an arrow (which goes from the first attachment node to the second one).

Note that Definition 2.1 implies that attachment nodes of each hyperedge are distinct, and so are external nodes. This restriction can be removed (i.e. we can consider graphs with loops), and all further definitions will be preserved; however, in this paper, we stick to the above definition.

Definition 2.2. *The function* $type_G$ *(or type, if* G *is clear) returns the number of nodes attached to an edge in a graph* G*:* $type_G(e) := |att_G(e)|$*. If* G *is a graph, then* $type(G) := |ext_G|$.

Definition 2.3. *A sub-hypergraph (or just subgraph)* H *of a graph* G *is a hypergraph such that* $V_H \subseteq V_G$*,* $E_H \subseteq E_G$*, and for all* $e \in E_H$ $att_H(e) = att_G(e)$*,* $lab_H(e) = lab_G(e)$*.*

Definition 2.4. *If* $H = \langle \{v_i\}_{i=1}^n, \{e_0\}, att, lab, v_1 \dots v_n\rangle$*,* $att(e_0) = v_1 \dots v_n$ *and* $lab(e_0) = a$*, then* H *is called* a handle*. It is denoted by* $a^\bullet$*.*

Definition 2.5. An isomorphism *between graphs* G *and* H *is a pair of bijective functions* $\mathcal{E} : E_G \to E_H$*,* $\mathcal{V} : V_G \to V_H$ *such that* $att_H \circ \mathcal{E} = \mathcal{V} \circ att_G$*,* $lab_G = lab_H \circ \mathcal{E}$*,* $\mathcal{V}(ext_G) = ext_H$*. In this work, we do not distinguish between isomorphic graphs.*

Strings can be considered as graphs with the string structure. This is formalized in

Definition 2.6. A string graph *induced by a string* $w = a_1 \dots a_n$ *is a graph of the form* $\langle \{v_i\}_{i=0}^n, \{e_i\}_{i=1}^n, att, lab, v_0 v_n\rangle$ *where* $att(e_i) = v_{i-1} v_i$*,* $lab(e_i) = a_i$*. In this work, we denote it by* $\mathrm{SG}(w)$*.*

We additionally introduce the following definition (not from [2]):

Definition 2.7. *Let $H \in \mathcal{H}(C)$ be a graph, and let $f : E_H \to C$ be a function. Then $f(H) := \langle V_H, E_H, att_H, lab_{f(H)}, ext_H \rangle$ where $lab_{f(H)}(e) = f(e)$ for all e in E_H. It is required that $type(lab_H(e)) = type(f(e))$ for $e \in E_H$.*

If one wants to relabel only one edge e_0 within H with a label a, then the result is denoted by $H[e_0 := a]$.

2.1 Hyperedge Replacement Grammars

Hyperedge replacement grammars (HRGs in short) are based on the procedure of hyperedge replacement. The replacement of an edge e_0 in G with a graph H can be done if $type(e_0) = type(H)$ as follows:

1. Remove e_0;
2. Insert an isomorphic copy of H (H and G have to consist of disjoint sets of nodes and edges);
3. For each i, fuse the i-th external node of H with the i-th attachement node of e_0.

The result is denoted by $G[e_0/H]$. It is known that if several edges of a graph are replaced by other graphs, then the result does not depend on the order of replacements; moreover the result is not changed, if replacements are done simultaneously (see [2]). The following notation is in use: if $e_1, \dots, e_k$ are distinct edges of a graph H and they are simultaneously replaced by graphs $H_1, \dots, H_k$ resp. (this requires $type(H_i) = type(e_i)$), then the result is denoted $H[e_1/H_1, \dots, e_k/H_k]$.

A *hyperedge replacement grammar (HRG)* is a tuple $HGr = \langle N, \Sigma, P, S \rangle$, where N and Σ are disjoint finite ranked alphabets (of nonterminal and terminal symbols resp.), P is a set of productions, and $S \in N$. Each production is of the form $A \to H$ where $A \in N$, $H \in \mathcal{H}(N \cup \Sigma)$ and $type(A) = type(H)$. A grammar generates the language of all terminal graphs (i.e. graphs with terminal labels only) that can be obtained from $S^\bullet$ by applying productions from P. Such a language is called a hypergraph context-free language (HCFL in short). Two grammars are said to be equivalent if they generate the same language.

3 Hypergraph Lambek Calculus

In this section, the hypergraph Lambek calculus (HL) is introduced. As in the string case, we are going to define types, sequents, an axiom and rules of the calculus; now, however, they are expected to be based on hypergraphs. The intuition of the definitions we aim to introduce comes from the procedure of convertion of a context-free grammar into an equivalent Lambek grammar in the string case. This procedure can be illustrated by the following

Example 3.1. Consider a context-free grammar $Gr = \langle \{S, Q\}, \{a, b\}, P, S \rangle$ with the set P including three productions: $S \to aQ$, $Q \to Sb$, $Q \to b$. This grammar generates the language $\{a^n b^n \mid n > 0\}$. E.g. $S \Rightarrow aQ \Rightarrow aSb \Rightarrow aaQb \Rightarrow aabb$ shows that $aabb$ is generated by Gr. Note that Gr is lexicalized; that is, there is exactly one terminal symbol in the right-hand side of each production. This enables one to convert Gr into an equivalent Lambek grammar as follows:

$$\begin{array}{lcl} S \to aQ & \rightsquigarrow & a \triangleright s/q \\ Q \to Sb & \rightsquigarrow & b \triangleright s\backslash q \\ Q \to b & \rightsquigarrow & b \triangleright q \end{array}$$

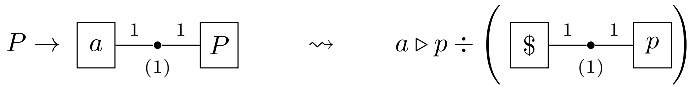
This procedure is quite intuitive. For example, the production $S \to aQ$ can be read as "a structure of the type S can be obtained, if one concatenates a symbol a and a structure of the type Q (in such order)". In comparison, $a \triangleright s/q$ can be understood as the statement "a is such a symbol that, whenever a structure of the type q appears to its right, they together form a structure of the type s". Clearly, the above two statements are equivalent. In general, a correspondence $a \triangleright A/B$ in a Lambek grammar can be informally understood as follows: a is such a symbol that it waits for a structure (a string) of the type B from the right in order to form a string of the type A together with it. Similarly, one can describe the meaning of a correspondence $a \triangleright B\backslash A$ with the only difference that a symbol requires a structure of the type B from the left.

The lexicalized normal form (or the weak Greibach normal form) for context-free grammars can be generalized to HRGs, which is studied in [10].

