Grammars Based on a Logic of Hypergraph Languages

Tikhon Pshenitsyn

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

`ptihon at yandex.ru`

The hyperedge replacement grammar (HRG) formalism is a natural and well-known generalization of context-free grammars. HRGs inherit a number of properties of context-free grammars, e.g. the pumping lemma. This lemma turns out to be a strong restriction in the hypergraph case: it implies that languages of unbounded connectivity cannot be generated by HRGs. We introduce a formalism that turns out to be more powerful than HRGs while having the same algorithmic complexity (NP-complete). Namely, we introduce hypergraph Lambek grammars; they are based on the hypergraph Lambek calculus, which may be considered as a logic of hypergraph languages. We explain the underlying principles of hypergraph Lambek grammars, establish their basic properties, and show some languages of unbounded connectivity that can be generated by them (e.g. the language of all graphs, the language of all bipartite graphs, the language of all regular graphs).

1 Introduction: Productions vs Types

Formal grammar theory is an area at the intersection of linguistics, logic, programming etc., which studies an issue of how complex, unboundedly large, but in some sense regular families of objects (strings, terms, graphs, pictures, ...), can be described using some class of finite-sized mechanisms. There exist numerous kinds of formal grammars based on different principles. For instance, standard context-free grammars deal with strings; they consist of terminal and nonterminal alphabets, a nonterminal start symbol S, and the list of productions of the form $X \to \alpha$, which allow one to replace the nonterminal symbol X by the string α.

Example 1. Look at the following "linguistic" example (which is extremely primitive from the linguistic point of view):

$$\text{Productions:} \quad S \to NP \text{ sleeps} \qquad NP \to \text{the } N \qquad N \to \text{cat}$$
$$\text{Derivation:} \quad S \Rightarrow NP \text{ sleeps} \Rightarrow \text{the } N \text{ sleeps} \Rightarrow \text{the cat sleeps}$$

In the production $[S \to NP \text{ sleeps}]$, there is exactly one terminal object (*sleeps* here) in its right-hand side. Let us transform this production as follows: sleeps $\triangleright NP \backslash S$. The $\triangleright$ sign is to be read as "is of the type", and $NP \backslash S$ stands for the type of such objects u that, whenever we add an object v of the type NP (NP stands for the class of *singular noun phrases*, which includes e.g. phrases *the cat*, *Helen*, *a colorless green idea* etc.) to the left of u, vu forms an object of the type S (S stands for *sentence*). Therefore, sleeps $\triangleright NP \backslash S$ means that the verb *sleeps* is such an object that, whenever a noun phrase (singular) appears to its left, they together form a sentence. This is correct, to a first approximation: *the cat sleeps*, *Helen sleeps*, *a green colorless idea sleeps* etc. are correct English sentences. Similarly, we

---

can transform the production $[NP \to$ the $N]$ into a correspondence: the $\triangleright NP/N$ (N stands for singular nouns with their dependents that do not represent a specific object but rather a class of objects: *cat*, *colorless green idea*). Note that the direction of the division is different from the previous one. The following reduction laws hold: the left one $A, A\backslash B \to B$, and the right one $B/A, A \to B$. The type A/B ($B\backslash A$) can be understood as the type of functions that take an argument of the type B from the right (from the left resp.) and return a value of the type A.

In linguistics, however, it is not enough to have the above reduction laws to describe language phenomena of interest. E.g. sometimes it is useful to have the rule $NP \to S/(NP\backslash S)$ or the rule $A/B, B/C \to A/C$. Besides, there is a need for an operation that would store pairs of units of certain types. E.g. when we consider sentences like *Tim gave the lemon to Melany and the lime to Amelie*, we would like to assign the type $((NP \cdot PP)\backslash(NP \cdot PP))/(NP \cdot PP)$ to the word *and* (where PP is the type of prepositional phrases with *to*: *to Melany*, *to Amelie*) since it receives a pair consisting of a noun phrase and a prepositional phrase from the right, and a similar pair from the left. $A \cdot B$ is the type of pairs uv such that u is of the type A, and v is of the type B; therefore, $A \cdot B$ works as pairwise concatenation.

The above connectives $\backslash, \cdot, /$ belong to the Lambek calculus (L) — a logical calculus introduced in [7]. Types in the Lambek calculus are built from primitive types $p_1, p_2, \ldots \in Pr$ using $\backslash, \cdot, /$. We focus on the Lambek calculus in the Gentzen style; this means that it deals with sequents, which are structures of the form $A_1, \ldots, A_n \to A$ for A_i, A being types ($A_1, \ldots, A_n$ is called an antecedent, and A is called a succeedent). The calculus L has one axiom and six inference rules:

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

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

An axiom can be considered as an inference rule with zero premises. Hereinafter, primitive types are denoted by small Latin letters (in particular, from now on we write s instead of S, np instead of NP etc.), types are denoted by capital Latin letters, and sequences of types are denoted by capital Greek letters; besides, Π, Ψ above are nonempty. A sequent $\Pi \to A$ is called *derivable in* L (denoted $L \vdash \Pi \to A$) if it can be obtained by applications of inference rules.

Example 2. The following sequents are derivable in L (their derivations are presented below them):

- $np/n, n, np\backslash s \to s$;
- $np \to s/(np\backslash s)$;
- $p \to (p \cdot q)/q$.

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

Finally, the above reasonings bring us to using the Lambek calculus as a grammar formalism. A grammar is a correspondence (i.e., a binary relation) $\triangleright$ between terminal objects and types along with some fixed type S; a string is said to be correct if its elements can be replaced by coresponding types in such a way that the sequent composed of these types in the antecedent and of S in the succeedent is derivable in L.