Definition 3.1. *An HRG HGr is in* the weak Greibach normal form *if there is exactly one terminal label in the right-hand side of each production.*

Let $A \to H$ be a production in such a grammar; that is, H is a graph with exactly one terminal label. In order to perform a convertion similar to the one explained above for context-free grammars, one would like to "extract" the only terminal label from H and to associate this label with a type, which would look like A/H. In H, however, we should mark the place, from which we take the only terminal label; let us do this using a special \$ label (which would be allowed to label edges of different types[1]). Besides, in order to distinguish between string divisions and a new hypergraph division we shall write $\div$ instead of / or \\.

Example 3.2. Below we present a production of some HRG and the result of its convertion (the result has no formal sense yet, it is just a game with symbols):



Let us investigate how \\ and / of the Lambek calculus correlate with $\div$. It is known that context-free grammars can be embedded in HRGs using string graphs. For example, the grammar Gr from Example 3.1 can be transformed into an HRG with the following productions: $S \to \mathrm{SG}(aQ)$, $Q \to \mathrm{SG}(Sb)$, $Q \to \mathrm{SG}(b)$. Now let us apply the above convertion to the first and the second productions:

[1] To be consistent with the definition of a hypergraph one may assume that there are different symbols $\$_n, n \geq 0$ instead such that $type(\$_n) = n$.

$$\begin{tikzcd} S \to \ (1)\ \bullet \arrow[r, "a"] & \bullet \arrow[r, "Q"] & \bullet\ (2) \quad \rightsquigarrow \quad a \triangleright s \div \Big((1)\ \bullet \arrow[r, "\$"] & \bullet \arrow[r, "q"] & \bullet\ (2) \Big) \end{tikzcd}$$

$$\begin{tikzcd} Q \to \ (1)\ \bullet \arrow[r, "S"] & \bullet \arrow[r, "b"] & \bullet\ (2) \quad \rightsquigarrow \quad b \triangleright q \div \Big((1)\ \bullet \arrow[r, "s"] & \bullet \arrow[r, "\$"] & \bullet\ (2) \Big) \end{tikzcd}$$

The resulting types in the right-hand side may be considered as graph counterparts of types s/q and $s\backslash q$ resp. Note that the difference between the left and the right divisions is represented in these types by the position of the \$ symbol.

For now, this convertion is only juggling symbols, and $\div$ does not have a functional definition yet. It will be presented later. Now, let us discuss the issue of generalizing the multiplication operation. In the string case, $A \cdot B$ can be informally understood as the set of all strings of the form uv where u is of the type A, and v is of the type B; that is, $A \cdot B$ corresponds to the concatenation operation. In the hypergraph case, one needs a "generalized concatenation". Implementing this idea we introduce a unary operation $\times$. If M is a hypergraph labeled by types, then $\times(M)$ represents all substitution instances of M, that is, all hypergraphs that are obtained from M by replacing each edge labeled by some type by a hypergraph of this type. The connection between $\times$ and $\cdot$ is shown in Section 3.3.

3.1 Formal Definition of Types and Sequents

In this section, we give formal meanings to the operations introduced above. Let us fix a countable set Pr of primitive types and a function $type : Pr \to \mathbb{N}$ such that for each $n \in \mathbb{N}$ there are infinitely many $p \in Pr$ for which $type(p) = n$. Types are constructed from primitive types using division $\div$ and multiplication $\times$ operations. Simultaneously, the function *type* is defined on types: this is obligatory since we are going to label edges by types.

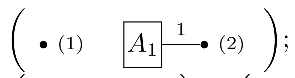
Definition 3.2. *The set $Tp(\mathrm{HL})$ of types is defined inductively as follows:*

1. *$Pr \subseteq Tp(\mathrm{HL})$.*
2. *Let N ("numerator") be in $Tp(\mathrm{HL})$. Let D ("denominator") be a graph such that exactly one of its edges (call it e_0) is labeled by \$, and the other edges (possibly, there are none of them) are labeled by elements of $Tp(\mathrm{HL})$; let also $type(N) = type(D)$. Then $T = (N \div D)$ also belongs to $Tp(\mathrm{HL})$, and $type(T) := type_D(e_0)$.*
3. *Let M be a graph such that all its edges are labeled by types from $Tp(\mathrm{HL})$ (possibly, there are no edges at all). Then $T = \times(M)$ belongs to $Tp(\mathrm{HL})$, and $type(T) := type(M)$.*

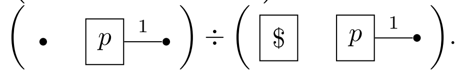
In types with division, D is usually drawn as a graph in brackets, so instead of $(N \div D)$ (a formal notation) a graphical notation $N \div (D)$ is in use. Sometimes brackets are omitted.

Example 3.3. The following structures are types:

– $$\begin{tikzcd} A_1 = p \div \Big(\boxed{\$} \arrow[r, dash, "1"] & \bullet\ (1) \quad \boxed{p} \arrow[r, dash, "1"] & \bullet \Big); \end{tikzcd}$$

– $A_2 = \times$ 

;

– $A_3 = \times$ 

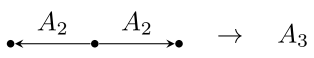
.

Here $type(p) = 1$, $type(A_1) = 1$, $type(A_2) = 2$, $type(A_3) = 0$.

Definition 3.3. A graph sequent *is a structure of the form* $H \to A$ *where* $A \in Tp(\mathrm{HL})$ *is a type,* $H \in \mathcal{H}(Tp(\mathrm{HL}))$ *is a graph labeled by types and* $type(H) = type(A)$*.* H *is called the antecedent of the sequent, and* A *is called the succedent of the sequent.*

Let $\mathcal{T}$ be a subset of $Tp(\mathrm{HL})$. We say that $H \to A$ is over $\mathcal{T}$ if $G \in \mathcal{H}(\mathcal{T})$ and $A \in \mathcal{T}$.

Example 3.4. The following structure is a graph sequent (where A_2, A_3 are from Example 3.3):



3.2 Axiom and Rules

The hypergraph Lambek calculus (denoted HL) we introduce here is a logical system that defines what graph sequents are derivable (=provable). HL includes one axiom and four rules, which are introduced below. Each rule is illustrated by an example exploiting string graphs.

The only axiom is the following: $A^\bullet \to A, \quad A \in Tp(\mathrm{HL})$.

Rule $(\div\to)$**.** Let $N \div D$ be a type and let $E_D = \{d_0, d_1, \dots, d_k\}$ where $lab(d_0) = \$$. Let $H \to A$ be a graph sequent and let $e \in E_H$ be labeled by N. Let finally $H_1, \dots, H_k$ be graphs labeled by types. Then the rule $(\div\to)$ is the following:

$$\frac{H \to A \quad H_1 \to lab(d_1) \quad \dots \quad H_k \to lab(d_k)}{H[e/D][d_0 := N \div D][d_1/H_1, \dots, d_k/H_k] \to A}\ (\div\to)$$

This rule explains how a type with division appears in an antecedent: we replace an edge e by D, put a label $N \div D$ instead of \$ and replace the remaining labels of D by corresponding antecedents.