Another feature characterizing the Lambek calculus is its language models. Namely, let us consider a function $w : Pr \to \mathscr{P}(\Sigma^*)$ called *a valuation*; this function assigns a formal language over some fixed alphabet Σ to each primitive type. It can be extended to types and sequents according to principles stated earlier; namely, $\overline{w}$ is defined as follows:

1. $\overline{w}(B\backslash A) = \{u \in \Sigma^* \mid \forall v \in \overline{w}(B)\ vu \in \overline{w}(A)\};$ 4. $\overline{w}(A_1, \ldots, A_n) = \overline{w}(A_1 \cdot \ldots \cdot A_n);$

2. $\overline{w}(A/B) = \{u \in \Sigma^* \mid \forall v \in \overline{w}(B)\ uv \in \overline{w}(A)\};$

3. $\overline{w}(A \cdot B) = \{uv \mid u \in \overline{w}(A), v \in \overline{w}(B)\};$ 5. $\overline{w}(\Pi \to A)$ is true if and only if $\overline{w}(\Pi) \subseteq \overline{w}(A).$

Pentus [9] proved that $L \vdash \Pi \to A$ if and only if $\overline{w}(\Pi \to A)$ is true for all valuations w. This allows us to call the Lambek calculus the logic of formal languages in the sense that it describes all the true facts about formal languages in the signature $\backslash, \cdot, /, \subseteq$ and only them.

It is often important to work with more complex structures than strings. This is the reason why a wide variety of extensions of generative grammars to terms, graphs etc. has been introduced. In this paper, we focus on a particular approach called *hyperedge replacement grammars (HRGs)*. A survey on HRGs can be found in [3, 5]; in this paper, we mainly follow the definitions and notation from [3]. We chose HRGs as the basis for our studies since they are very close to context-free grammars in the sense of definitions, underlying mechanisms and properties. Our main goal is to extend the Lambek calculus and Lambek grammars to hypergraphs in a natural way and to study the resulting notions.

In Section 2, we define fundamental notions regarding hypergraphs and hyperedge replacement. In Section 3, we introduce the hypergraph Lambek calculus extending notions of types, sequents, axioms and rules. The formal definition of hypergraph Lambek grammars will be given in Section 5. There we also study the power of these grammars; it turns out that they can generate more languages than HRGs, e.g. the language of all graphs, the language of all bipartite graphs, the language of all regular graphs. Since the membership problem for HL-grammars is NP-complete and since they generate all isolated-node bounded languages generated by HRGs, they can be considered as an attractive alternative to HRGs. In Section 6, we conclude and outline further research directions regarding HL and HL-grammars.

2   Hyperedge Replacement

Formal definitions of hypergraphs and of hyperedge replacement are given in this section according to [3, Chapter 3].

$\mathbb{N}$ includes 0. Σ^* is the set of all strings over the alphabet Σ including the empty string Λ. $\Sigma^{\circledast}$ is the set of all strings consisting of distinct symbols (they may be considered as ordered sets). 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 $f : \Sigma^* \to \Delta^*$ ($f(\sigma_1 \ldots \sigma_k) = f(\sigma_1) \ldots f(\sigma_k)$).

Definition 1. A ranked set is a set C along with the ranking function $rank : C \to \mathbb{N}$.

Definition 2. *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 an ordered set of attachment nodes to each edge, $lab : E \to C$ labels each edge by some element of C in such a way that $rank(lab(e)) = |att(e)|$ whenever $e \in E$, and $ext \in V^{\circledast}$ is an ordered set 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 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 definitions will be preserved; however, in this paper, we stick to the above definition.

Definition 3. The function $rank_G$ (or $rank$, if G is clear) returns the number of nodes attached to an edge in a graph G: $rank_G(e) := |att_G(e)|$. If G is a graph, then $rank(G) := |ext_G|$.

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 $e \in E_H$ $att_H(e) = att_G(e)$, $lab_H(e) = lab_G(e)$.

If $H = \langle \{v_i\}_{i=1}^n, \{e_0\}, att, lab, v_1 \ldots v_n \rangle$, $att(e_0) = v_1 \ldots v_n$ and $lab(e_0) = a$, then H is called *a handle*. In this work, we denote it by $a^\bullet$.

An isomorphism between graphs G and H is a pair of bijective functions $\mathscr{E} : E_G \to E_H$, $\mathscr{V} : V_G \to V_H$ such that $att_H \circ \mathscr{E} = \mathscr{V} \circ att_G$, $lab_G = lab_H \circ \mathscr{E}$, $\mathscr{V}(ext_G) = ext_H$. 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 4. A *string graph* induced by a string $w = a_1 \ldots a_n$ 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 $att(e_i) = v_{i-1} v_i$, $lab(e_i) = a_i$. It is denoted by SG(w).

We additionally introduce the following definition (not from [3]):

Definition 5. Let $H \in \mathscr{H}(C)$ be a graph. Let $f : E_H \to C$ be a function, where $rank(lab_H(e)) = rank(f(e))$ for $e \in E_H$. Then $f(H) = \langle V_H, E_H, att_H, f, ext_H \rangle$.

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

Definition 6. *Hyperedge replacement* is defined in [3], and it plays a fundamental role in hyperedge replacement grammars. The replacement of an edge e_0 in G with a graph H can be done if $rank(e_0) = rank(H)$ 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 in $1..rank_G(e_0)$, fuse the i-th external node of H with the i-th attachment 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 [3]). The following notation is in use: if $e_1, \ldots, e_k$ are distinct edges of a graph H and they are simultaneously replaced by graphs $H_1, \ldots, H_k$ resp. (this requires $rank(H_i) = rank(e_i)$), then the result is denoted $H[e_1/H_1, \ldots, e_k/H_k]$.