Example 3.5. Consider the following rule application with T_i being some types and with T being equal to $q \div \mathrm{SG}(T_2\$T_3)$:

$$\frac{\mathrm{SG}(pq) \to T_1 \quad \mathrm{SG}(rs) \to T_2 \quad \mathrm{SG}(tu) \to T_3}{\mathrm{SG}(prs\,T\,tu) \to T_1}\ (\div\to)$$

Rule $(\to\div)$**.** Let $F \to N \div D$ be a graph sequent; let $e_0 \in E_D$ be labeled by \$. Then

$$\frac{D[e_0/F] \to N}{F \to N \div D}\ (\to\div)$$

Formally speaking, this rule is improper since it is formulated from bottom to top. It is understood, however, as follows: if there are such graphs D, F and such a type N that in a sequent $H \to N$ the graph H equals $D[F/e_0]$, and $H \to N$ is derivable, then $F \to N \div D$ is also derivable.

Example 3.6. Consider the following rule application where T is some type (here we draw graphs instead of writing $\mathrm{SG}(w)$ to visualize the rule application):

$$\frac{(1)\,\bullet\xrightarrow{p}\bullet\xrightarrow{q}\bullet\xrightarrow{r}\bullet\,(2) \;\to\; T}{(1)\,\bullet\xrightarrow{p}\bullet\xrightarrow{q}\bullet\,(2) \;\to\; T \div \left((1)\,\bullet\xrightarrow{\$}\bullet\xrightarrow{r}\bullet\,(2)\right)}\;(\to\div)$$

Rule $(\times\to)$**.** Let $G \to A$ be a graph sequent and let $e \in E_G$ be labeled by $\times(F)$. Then

$$\frac{G[e/F] \to A}{G \to A}\;(\times\to)$$

This rule again is formulated from bottom to top. Informally speaking, there is a subgraph of an antecedent in the premise, and it is “compressed” into a single $\times(F)$-labeled edge.

Example 3.7. Consider the following rule application where U is some type:

$$\frac{(1)\,\bullet\xrightarrow{p}\bullet\xrightarrow{q}\bullet\xrightarrow{r}\bullet\xrightarrow{s}\bullet\,(2) \;\to U}{(1)\,\bullet\xrightarrow{p}\bullet\xrightarrow{\times(\mathrm{SG}(qr))}\bullet\xrightarrow{s}\bullet\,(2) \;\to U}\;(\times\to)$$

Rule $(\to\times)$**.** Let $\times(M)$ be a type and let $E_M = \{m_1, \dots, m_l\}$. Let $H_1, \dots,$ H_l be graphs over $Tp(\mathrm{HL})$. Then

$$\frac{H_1 \to lab(m_1) \quad \dots \quad H_l \to lab(m_l)}{M[m_1/H_1, \dots, m_l/H_l] \to \times(M)}\;(\to\times)$$

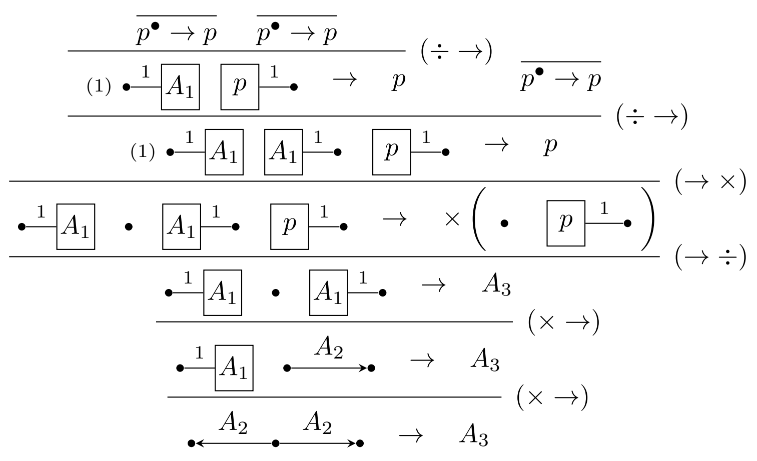
This rule is quite intuitive: several sequents can be combined into a single one via some graph structure M.

Example 3.8. Consider the following rule application with T_i being some types:

$$\frac{\mathrm{SG}(pq) \to T_1 \quad \mathrm{SG}(rs) \to T_2 \quad \mathrm{SG}(tu) \to T_3}{\mathrm{SG}(pqrstu) \to \times(\mathrm{SG}(T_1T_2T_3))}\;(\to\times)$$

Definition 3.4. *A graph sequent $H \to A$ is* derivable in HL *(*$\mathrm{HL} \vdash H \to A$*) if it can be obtained from axioms using rules of* HL*. A corresponding sequence of rule applications is called* a derivation *and its representation as a tree is called* a derivation tree*.*

Example 3.9. The sequent from Example 3.4 is derivable in HL. Here is its derivation tree:

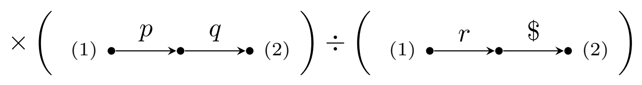


3.3 Embedding of the Lambek Calculus in HL

As expected, the Lambek calculus can be embedded in HL using string graphs. Consider the following embedding function $tr : Tp(\mathrm{L}) \to Tp(\mathrm{HL})$:

- $tr(p) := p, \quad p \in Pr, type(p) = 2$;
- $tr(A/B) := tr(A) \div \mathrm{SG}(\$\, tr(B))$;
- $tr(B\backslash A) := tr(A) \div \mathrm{SG}(tr(B)\, \$)$;
- $tr(A \cdot B) := \times(\mathrm{SG}(tr(A)\, tr(B)))$.

Example 3.10. The type $r\backslash(p \cdot q)$ is translated into the type



A string sequent $\Gamma \to A$ is transformed into a graph sequent as follows: $tr(\Gamma \to A) := \mathrm{SG}(tr(\Gamma)) \to tr(A)$. Let $tr(Tp(\mathrm{L}))$ be the image of tr.

Theorem 3.1.

1. *If* $\mathrm{L} \vdash \Gamma \to C$, *then* $\mathrm{HL} \vdash tr(\Gamma \to C)$.
2. *If* $\mathrm{HL} \vdash G \to T$ *is a derivable graph sequent over* $tr(Tp(\mathrm{L}))$, *then for some* Γ *and* C *we have* $G \to T = tr(\Gamma \to C)$ *(in particular,* G *has to be a string graph) and* $\mathrm{L} \vdash \Gamma \to C$.

The proof of this theorem is straightforward; it can be found in [9] along with the further discussion of HL.

4 Hypergraph Lambek Grammars

Now, we are finally ready to introduce a grammar formalism based on the hypergraph Lambek calculus.