3   Hypergraph Lambek Calculus

HRGs can be used to describe linguistic structures as well as context-free grammars since linguistic objects often have an underlying structure, which is more complex than a string. One of the recently studied applications is using HRGs for abstract meaning representation (see e.g. [2, 4, 6]): the meaning of a sentence is represented by a graph. In [1], HRGs are used for modelling nonprojective dependencies in Dutch. Another example where graph structures occur in linguistics is syntactic trees. Given a context-free grammar, it is natural to represent an internal hierarchical structure of constituents of a generated sentence by a tree.

Example 3. The HRG $HGr = \langle \{S, NP, N\}, \{\text{the}, \text{cat}, \text{sleeps}, l, r\}, P, S \rangle$ with the list of productions P defined below generates the graph *Syntree*:
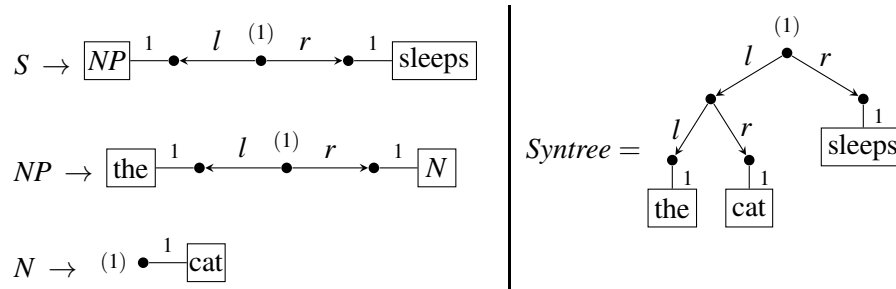
Here $rank(S) = rank(NP) = rank(N) = rank(\text{sleeps}) = rank(\text{the}) = rank(\text{cat}) = 1$, and $rank(l) = = rank(r) = 2$.

Syntree is a simplified syntactic tree for the sentence *the cat sleeps*; l and r distinguish left and right children in the tree.

Since there are such cases in linguistics where we need to work with graphs rather than with strings, we would like to generalize the categorial point of view discussed in Section 1 to hypergraphs. Our first attempt was a generalization of basic categorial grammars to hypergraphs; the resulting formalism called *hypergraph basic categorial grammar* was introduced at ICGT 2020 [10]. However, this formalism did not significantly improve our insight since most results were proved in the same way as for strings; in particular, such grammars generate the same set of languages as HRGs (with some nonsubstantial exceptions related to the number of isolated nodes). In this paper, we aim to go further and to discuss how the Lambek calculus along with its grammars can be generalized to hypergraphs. Note that, in the string case, Lambek grammars (i.e. grammars based on the Lambek calculus) are equivalent to context-free grammars and to basic categorial grammars. (This nontrivial result was proved in [8].)

It is known that strings can be represented as string graphs, e.g. represents the string *the cat sleeps* (this graph is denoted as SG(the cat sleeps)). The production $S \to NP$ sleeps then is transformed into the graph production $S \to \text{SG}(NP \text{ sleeps})$. We want to transform this production into a correspondence as it was done in Section 1; there we "took out" the terminal unit *sleeps* from the right-hand side, and assigned the type $NP\backslash S$ to it. Now, we are going to do the same operation but we shall mark an edge, from which we took out the label *sleeps*, by a special $ symbol:

$$S \to \text{SG}(NP \text{ sleeps}) \quad \rightsquigarrow \quad \text{sleeps} \triangleright S/\text{SG}(NP \$)$$

In general, $ denotes the hyperedge, from which we took out its label. Similarly, the production $[NP \to \text{the } N]$ is transformed as follows: the $\triangleright NP/\text{SG}(\$ N)$. Note that now we do not need two divisions $\backslash$ and $/$ anymore since the difference between them is now expressed by the position of the $-labeled edge. In order to distinguish the string divisions and the new graph division, and also to stress that the latter is undirected, we write $A \div D$ instead of A/D for the latter.

The conversion of a production into a correspondence between a terminal unit (a symbol, a label, a word) and a type requires that there is exactly one terminal unit in the right-hand side of the production. This property is called weak Greibach normal form. Note that, if we have e.g. a production $S \to \text{SG}(ABaCD)$ where S, A, B, C, D are nonterminal, and a is terminal, then we can also use the above $-notation and write $a \triangleright S \div \text{SG}(AB\$CD)$. As in Section 1, this means that a is such an object that whenever objects of types A, B, C and D are placed instead of corresponding edges in the graph SG($AB\$CD$), and a is placed on the $-labeled edge, the resulting structure forms an object of the type S. We can proceed similarly with an arbitrary hypergraph production, if the grammar is in the weak Greibach normal

form. This is the main idea of hypergraph basic categorial grammars. However, as in the string case, we would like to go further and to consider more complex types; besides, we would also like to generalize the operation $A \cdot B$. This results in the following

Definition 7. The set $Tp(\mathrm{HL})$ of types of the hypergraph Lambek calculus HL is defined inductively as the least set satisfying the following conditions:

1. Primitive types Pr are types. Pr is a countable ranked set such that for each n there are infinitely many types $p \in Pr$ such that $rank(p) = n$.

2. Let N ("numerator") be a type. Let D ("denominator") be a hypergraph such that exactly one of its hyperedges (call it e_0) is labeled by \$, and the other hyperedges (possibly, there are none of them) are labeled by elements of $Tp(\mathrm{HL})$; let also $rank(N) = rank(D)$. Then $T = (N \div D)$ is a type, and $rank(T) := rank_D(e_0)$.

3. Let M be a hypergraph such that all its hyperedges are labeled by types from $Tp(\mathrm{HL})$ (possibly, there are no hyperedges at all). Then $T = \times(M)$ is also a type, and $rank(T) := rank(M)$.

Below we often write $N \div (D)$ or $N \div D$ instead of $(N \div D)$. Item 2 generalizes the concept of $\div$ explained earlier: $N \div D$ is the type of such hypergraphs H that, if we replace the \$-labeled edge in D by H and for all the remaining edges $e_i, i > 0$, which are labeled by some types T_i, we replace them by hypergraphs H_i, which are of types T_i, then we obtain a hypergraph of the type N. In particular, this explains why we require $rank(N) = rank(D)$ and $rank(N \div D) = rank_D(e_0)$.

Example 4. The first production from Example 3 can be transformed into the following correspondence:

$$
\text{sleeps} \rhd s \div \left(\boxed{np} \xrightarrow{\;1\;} \bullet \xleftarrow{\;\;} \overset{p_l \;\;(1)\;\; p_r}{\bullet} \xrightarrow{\;\;} \bullet \xrightarrow{\;1\;} \boxed{\$} \right) \tag{1}
$$

Here p_l, p_r are primitive types introduced to deal with special "technical" labels l (r resp.). According to our general understanding of types, one may say that l (r) is of the type p_l (p_r resp.). Now, (1) means that

sleeps is such an object that, if we place it instead of \$ within the graph $\boxed{np} \xrightarrow{\;1\;} \bullet \xleftarrow{\;} \overset{p_l \;\;(1)\;\; p_r}{\bullet} \xrightarrow{\;} \bullet \xrightarrow{\;1\;} \boxed{\$}$, and we place any object (syntactic tree) of the type np instead of the np-labeled edge, then we obtain an object of the type s (we also need to replace p_l by l and p_r by r).

The operation $\times(M)$ can be called a hypergraph product, or a hypergraph concatenation. Its general semantics it the following: if $E_M = \{m_1, \ldots, m_l\}$, $lab_M(m_i) = T_i$, and H_i is a hypergraph of the type T_i ($i = 1, \ldots, l$), then the hypergraph $M[m_1/H_1, \ldots, m_l/H_l]$ is a hypergraph of the type $\times(M)$. Thus, $\times(M)$ is the type of all substitution instances of M.

Example 5. If *str* is the primitive type of all string graphs labeled by the blank symbol $*$, then the type

$$\times \left((1) \bullet \underset{str}{\overset{str}{\rightleftarrows}} \bullet (2) \right)$$ is the type of all graphs consisting of two parallel strings with the common start

and finish nodes, e.g. $(1) \bullet \!\!\!\begin{array}{c} \nearrow \bullet \rightarrow \bullet \searrow \\ \searrow \bullet \nearrow \end{array}\!\!\! \bullet (2)$.

Moving away from the intuition of $\div$ and $\times$, we would like to introduce a syntactic *calculus*, which would work with types introduced in Definition 7 by means of axioms and rules. We expect that this calculus should generalize the Lambek calculus in the Gentzen style introduced in Section 1. This is done in our preprint [12]; in this paper, we only introduce the axiom and rules of the hypergraph Lambek calculus (HL) without a detailed discussion of why they actually generalize those of L.

Definition 8. *A hypergraph sequent is a structure of the form $H \to A$, where $A \in Tp(\mathrm{HL})$ is a type, $H \in \mathscr{H}(Tp(\mathrm{HL}))$ is a hypergraph labeled by types and $rank(H) = rank(A)$. H is called the antecedent of the sequent, and A is called the succedent of the sequent.*

Remark. Returning to our intuition, $H \to A$ could be understood as the statement "each hypergraph of type H is also of type A" (in a similar way as for L that we described in Section 1). However, we do not have types of the form H for H being a hypergraphs. Note that there we defined $\overline{w}(A_1, \dots, A_n)$ as $\overline{w}(A_1 \cdot \dots \cdot A_n)$ in the string case. This allows us to conclude that here we should also understand $H \to A$ as the statement "each hypergraph of type $\times(H)$ is also of type A"

The hypergraph Lambek calculus HL deals with hypergraph sequents and explains, which of them are *derivable* using axioms and rules. The only **axiom** of HL is the following: $p^{\bullet} \to p$, $p \in Pr$ ($p^{\bullet}$ here is the p-handle). Rules are presented below along with some simple examples.

1. **Rule $(\div \to)$.** Let $N \div D$ be a type and let $E_D = \{d_0, d_1, \dots, d_k\}$ where $lab(d_0) = \$$, $lab(d_i) = T_i$ for $i \geq 1$. Let $H \to A$ be a hypergraph sequent and let $e \in E_H$ be labeled by N. Let finally $H_1, \dots, H_k$ be hypergraphs labeled by types. Then the rule $(\div \to)$ is the following:

$$\frac{H \to A \quad H_1 \to T_1 \quad \dots \quad H_k \to T_k}{H[e/D][d_0 := N \div D][d_1/H_1, \dots, d_k/H_k] \to A} \ (\div \to)$$

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

Example 6. Consider the following rule application with T_i being some types and with T being equal to $q \div SG(T_2 \$ T_3)$:

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

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

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

This rule is understood as follows: if there are such hypergraphs D, F and such a type N that in a sequent $H \to N$ the hypergraph H equals $D[e_0/F]$ and $H \to N$ is derivable, then $F \to N \div D$ is also derivable.

Example 7. Consider the following rule application where N equals $\times(SG(pqr))$ (here we draw string graphs instead of writing $SG(w)$ to visualize the rule application):

3. **Rule $(\times \to)$.** Let $G \to A$ be a hypergraph 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 is formulated from bottom to top as the previous one. Intuitively speaking, there is a sub-graph of an antecedent in a premise, and it is "compressed" into a single $\times(F)$-labeled hyperedge.