Definition 4.1. *A* hypergraph Lambek grammar (HL-grammar) *is a tuple* $HGr = \langle \Sigma, S, \triangleright \rangle$ *where* Σ *is a finite ranked alphabet,* $S \in Tp(\mathrm{HL})$ *is a distinguished type, and* $\triangleright \subseteq \Sigma \times Tp(\mathrm{HL})$ *is a finite binary relation such that* $a \triangleright T$ *implies* $type(a) = type(T)$*.*

We call the set $dict(HGr) = \{T \in Tp(\mathrm{HL}) : \exists a : a \triangleright T\}$ a dictionary of HGr.

Definition 4.2. The language $L(HGr)$ recognized by a hypergraph Lambek grammar $HGr = \langle \Sigma, S, \triangleright \rangle$ *is the set of all hypergraphs* $G \in \mathcal{H}(\Sigma)$*, for which a function* $f_G : E_G \to Tp(\mathrm{HL})$ *exists such that:*

1. $lab_G(e) \triangleright f_G(e)$ *whenever* $e \in E_G$*;*
2. $\mathrm{HL} \vdash f_G(G) \to S$*.*

Example 4.1. Consider the HL-grammar $EGr_1 := \langle \{a, b\}, s, \triangleright_1 \rangle$ where $s, q \in Pr$, $type(s) = type(q) = 2$ and $\triangleright_1$ is defined below:

– $a \triangleright_1 s \div \mathrm{SG}(\$q)$; – $b \triangleright_1 q \div \mathrm{SG}(s\$)$; – $b \triangleright_1 q$;

This grammar recognizes the language of string graphs $\{\mathrm{SG}(a^n b^n) \mid n > 0\}$. E.g. given the string graph $\mathrm{SG}(aabb)$, one relabels its edges by corresponding types and obtains the graph $\mathrm{SG}\left[(s \div \mathrm{SG}(\$q))\ (s \div \mathrm{SG}(\$q))\ q\ (q \div \mathrm{SG}(s\$))\right]$. Now it remains to derive the sequent $\mathrm{SG}\left[(s \div \mathrm{SG}(\$q))\ (s \div \mathrm{SG}(\$q))\ q\ (q \div \mathrm{SG}(s\$))\right] \to s$:

$$\dfrac{\dfrac{\dfrac{\mathrm{SG}(s) \to s \quad \mathrm{SG}(q) \to q}{\mathrm{SG}\left[(s \div \mathrm{SG}(\$q))\ q\right]^{\bullet} \to s}(\div \to) \quad \mathrm{SG}(s) \to s}{\mathrm{SG}\left[(s \div \mathrm{SG}(\$q))\ s\ (q \div \mathrm{SG}(s\$))\right] \to s}(\div \to) \quad \mathrm{SG}(q) \to q}{\mathrm{SG}\left[(s \div \mathrm{SG}(\$q))\ (s \div \mathrm{SG}(\$q))\ q\ (q \div \mathrm{SG}(s\$))\right] \to s}(\div \to)$$

It is not hard to notice that this derivation resembles that from Example 1.1. The above derivation illustrates Theorem 3.1.

Example 4.2. Consider an HL-grammar $EGr_2 = \langle \{a\}, A_3, \triangleright_2 \rangle$ where $a \triangleright_2 A_2$ (A_2, A_3 are from Example 3.3). Then the graph $\begin{tikzcd}\bullet & \bullet \arrow[l, "a"'] \arrow[r, "a"] & \bullet\end{tikzcd}$ belongs to $L(HGr)$. In order to show this, we need to relabel all edges of this graph by types corresponding to current labels via $\triangleright_2$ and then to check derivability of a resulting sequent. In this example, we can only relabel a by A_2. Now it suffices to check derivability of a sequent

$$\begin{tikzcd}\bullet & \bullet \arrow[l, "A_2"'] \arrow[r, "A_2"] & \bullet & \to & A_3\end{tikzcd}$$

which is derivable according to Example 3.9.

Note that, if the graph in the antecedent contained not 2, but arbitrary number (say, n) of A_2-labeled edges outgoing from a single node, then the sequent would be derivable as well: its derivation (from bottom to top) would consist of n applications of $(\times \to)$, of one application of $(\to \div)$ and of $(\to \times)$, and finally of n applications of $(\div \to)$. It can be proved that no more graph sequents with A_2 in the antecedent and A_3 in the succedent are derivable, so HGr recognizes the language of stars.

The definitions and principles of HL-grammars may look sophisticated; however, their main idea is the same as for Lambek grammars: instead of generating graphs by means of productions, an HL-grammar takes the whole graph at first, then tries to relabel its edges via a correspondence $\triangleright$ and to derive the resulting sequent (with the succedent S) in HL.

Hypergraph Lambek grammars recognize hypergraph languages; thus they represent an alternative tool to HRGs. As in the string case, the major problem is describing the class of languages recognized by HL-grammars and comparing it with the class of languages generated by HRGs. In the string case the following theorem holds:

Theorem 4.1. *The class of languages recognized by Lambek grammars coincides with the class of context-free languages without the empty word.*

This theorem has two directions. The first one (CFGs $\rightsquigarrow$ LGs) was proved by Gaifman in 1960 [1] while the other one (LGs $\rightsquigarrow$ CFGs) was proved by Pentus in 1993 [7]. The first part is more simple; its proof is based on the Greibach normal form for context-free grammars (see Example 3.1). The second part appeared to be a hard problem; Pentus proved it using delicate logical and algebraic techniques including the free group interpretation and interpolants.

Summing up, in the string case context-free grammar and Lambek grammar approaches are equivalent. Regarding the graph case our first expectation was that similar things happen: HRGs and HL-grammars are equivalent disregarding, possibly, some nonsubstantive cases. As in the string case, we can introduce the analogue of the Greibach normal form for HRGs and study how to convert HRGs in this normal form into HL-grammars. However, this was not clear at all whether it is possible to perform the convertion of HL-grammars into equivalent HRGs: the proof of Pentus exploits free group interpretation, which is hard to generalize to graphs (we have no idea how to do this). Surprisingly, this convertion cannot be done at all! We figured out that hypergraph Lambek grammars recognize a wider class of languages than hypergraph context-free languages. Moreover, even the pumping lemma and the Parikh theorem do not hold for HL-grammars.

4.1 Properties of HL-grammars

Let us formulate several formal properties of HL and of HL-grammars that will help us later to prove the main theorem.

Theorem 4.2 (cut elimination). *If graph sequents $H \to A$ and $G \to B$ are derivable, and $e_0 \in E_G$ is labeled by A, then $G[e_0/H] \to B$ is also derivable.*

The proof of this theorem can be found in [9]. This theorem directly implies

Proposition 4.1 (reversibility of $(\times \to)$ and $(\to \div)$).