Example 8. Consider the following rule application where U equals $\times(\mathrm{SG}(pqrs))$:

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

4. **Rule** $(\to\times)$. Let $\times(M)$ be a type and let $E_M=\{m_1,\ldots,m_l\}$. Let $H_1,\ldots,H_l$ be graphs. Then

$$\dfrac{H_1\to lab(m_1)\quad\ldots\quad H_l\to lab(m_l)}{M[m_1/H_1,\ldots,m_l/H_l]\to\times(M)}\ (\to\times)$$

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

Example 9. Consider the following rule application with T_i being some types:

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

Definition 9. A hypergraph sequent $H\to A$ is *derivable in* HL, written as $\mathrm{HL}\vdash H\to A$, 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 10. Let $rank(s)=rank(np)=rank(n)=1$, $rank(p_l)=rank(p_r)=2$, and let

$$V=s\div\left(\boxed{np}\xrightarrow{\ 1\ }\bullet\xleftarrow{\ p_l\ (1)\ p_r\ }\bullet\xrightarrow{\ 1\ }\boxed{\$}\right);\qquad Adj=np\div\left(\boxed{\$}\xrightarrow{\ 1\ }\bullet\xleftarrow{\ p_l\ (1)\ p_r\ }\bullet\xrightarrow{\ 1\ }\boxed{n}\right).$$

Then the following is the derivation of the below sequent:

$$\dfrac{s^\bullet\to s\quad np^\bullet\to np\quad p_l^\bullet\to p_l\quad p_r^\bullet\to p_r}{\boxed{np}\ \ \boxed{V}\ \to s}\ (\div\to)$$

one would expect that hypergraph languages can be considered as models of HL. In Section 4, we formally define language models (L-models in short) for HL in a way similar to how we introduced the notion of valuation $\overline{w}$ in Section 1. We establish correctness of HL with respect to L-models, and completeness of its $\times$-free fragment.

The following statements can be proved in a similar way as for strings (see [12]):

Theorem 1 (cut elimination). *If $H \to A$ and $G \to B$ are derivable in* HL*, and $e_0 \in E_G$ is labeled by A, then $G[e_0/H] \to B$ is also derivable in* HL*.*

Proposition 1 (reversibility of $(\times \to)$ and $(\to \div)$)**.**

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

2. *If* $\text{HL} \vdash H \to N \div D$ *and* $e_0 \in E_D$ *is labeled by* \$*, then* $\text{HL} \vdash D[e_0/H] \to N$*.*

These statements will be used in proofs of some results in this paper.

4 Language Models for HL

In previous sections, we looked at types of the Lambek calculus (either in its string or in its hypergraph versions) as hypergraph languages; divisions and product were interpreted as operations on languages. In this section, we are going to formalize this idea for HL in a way similar to what we have done in Section 1 and in Examples 4 and 5.

Definition 10. Given a ranked alphabet Σ, we call a function $w : Pr \to \mathscr{P}(\mathscr{H}(\Sigma))$ *a valuation* if for each $p \in Pr$ $rank(H) = rank(p)$ whenever $H \in w(p)$. This function assigns a hypergraph language to each primitive type. Its extension $\overline{w}$ is the following function from the set of hypergraph types to $\mathscr{P}(\mathscr{H}(\Sigma))$:

1. Let $N \div D$ be a type and let $E_D = \{d_0, \ldots, d_k\}$, $lab_D(d_0) = \$$, $lab_D(d_i) = T_i$. Then $\overline{w}(N \div D)$ consists of all graphs G such that $D[d_0/G, d_1/H_1, \ldots, d_k/H_k]$ belongs to $\overline{w}(N)$ whenever $H_1 \in \overline{w}(T_1)$, ..., $H_k \in \overline{w}(T_k)$.

2. Let $\times(M)$ be a type and let $E_M = \{m_1, \ldots, m_l\}$, $lab_M(m_i) = T_i$. Then $\overline{w}(\times(M))$ consists of all graphs of the form $M[m_1/H_1, \ldots, m_l/H_l]$ for all $H_i \in \overline{w}(T_i)$.

3. We additionally define $\overline{w}(H \to A)$ as the statement $\overline{w}(\times(H)) \subseteq \overline{w}(A)$.

Thus we defined language models (or L-models) for the hypergraph Lambek calculus. Now we formulate some standard model-theoretic results (their proof in the hypergraph case does not differ from that in the string case).

Theorem 2. *If* $\text{HL} \vdash H \to A$*, then* $\overline{w}(H \to A)$ *is true for each valuation w.*

This theorem is proved by a straightforward induction on the length of a derivation. The other direction (i.e. completeness) is an open question (in the string case, this direction was a hard open problem until it was proved in [9]). We expect that it holds in the hypergraph case but we have no idea how to generalize the proof from [9]. However, if we consider the product-free fragment of HL (that is, we will consider types with $\div$ only), then the completeness theorem can be easily proved using the canonical model. In the string case, the canonical model is the following: given a type A, we assign the set of antecedents Π such that $\Pi \to A$ is derivable to A considering types as symbols of an (infinite) alphabet.

Theorem 3. *If* $\overline{w}(H \to A)$ *is true for each valuation w, and types in $H \to A$ do not contain $\times$, then* $\text{HL} \vdash H \to A$*.*

Proof. Let us denote the fragment of HL, in which we consider only types without $\times$, as $\mathrm{HL}(\div)$. We fix the alphabet $\Sigma = Tp(\mathrm{HL}(\div))$ (i.e. types without $\times$ are now symbols of the alphabet) and introduce a valuation w_0 for all primitive types: $w_0(p) = \{G \in \mathscr{H}(\Sigma) \mid \mathrm{HL} \vdash G \to p\}$. Note that such a definition of w_0 can be considered not only for primitive types p but for all types $T \in Tp(\mathrm{HL}(\div))$. We claim that $\overline{w_0}(T) = w_0(T)$ for all such types; that is, the function $\overline{w_0}$ obtained from Definition 10 coincides with w_0. Indeed: if $T = N \div D$, $lab_D(d_0) = \$$, and $lab_D(d_i) = T_i$ $(i = 1, \ldots, k)$, then

$$G \in w_0(N \div D) \Leftrightarrow \mathrm{HL} \vdash G \to N \div D \Leftrightarrow \mathrm{HL} \vdash D[d_0/G] \to N \text{ (see Proposition 1)} \Leftrightarrow$$
$$\Leftrightarrow \forall H_i : (\mathrm{HL} \vdash H_i \to T_i, \ i = 1, \ldots, k) \quad \mathrm{HL} \vdash D[d_0/G, d_1/H_1, \ldots, d_k/H_k] \to N \Leftrightarrow G \in \overline{w_0}(N \div D).$$