1. *If $\mathrm{HL} \vdash H \to C$ and $e_0 \in E_H$ is labeled by $\times(M)$, then $\mathrm{HL} \vdash H[e_0/M] \to C$.*
2. *If $\mathrm{HL} \vdash H \to N \div D$ and $e_0 \in E_D$ is labeled by \$, then $\mathrm{HL} \vdash D[e_0/H] \to N$.*

Definition 4.3. *A type A is called simple if one of the following holds:*

– *A is primitive;*
– *$A = \times(M)$, $E_M = \{m_1, \dots, m_l\}$ and $lab(m_1), \dots, lab(m_l)$ are simple;*
– *$A = N \div D$, $E_D = \{d_0, \dots, d_k\}$, $lab(d_0) = \$$, N is simple, and $lab(d_1)$, ..., $lab(d_k)$ are* primitive.

The next result is proved straightforwardly, but it is very useful in investigating what can be recognized by an HL-grammar.

Theorem 4.3. *Let* HL $\vdash H \to P$ *where H is labeled by simple types and P is either primitive or is of the form $\times(K)$ where all edge labels in K are primitive. Then there exists a* simple derivation *of $H \to P$, i.e. such a derivation that*

1. *The rule $(\to \times)$ either does not appear or is applied once to one of the leaves of the derivation tree.*
2. *In each application of $(\div \to)$ all the premises, except for, possibly, the first one, are of the form $q^\bullet \to q$, $q \in Pr$.*
3. *If a sequent $H' \to p$ within the derivation tree contains a type of the form $\times(M)$ in the antecedent, then the rule, after which $H' \to p$ appears in the derivation, must be $(\times \to)$.*

5 Power of Hypergraph Lambek Grammars

We start with showing that languages generated by HRGs can be generated by HL-grammars as well, except for some nonsubstantive cases (related to the empty word issue in the string case). In order to do this we use the weak Greibach normal form for HRGs introduced in [10] (Definition 3.1). Let us denote the number of isolated nodes in H by $isize(H)$.

Definition 5.1. *A hypergraph language L is isolated-node bounded if there is a constant $M > 0$ such that for each $H \in L$ $isize(H) < M \cdot |E_H|$.*

Theorem 5.1. *For each HRG generating an isolated-node bounded language there is an equivalent HRG in the weak Greibach normal form.*

This theorem is proved in [10]. Now, we can prove

Theorem 5.2. *For each HRG generating an isolated-node bounded language there is an equivalent hypergraph Lambek grammar.*

Proof. Let an HRG be of the form $HGr = \langle N, \Sigma, P, S\rangle$. Applying Theorem 5.1 we can assume that HGr is in the weak Greibach normal form.

The proof is essentially similar to that in the string case. Consider elements of N as elements of Pr with the same function *type* defined on them. Since HGr is in the weak Greibach normal form, each production in P is of the form $\pi = X \to G$ where G contains exactly one terminal edge e_0 (say $lab_G(e_0) = a \in \Sigma$). We convert this production into a type $T_\pi := X \div G[e_0 := \$]$. Then we introduce the HL-grammar $HGr' = \langle \Sigma, S, \triangleright\rangle$ where $\triangleright$ is defined as follows: $a \triangleright T_\pi$. An illustration of this transformation is given in Example 3.2. If $G = a^\bullet$, then we can simply write $a \triangleright X$ instead.

The main objective is to prove that $L(HGr) = L(HGr')$. Firstly, we are going to prove that $T^\bullet \overset{k}{\Rightarrow} H$ for $T \in N$, $H \in \mathcal{H}(\Sigma)$ **only if** $\mathrm{HL} \vdash f(H) \to T$ for some $f : E_H \to Tp(\mathrm{HL})$ such that $lab_H(e) \triangleright f(e)$ for all $e \in E_H$ (this would imply $L(HGr) \subseteq L(HGr')$). This is done by induction on k.

Basis. If $k = 1$, then $\pi = T \to H$ belongs to P and $E_H = \{e_0\}$. Then we can derive $\mathrm{HL} \vdash H[e_0 := T \div H[e_0 := \$]] \to T$ in one step using $(\div \to)$.

Step. Let the first step of the derivation be of the form $T \Rightarrow G$ ($\pi = T \to G \in P$) and let $E_G = \{e_0, \dots, e_n\}$ where $lab_G(e_0) \in \Sigma$ and $lab_G(e_i) \in N$ otherwise. Let $G_i \in \mathcal{H}(\Sigma)$ be a graph that is obtained from $T_i = lab_G(e_i)$ in the derivation process ($i = 1, \dots, n$). Note that $H = G[e_1/G_1, \dots, e_n/G_n]$. By the induction hypothesis, $\mathrm{HL} \vdash f_i(G_i) \to T_i$ for such $f_i : E_{G_i} \to Tp(\mathrm{HL})$ that $lab_{G_i}(e) \triangleright f_i(e)$. Then f_i can be combined into a single function f as follows: $f(e) := f_i(e)$ whenever $e \in G_i$ and $f(e_0) := T_\pi$. Then we construct the following derivation (recall that $T_\pi = T \div G[e_0 := \$]$):

$$\frac{T^\bullet \to T \quad f_1(G_1) \to T_1 \quad \dots \quad f_n(G_n) \to T_n}{(G[e_0 := \$])[e_0 := T_\pi][e_1/f_1(G_1), \dots, e_n/f_n(G_n)] \to T}\ (\div \to)$$

This completes the proof since $G[e_0 := T_\pi][e_1/f_1(G_1), \dots, e_n/f_n(G_n)] = f(H)$.

Secondly, we explain why $L(HGr') \subseteq L(HGr)$. Note that types in the dictionary of HGr' are simple; thus for each derivable sequent of the form $H \to S$ where H is over this dictionary we can apply Theorem 4.3 and obtain a derivation where each premise except for, possibly, the first one is an axiom. Now we can transform a derivation tree in HL into a derivation of the HRG HGr: each application of $(\div \to)$ such that a type T_π appears after it is transformed into an application of π in HGr. Formally, we have to use induction again. □

This theorem also shows that hypergraph basic categorial grammars introduced in [8] can be embedded in HL-grammars similarly. The rule $(\div)$, which is the main transformation in [8], is closely related to $(\div \to)$.

Corollary 5.1. *The membership problem for HL-grammars is NP-complete.*

Proof. The problem is in NP since if a graph H belongs to $L(HGr)$ given H and HGr, then this can be justified by listing a function f from edges of H to types from $dict(HGr)$ and by a derivation tree of $f(H) \to S$ including descriptions of all arising isomorphisms (here S is a distinguished type in HGr). Such a certificate has polynomial size w.r.t. H.

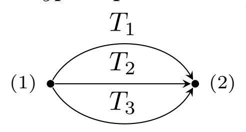
The problem is NP-complete, because there is an NP-complete isolated-node bounded language generated by an HRG (see [2]), and hence by some HL-grammar too. □

5.1 Finite Intersections of HCFLs

It is known that multiplication in L (i.e. an operation $A \cdot B$) may be considered as some kind of conjunction of A and B such that we have both A and B combined in a single type. In the graph case, we can use multiplication (i.e. $\times$) in a more general way than for strings: any graph structure can be put inside $\times$. What if there is a way to imitate behaviour of a real conjunction using $\times$ and thus to model intersections of languages? Below we investigate this idea.

Definition 5.2. *An ersatz conjunction $\wedge_E(T_1,\dots,T_k)$ of types $T_1,\dots,T_k \in Tp(\mathrm{HL})$ (such that $type(T_1) = \dots = type(T_k) = m$) is the type $\times(H)$ where*

1. *$V_H = \{v_1,\dots,v_m\}$;*
2. *$E_H = \{e_1,\dots,e_k\}$;*
3. *$att_H(e_i) = v_1 \dots v_m$;*
4. *$lab_H(e_i) = T_i$;*
5. *$ext_H = v_1 \dots v_m$.*

Example 5.1. Let T_1, T_2, T_3 be types with *type* equal to 2. Then their ersatz conjunction equals $\wedge_E(T_1,T_2,T_3) = \times\left(\right.$  $\left.\right)$.

Using this construction we can prove the main result of this paper.

Theorem 5.3. *If $HGr'_1,\dots,HGr'_k$ are HRGs generating isolated-node bounded languages, then there is an HL-grammar HGr such that $L(HGr) = L(HGr'_1) \cap \dots \cap L(HGr'_k)$.*

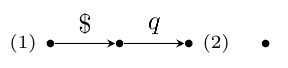
Proof. Following the proof of Theorem 5.2 we construct an HL-grammar HGr_i for each $i = 1\dots,k$ such that $L(HGr'_i) = L(HGr_i)$. We assume without loss of generality that types involved in HGr_i and HGr_j for $i \neq j$ do not have common primitive subtypes (let us denote the set of primitive subtypes of types in $dict(HGr_i)$ as Pr_i). Let us denote $HGr_i = \langle \Sigma, s_i, \triangleright_i \rangle$. Note that $type(s_1) = \dots = type(s_k)$ (otherwise $L(HGr_1) \cap \dots \cap L(HGr_k) = \emptyset$, and the theorem holds due to trivial reasons). The main idea then is to do the following: given $a \triangleright_i T_i,\ i = 1,\dots,k$ we join $T_1,\dots,T_k$ using ersatz conjunction. A distinguished type of the new grammar will also be constructed from $s_1,\dots s_k$ using $\wedge_E$. Then a derivation is expected to split into k independent parts corresponding to derivations in grammars $HGr_1,\dots,HGr_k$. However, there is a nuance that spoils simplicity of this idea; it is related to the issue of isolated nodes. This nuance leads to a technical trick, which we call “tying balloons”.

Let us fix $(k-1)$ new primitive types $b_1,\dots,b_{k-1}$ (“balloon” labels) such that $type(b_i) = 1$. For $j < k$ we define a function $\varphi_j : dict(HGr_j) \to Tp(\mathrm{HL})$ as follows: $\varphi_j(p) = p$ whenever $p \in Pr$; $\varphi_j(p \div D) = \times(M) \div D'$ where

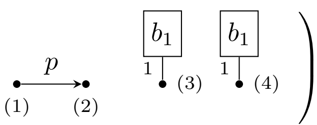
1. $D' = \langle V_D, E_D, att_D, lab_D, ext_D w\rangle$ where $[w] = V_D \backslash [ext_D]$ (that is, w consists of nodes that are not external in D).
2. Denote $m = |w| = |V_D| - |ext_D|$, and $t = type(p)$. Then $M = \langle \{v_1,\dots,v_{t+m}\}, \{e_0,e_1,\dots,e_m\}, att, lab, v_1 \dots v_{t+m}\rangle$ where $att(e_0) = v_1 \dots v_t$, $lab(e_0) = p$; $att(e_i) = v_{t+i}$, $lab(e_i) = b_j$ whenever $i = 1,\dots,m$.

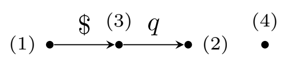
Informally, we make all nodes in the denominator D external, while $\times(M)$ "ties a balloon" labeled b_j to each node corresponding to a nonexternal one in D. Presence of these "balloon edges" is compensated by modified types of the grammar HGr_k. Namely, we define a function $\varphi_k : dict(HGr_k) \to Tp(\mathrm{HL})$ as follows: $\varphi_k(p) = p$ whenever $p \in Pr$; $\varphi_k(p \div D) = p \div D'$ where $D' = \langle V_D, E_D \cup \{e_1, \ldots, e_{(k-1)m}\}, att, lab, ext_D\rangle$ such that:

1. $m = |V_D| - |ext_D|$;
2. $e_1, \ldots, e_{(k-1)m}$ are new edges;
3. $att|_{E_D} = att_D$, $lab|_{E_D} = lab_D$;
4. If $v_1, \ldots, v_m$ are all nonexternal nodes of D, then $att(e_i) = v_{\lceil i/(k-1)\rceil}$ for $i = 1, \ldots, (k-1)m$. In other words, we attach $(k-1)$ new edges to each nonexternal node of D.
5. $lab(e_i) = b_{g(i)}$, $i = 1, \ldots, (k-1)m$ where $g(i) = [i \mod (k-1)]$ if $(k-1) \nmid i$ and $g(i) = k-1$ otherwise. That is, for each $b_i, i = 1, \ldots, (k-1)$ and for each nonexternal node there is a b_i-labeled edge attached to it.

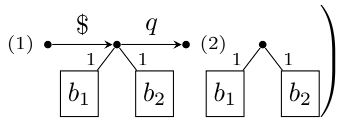
Example 5.2. Let $k = 3$, and let $T = p \div \Big($ 

$\Big)$. Then

– $\varphi_1(T) = \times \Big($ 

$\Big) \div \Big($ 

$\Big)$

– $\varphi_3(T) = p \div \Big($ 

$\Big)$

Now we are ready to introduce HGr: $HGr = \langle \Sigma, S, \triangleright\rangle$ where

– $a \triangleright T \Leftrightarrow T = \wedge_E(\varphi_1(T_1), \ldots, \varphi_k(T_k))$ and $\forall i = 1, \ldots, k \quad a \triangleright_i T_i$;
– $S = \wedge_E(s_1, \ldots, s_k)$.

The proof of $L(HGr) = L(HGr_1) \cap \cdots \cap L(HGr_k)$ is divided into two parts: the $\subseteq$-inclusion proof and the $\supseteq$-inclusion proof.

Proof of the $\supseteq$-inclusion. A hypergraph $H \in \mathcal{H}(\Sigma)$ belongs to $L(HGr_1) \cap \cdots \cap L(HGr_k)$ if and only if there are relabeling functions $f_i : E_H \to Tp(\mathrm{HL})$ such that $lab_H(e) \triangleright_i f_i(e)$ for all $e \in E_H$, and $\mathrm{HL} \vdash f_i(H) \to s_i$. Using these relabelings we construct a relabeling $f : E_H \to Tp(\mathrm{HL})$ as follows: if $f_i(e) = T_i$, then $f(e) := \wedge_E(\varphi_1(T_1), \ldots, \varphi_k(T_k))$. It follows directly from the definition that $lab_H(e) \triangleright f(e)$. Now we construct a derivation of $f(H) \to \wedge_E(s_1, \ldots, s_k)$ from bottom to top:

1. We apply rules $(\times \to)$ to all ersatz conjunctions in the antecedent. This yields a graph without $\times$-labels, which has k "layers" belonging to grammars $HGr_1, \ldots, HGr_k$.