The penultimate equivalence follows from the fact that $\mathrm{HL} \vdash T_i^\bullet \to T_i$ and from the cut elimination Theorem 1. Since $\overline{w_0}(H \to A)$ is true, $w_0(\times(H)) \subseteq w_0(A)$; H belongs to $w_0(\times(H))$ ($\mathrm{HL} \vdash H \to \times(H)$), hence H belongs to $w_0(A)$. By the definition of w_0, this yields that $\mathrm{HL} \vdash H \to A$. □

Unfortunately, a similar proof does not work for HL with $\times$ (like in the string case). Thus, there is much space for further investigations. Nevertheless, Theorem 2 and Theorem 3 partially justify the periphrastic name of the hypergraph Lambek calculus given in the title: it is a logic of hypergraph languages.

5 Hypergraph Lambek Grammars and Their Power

Although in this paper we devote a great deal of attention to the hypergraph Lambek calculus itself and to its model-theoretic motivation, our main goal is to study the concept of *hypergraph Lambek grammars* (HL-grammars in short). They are defined in a similar way to Lambek grammars (in the string case). A grammar is essentially a finite set of correspondences of the form $a \triangleright T$ where a is a terminal label, and T is a type; besides, in a grammar some type S (not necessarily primitive) is distinguished. Then a hypergraph G belongs to the language generated by the grammar if we can replace labels of its hyperedges by corresponding types (let us denote the resulting graph G') and derive the sequent $G' \to S$ in HL.

We consider a ranked alphabet Σ.

Definition 11. A *hypergraph Lambek grammar (HL-grammar)* is a tuple $HGr = \langle \Sigma, S, \triangleright \rangle$ where Σ is a finite set (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 $rank(a) = rank(T)$.

Definition 12. *The type set of an HL-grammar* $HGr = \langle \Sigma, S, \triangleright \rangle$ is the set $\mathrm{ts}(HGr) = \{T \mid \exists a \in \Sigma : a \triangleright T\}$.

Definition 13. *The language $L(HGr)$ generated by a hypergraph Lambek grammar* $HGr = \langle \Sigma, S, \triangleright \rangle$ is the set of all hypergraphs $G \in \mathscr{H}(\Sigma)$ 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 11. The HL-grammar $SGr = \langle \{a, b\}, s, \triangleright \rangle$ where s is primitive, and

- $a \triangleright s \div \mathrm{SG}(\$sp) = Q$,

- $b \triangleright p, \ b \triangleright s$

generates the language $\{\mathrm{SG}(a^n b^{n+1}) \mid n \geq 0\}$. For example, if one wants to check that $G = \mathrm{SG}(aabbb)$ belongs to $L(SGr)$, he/she follows such steps:

1. Relabel each edge in G in such a way that each label is replaced by a type corresponding to it. We do this as follows: $G = \mathrm{SG}(aabbb) \rightsquigarrow f_G(G) = \mathrm{SG}(QQspp)$.

2. Consider the sequent $f_G(G) \to s$ and derive it in HL:

$$
\cfrac{\cfrac{s^\bullet \to s \quad s^\bullet \to s \quad p^\bullet \to p}{\mathrm{SG}(Qsp) \to s}\ (\div \to) \quad s^\bullet \to s \quad p^\bullet \to p}{\mathrm{SG}(QQspp) \to s}\ (\div \to)
$$

Now, notice the following: if a sequent $G' \to s$ is derivable, and G' is labeled only by types from the type set of SGr, then each derivation of $G' \to s$ consists only of applications of $(\div \to)$. The rule $(\div \to)$ consists of several replacements in the antecedent of a sequent, and hence the grammar SGr works in a way similar to the hyperedge replacement grammar with the following set of productions:

$$
S \to \mathrm{SG}(aSP) \qquad S \to \mathrm{SG}(b) \qquad P \to \mathrm{SG}(b)
$$

Note that the conversion of this HRG back into SGr can be made according to the principles explained in Section 3. The new grammar is actually a graph variant of a context-free grammar with the productions $S \to aSP, S \to b, P \to b$, which, clearly, generates the language $\{a^n b^{n+1} \mid n \geq 0\}$.

The transformation considered in Example 4 and in Example 11 is possible, if there is exactly one terminal label (say a) in a production; then we place $\$$ instead of a, and establish a correspondence $\triangleright$ between a and a type made on the basis of this production.

Definition 14. An HRG HGr is in *the weak Greibach normal form* if there is exactly one terminal edge in the right-hand side of each production.

Denote by $isize(H)$ the number of isolated nodes in H.

Definition 15. A hypergraph language L is isolated-node bounded if there is a constant $M > 0$ such that for each $H \in L$ $isize(H) < M \cdot |E_H|$.

In [11] we prove the following

Theorem 4. *For each HRG generating an isolated-node bounded language there is an equivalent HRG in the weak Greibach normal form.*

Using it, we can prove the following theorem applying standard techniques.