2. We remodel a derivation of $f_1(H) \to s_1$, which consists of $(\div \to)$-applications only, using types of the form $\varphi_1(f_1(e)), e \in E_H$ that are present in $f(H)$. The only difference now is that nonexternal nodes do not "disappear" (recall that a derivation is considered from bottom to top) but edges labeled by types with $\times$ appear. Every time when $\times$ appears in the left-hand side we immediately apply $(\times \to)$, which results in adding one edge labeled by a primitive type from Pr_1 and in adding balloon edges to all nodes that would disappear in the derivation of $f_1(H) \to s_1$.
   The result of this part of a derivation is that now all types corresponding to HGr_1 left the antecedent, except for the only s_1-labeled edge attached to the external nodes of the antecedent in the right order; besides, for each nonexternal node in the antecedent there is now a balloon edge labeled by b_1 attached to it.
3. We perform $(k-2)$ more steps similarly to Step 2 using types of the form $\varphi_i(f_i(e)), 1 < i < k$ and thus remodeling a derivation of the sequent $f_i(H) \to s_i$. Upon completion of all these steps the antecedent contains:
   – Types of the form $\varphi_k(f_k(e)), e \in E_H$;
   – $(k-1)$ edges labeled by $s_1, \ldots, s_{k-1}$ resp. and attached to external nodes of the antecedent;
   – Balloon edges such that for each $j \in \{1, \ldots, k-1\}$ and for each nonexternal node there is a b_j-labeled edge attached to it.
4. We remodel a derivation of $f_k(H) \to s_k$ using types of the form $\varphi_k(f_k(e))$. A situation differs from those at steps 2 and 3 because now nonexternal nodes do disappear, and each time when this happens all balloon edges attached to a nonexternal node disappear as well.
   After this step, all balloon edges are removed, and we obtain a graph with $type(s_1)$ nodes such that all of them are external, and with k edges labeled by $s_1, \ldots, s_k$ such that their attachment nodes coincide with external nodes of the graph. This ends the proof since $\wedge_E(s_1, \ldots, s_k)$ is exactly this graph standing under $\times$.

Proof of the $\subseteq$-inclusion. Let H be in $L(HGr)$; then there is a function $\Phi : E_H \to Tp(\mathrm{HL})$ such that $\Phi(e) = \wedge_E(\varphi_1(T_1(e)), , \ldots, \varphi_k(T_k(e)))$ whenever $e \in E_H$, $lab(e) \triangleright_i T_i(e)$, and $\Phi(H) \to S$ is derivable in HL. We aim to decompose the derivation of this sequent into k ones in grammars $HGr_1, \ldots, HGr_k$. In order to do this we transform the derivation in stages:

Stage 1. Using Proposition 4.1 we replace each edge in $\Phi(H)$ labeled by a type of the form $\times(M)$ by M. A new sequent (denote it by $H' \to S$) is derivable as well.

Stage 2. The sequent $H' \to S$ fits in Theorem 4.3; hence there exists its simple derivation. Let us fix some simple derivation of $H' \to S$ and call it Δ.

Furthermore we consider all derivations from bottom to top. In particular, if we state "X is after Y" regarding some places X and Y in a derivation, then we mean that X is above Y in the derivation tree (e.g. regarding Example 3.9 we would say that $(\to \times)$ is applied after $(\to \div)$).

Stage 3. Design of types $\varphi_i(T)$ differs in the case $i < k$ and $i = k$. Namely, if $\varphi_i(T)$ for $i < k$ participates in the rule $(\div \to)$ in Δ, this affects only primitive types from Pr_i; on the contrary, participating of $\varphi_k(T)$ in $(\div \to)$ affects types from Pr_k but also balloon types $b_1, \ldots, b_{k-1}$, which appear after rule applications of $(\div \to)$ and $(\times \to)$ to types of the form $\varphi_i(T)$, $i < k$. This allows us to come up with the following conclusion: if an application of the rule $(\div \to)$ to a type of the form $\varphi_k(T)$ preceeds a rule application of $(\div \to)$ to a type of the form $\varphi_i(T)$ for $i < k$, then we can change their order (note also that all nodes in the denominator of $\varphi_i(T)$ are external). Thus Δ can be remade in such a way that all rules affecting $\varphi_k(T)$ will occur upper than rules affecting $\varphi_i(T), i < k$ in a derivation (and it will remain simple). Let us call a resulting derivation Δ'.

Stage 4. A denominator of a type $\varphi_i(T)$ for $i < k$ contains edges labeled by elements of Pr_i only. Since Δ' is simple, applications of the rule $(\div \to)$ to types of the form $\varphi_i(T)$ and $\varphi_j(T')$ for $i \neq j$ are independent, and their order can be changed. This means that we can reorganize Δ' in the following way (from bottom to top):

1. Set $i = 1$.
2. Apply the rule $(\div \to)$ to a type of the form $\varphi_i(T)$ and right away the rule $(\times \to)$ to its numerator.
3. If there still are types of the form $\varphi_i(T)$, repeat step 2;
4. If $i = k - 1$, go forward; otherwise, set $i = i + 1$ and go back to step 2.
5. Apply the rule $(\div \to)$ to types of the form $\varphi_k(T)$.
6. Now, an antecedent of the major sequent (denote this sequent as $G \to S$) does not include types with $\div$ or $\times$. S is of the form $\times(M_S)$, and Theorem 4.3 provides that the last rule applied has to be $(\to \times)$; therefore, $G = M_S$ and we reach the sequent $M_S \to S$. Consequently, $G = M_S$ consists of k edges labeled by $s_1, \ldots, s_k$ resp.

Let us call this derivation Δ_0. Observe that, after steps 1-4 in the above description, balloon edges with labels $b_1, \ldots, b_{k-1}$ may occur in the antecedent of a sequent (denote this sequent, which appears after step 4, as $G' \to S$). There is only one way for them to disappear: they have to participate in the rule $(\div \to)$ with a type of the form $\varphi_k(T)$ (since only for such types it is the case that their denominators may contain balloon edges). Note, however, that balloon edges within the denominator of $\varphi_k(T)$ are attached only to nonexternal nodes. Therefore, balloon edges in G' can be attached only to nonexternal nodes as well. Besides, if some balloon edge labeled by b_i is attached to a node $v \in V_{G'} \setminus [ext_{G'}]$, then the set of balloon edges attached to v has to consist of exactly $k - 1$ edges labeled by $b_1, \ldots, b_{k-1}$ (because in the denominator of $\varphi_k(T)$ exactly such edges are attached to each nonexternal node). Finally, note that after step 5 all nonexternal nodes disappear since M_S contains exactly $type(S)$ nodes, all of which are external. This allows us to conclude that balloon edges have to be present on all nonexternal nodes (otherwise, a nonexternal node cannot go away interacting with a type of the form $\varphi_k(T)$). Informally, a balloon edge labeled by b_i indicates that a node was used by a type from the i-th grammar HGr_i, and $\varphi_k(T)$ verifies that each nonexternal node is used by the i-th grammar exactly once.