Theorem 5. *For each HRG generating an isolated-node bounded language there is an equivalent HL-grammar.*

The proof of this theorem can be found in [13]. It is not, however, of interest in this paper; we formulate this theorem here only to show the reader that HL-grammars are not weaker than HRGs (isolated-node boundedness is a nonsubstantial limitation). Our objective now is to show that HL-grammars are more powerful than HRGs; to do this, we will introduce several examples of grammars generating languages that can be generated by no HRGs.

5.1 All Binary Graphs

One of restrictions known for languages generated by HRGs is that they are of bounded connectivity (see [3]); this follows from the pumping lemma (see [5], Chapter IV.2). Consequently, no HRG can generate the set of all binary graphs (i.e. of usual graphs with edges of rank 2). This might seem unnatural because

HRGs represent a context-free formalism, and the language of all binary graphs seems to be very simple and regular. Below we show that HL-grammars are powerful enough to generate such a language.

Consider the language $\mathcal{L}_1$ of all binary graphs without isolated nodes (the empty graph is not included in $\mathcal{L}_1$ as well) over the alphabet $\{*\}$ ($rank(*) = 2$) that are, besides, without external nodes. Consequently, each graph in this language has at least two nodes. Let s, p be primitive types ($rank(s) = 0$, $rank(p) = 1$). Let us define the following types:

$$Q_1 = p, \quad Q_2 = p \div \left(\begin{array}{cc} \boxed{\$} & \boxed{p} \\ {}_1 \Big| & \Big|{}_1 \\ \bullet & \bullet \\ & (1) \end{array} \right), \quad Q_3 = s \div \left(\begin{array}{cc} \boxed{\$} & \boxed{p} \\ {}_1 \Big| & \Big|{}_1 \\ \bullet & \bullet \end{array} \right);$$

$$M_{11}^{ij} = \times \left(\begin{array}{cc} \boxed{Q_i} & \boxed{Q_j} \\ {}_1 \Big| & \Big|{}_1 \\ \bullet & \bullet \\ (1) & (2) \end{array} \right), \; M_{12}^i = \times \left(\begin{array}{cc} \boxed{Q_i} & \\ {}_1 \Big| & \\ \bullet & \bullet \\ (1) & (2) \end{array} \right), \; M_{21}^j = \times \left(\begin{array}{cc} & \boxed{Q_j} \\ & \Big|{}_1 \\ \bullet & \bullet \\ (1) & (2) \end{array} \right), \; M_{22} = \times \left(\begin{array}{cc} \bullet & \bullet \\ (1) & (2) \end{array} \right).$$
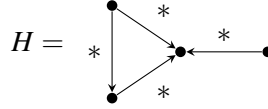
Consider a grammar $HGr_1 = \langle \{*\}, s, \rhd \rangle$ where $* \rhd N$ whenever $N \in \{M_{11}^{ij}, M_{12}^i, M_{21}^j, M_{22} | 1 \leq i, j \leq 3\}$.
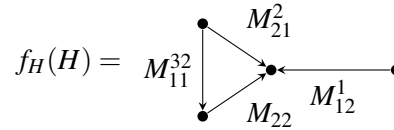
Theorem 6. $L(HGr_1) = \mathcal{L}_1$.

Proof. To prove that $L(HGr_1) \subseteq \mathcal{L}_1$ it suffices to note that denominators of types in ts(HGr_1) do not contain isolated nodes; since isolated nodes may appear only after applications of rules $(\div \rightarrow)$ or $(\rightarrow \times)$, all graphs in $L(HGr_1)$ do not have them.

The other inclusion $L(HGr_1) \supseteq \mathcal{L}_1$ is of central interest. We start with an example of a specific derivation in this grammar. After, we provide the proof in a general case, but we suppose that this example is enough to understand the construction of HGr_1.

Example 12. Consider a binary graph

$$H = \quad \begin{array}{c} \bullet \\ {}_* \diagup \; \Big\uparrow \; {}_* \quad {}_* \\ \bullet \!\!\rightarrow\!\! \bullet \!\!\leftarrow\!\! \bullet \\ {}_* \diagdown \; \Big\downarrow \; {}_* \\ \bullet \end{array}$$

In order to check that H belongs to $L(HGr_1)$ we relabel it by corresponding types as follows:

$$f_H(H) = \quad M_{11}^{32} \begin{array}{c} \bullet \\ \Big\uparrow \quad M_{21}^2 \\ \bullet \!\!\rightarrow\!\! \bullet \!\!\leftarrow\!\! \bullet \\ M_{22} \quad M_{12}^1 \\ \bullet \end{array}$$

Then we check derivability of $f_H(H) \rightarrow s$ (see Figure 1). The idea behind types $M_{ij}^{\cdot\cdot}$ is the following: each edge is replaced in the derivation (considered from bottom to top) by a pair of hyperedges of rank 1 attached to nodes (labeled by Q_i). Each node is intended to be attached to exactly one hyperedge; then we need to label exactly one hyperedge by Q_1 (from which rule applications of $(\div \rightarrow)$ should start), exactly one hyperedge by Q_3, and the remaining ones by Q_2.

In general, let H be in $\mathcal{L}_1$. Since there are no isolated nodes in H there exists a function $h : V_H \rightarrow E_H$ such that $h(v)$ is attached to v whenever $v \in V_H$. We choose two arbitrary nodes v_b (begin) and v_e (end) such that $v_b \neq v_e$. After that, we define a function $c : V_H \rightarrow \{1, 2, 3\}$ as follows: $c(v_b) = 1$, $c(v_e) = 3$, and $c(v) = 2$ whenever $v \notin \{v_b, v_e\}$.