Summarizing all the above observations, we conclude that, after steps 1-4, there is exactly one balloon edge labeled by b_i on each nonexternal node of G'

for all $i = 1, \dots, k-1$ (and no balloon edge is attached to some external node of G'). The only way for b_i to appear attached to a node (recall that we consider the derivation from bottom to top) is to participate in the rule $(\times \to)$ after the application of $(\div \to)$ to a type of the form $\varphi_i(T)$. Now we are ready to decompose Δ_0 into k ones:

- For $1 \le i < k$ we consider step 2 of Δ_0 with that only difference that we disregard balloon edges and additional external nodes added in the construction of HGr. Then the combination of rules $(\div \to)$ and $(\times \to)$ applied to a type $\varphi_i(T)$ turns into an application of the rule $(\div \to)$ to T in the HGr_i. Take into account that the only type that is built of elements of Pr_i and remains to step 6 is s_i attached to external nodes in the right order. Therefore, if we remove from H' all edges not related to HGr_i and relabel each edge having a label $\varphi_i(T)$ by T (call the resulting graph H'_i), then $H'_i \to s_i$ is derivable.
- For $i = k$ everything works similarly; however, instead of step 2 we have to look at step 5 and again not to consider balloon edges. Then an application of $(\div \to)$ to $\varphi_k(T)$ is transformed into a similar application of $(\div \to)$ to T in HGr_k. After the whole process, only s_k remains, so, if H'_k is a graph obtained from H' by removing edges not related to HGr_k and changing each label of the form $\varphi_k(T)$ by T, then $H'_k \to s_k$ is derivable.

Finally note that $H'_i = \Phi_i(H)$ where $\Phi_i(e) = T_i(e)$. The requirement $lab(e) \triangleright_i T_i(e)$ completes the proof, because thus $H \in L(HGr_i)$ for all $i = 1, \dots, k$. □

The balloon trick is used in the proof to control that making all nodes in denominators of $\varphi_i(T)$ external $(i < k)$ does not lead to using, e.g., a nonexternal isolated node in rules $(\div \to)$ more than once.

Corollary 5.2. *The language* $\{\mathrm{SG}(a^{2n^2}), n > 0\}$ *can be generated by some* HL-*grammar.*

Proof. The string language $L_1 = (\{a^n b^n \mid n > 0\})^+$ is context-free, and so is $L_2 = \{a^k b^{m_1} a^{m_1} b^{m_2} a^{m_2} \dots b^{m_{k-1}} a^{m_{k-1}} b^l | k, m_i, l > 0\}$. Consequently, languages $\mathrm{SG}(L_1) = \{\mathrm{SG}(w) \mid w \in L_1\}$ and $\mathrm{SG}(L_2)$ are generated by some HRGs. The language $L_3 = L_1 \cap L_2$ equals $L_3 = \{(a^n b^n)^n \mid n > 0\}$, so $\mathrm{SG}(L_3)$ is a finite intersection of HCFLs and can be generated by some HL-grammar $\langle \{a, b\}, S, \triangleright \rangle$. Finally, note that $HGr := \langle \{a\}, S, \triangleright' \rangle$ where $a \triangleright' T \Leftrightarrow a \triangleright T$ or $b \triangleright T$ recognizes the language $L = \{\mathrm{SG}(a^{2n^2}), n > 0\}$. □

Corollary 5.3. *The pumping lemma and the Parikh theorem do not hold for languages generated by HL-grammars.*

Proof. The language $\{\mathrm{SG}(a^{2n^2}), n > 0\}$ contradicts both theorems. □

6 Conclusion

In the string case, there is a disparity between context-free grammars and Lambek grammars: while generating the same set of languages, the former can be parsed in polynomial time (the CYK algorithm) but the membership problem for the latter is NP-complete (see [6]). Situation changes dramatically in the hypergraph case: hypergraph Lambek grammars introduced in this paper have the same algorithmic complexity as hyperedge replacement grammars but they generate much more languages.

Of a particular note is that HL-grammars recognize more languages than finite intersections of hypergraph context-free languages (due to article limits, we shall not prove this here). Moreover, nontrivial upper bounds restricting the power of HL-grammars (like the pumping lemma for HRGs) are unknown; it is subject to be investigated further. In particular, it would be interesting to answer the question whether languages generated by HL-grammars are closed under intersection (Theorem 5.3 gives us a hope that this could be true). For now there is no clear way for us how to prove this.

To conclude, HL-grammars represent a curious mechanism opposed to HRGs in the underlying concepts. From the point of view of parsing, HL-grammar is not a simple formalism; however, it is more powerful than HRGs and hence is worth consideration. It is also important to notice that the hypergraph Lambek calculus itself is an interesting logic, so we hope that in the future all these formalisms will be studied deeper in their different aspects.

Acknowledgments

I thank my scientific advisor prof. Mati Pentus for fruitful discussions.

References

1. Bar-Hillel, Y., Gaifman, C., Shamir, E.: On categorial and phrase-structure grammars. Bulletin of the Research Council of Israel F(9), 1–16 (1963)
2. Drewes, F., Kreowski, H.-J., Habel, A.: Hyperedge replacement graph grammars (1997)
3. Kanazawa, M.: Second-Order Abstract Categorial Grammars as Hyperedge Replacement Grammars. J of Log Lang and Inf 19, 137–161 (2010).
4. Lambek, J.: The mathematics of sentence structure. Am. Math. Monthly 65(3), 154–170 (1958).
5. Moortgat, M.: Multimodal Linguistic Inference. J. Log. Lang. Inf. 5(3/4), 349-385 (1996).
6. Pentus, M.: Lambek calculus is NP-complete. Theor. Comput. Sci. 357 (1-3), 186-201 (2006)
7. Pentus, M.: Lambek Grammars Are Context Free. Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada. IEEE Computer Society (1993).
8. Pshenitsyn, T.: Hypergraph Basic Categorial Grammars. In: Gadducci F., Kehrer T. (eds) Graph Transformation. ICGT 2020. Lecture Notes in Computer Science, vol 12150. Springer, Cham (2020).
9. Pshenitsyn, T.: Introduction to a Hypergraph Logic Unifying Different Variants of the Lambek Calculus. Preprint at arXiv.org. https://arxiv.org/abs/2103.01199
10. Pshenitsyn, T.: Weak Greibach Normal Form for Hyperedge Replacement Grammars. Electronic Proceedings in Theoretical Computer Science 330, 108-125. Open Publishing Association (2020).