Now we present a relabeling $f_H : E_H \rightarrow Tp(\text{HL})$. Let e belong to E_H and let $att_H(e) = v_1 v_2$.
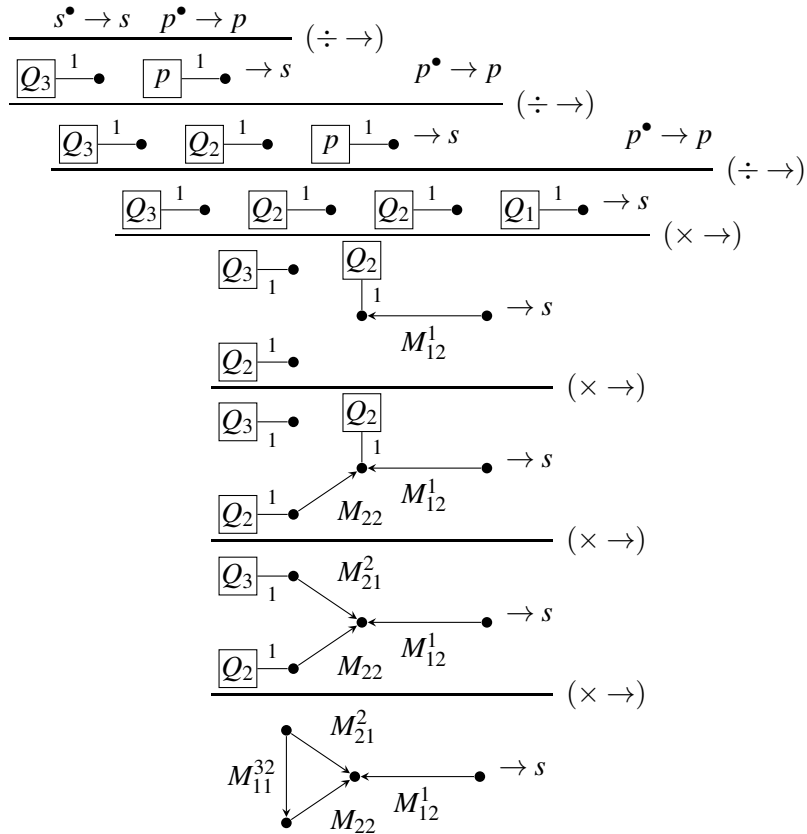
Figure 1: Derivation of a sequent corresponding to the binary graph H.

- If $h(v_1) = h(v_2) = e$, then $f_H(e) := M_{11}^{c(v_1)c(v_2)}$;   • If $h(v_1) \neq e, h(v_2) = e$, then $f_H(e) := M_{21}^{c(v_2)}$;

- If $h(v_1) = e, h(v_2) \neq e$, then $f_H(e) := M_{12}^{c(v_1)}$;   • If $h(v_1) \neq e, h(v_2) \neq e$, then $f_H(e) := M_{22}$.

We aim to check derivability of the sequent $f_H(H) \to s$. Its derivation from bottom to top starts with the rule $(\times \to)$ applied $|E_H|$ times to all types in the antecedent. It turns out that, after such applications of $(\times \to)$, the antecedent of a sequent includes one edge labeled by Q_1, one edge labeled by Q_3, and the remaining edges labeled by Q_2; besides, for each node there is exactly one edge attached to it (this is satisfied by the definition of the function h). Then we apply (again from bottom to top) the rule $(\div \to)$, and using it we "reduce" the only Q_1-labeled edge (recall that $Q_1 = p$) with a Q_2-labeled edge; after this we obtain a new p-labeled edge and repeat the procedure. Thus we eliminate all nodes and edges one-by-one. Finally, we obtain a graph with two nodes, with a Q_3-labeled edge attached to the first one and a p-labeled edge attached to the second one. Applying $(\div \to)$ once more, we "contract" Q_3 with p and obtain the sequent $s^\bullet \to s$, which is an axiom. $\square$

Therefore, we have established that HL-grammars are stronger than HRGs and that they moreover disobey the pumping lemma introduced in [2].

5.2 Bipartite graphs

Another example is the language $\mathscr{L}_2 \subseteq \mathscr{L}_1$ of all bipartite binary graphs without isolated nodes.

Definition 16. A binary graph H is *bipartite* if its nodes can be divided into two disjoint subsets V_1 and V_2 in such a way that each edge of H outgoes from a node belonging to V_1 to a node belonging to V_2.

Let us define the following types (where p, q are primitive, $rank(p) = rank(q) = 1$):

- $R_1(r) := r$;

- $R_2(r) := r \div \left(\begin{array}{c} \boxed{\$} \quad \boxed{r} \\ \swarrow \searrow \\ 1 \quad \bullet \quad 1 \\ (1) \end{array} \right)$;

- $R_3(r) := r \div \left(\begin{array}{cc} \boxed{\$} & \boxed{r} \\ 1| & 1| \\ \bullet_{(1)} & \bullet \end{array} \right)$;

- $R_4(r) := r \div \left(\begin{array}{c} \boxed{\$} \quad \boxed{r} \quad \boxed{r} \\ \swarrow \searrow \quad | 1 \\ 1 \bullet 1 \quad \bullet \\ (1) \end{array} \right)$;

- $M^{ij} := \times \left(\begin{array}{cc} \boxed{R_i(p)} & \boxed{R_j(q)} \\ 1| & 1| \\ \bullet_{(1)} & \bullet_{(2)} \end{array} \right)$, $1 \leq i, j \leq 4$;

- $S := \times \left(\begin{array}{cc} \boxed{p} & \boxed{q} \\ 1| & 1| \\ \bullet & \bullet \end{array} \right)$.

We define $HGr_2 := \langle \{*\}, S, \triangleright \rangle$ as follows: $* \triangleright M^{ij}$ for all $1 \leq i, j \leq 4$.

Proposition 2. $\mathscr{L}_2 = L(HGr_2)$.

The proof of this proposition is divided into two parts. It straightforward to show that $\mathscr{L}_2 \subseteq L(HGr_2)$ by deriving sequents corresponding to graphs from $\mathscr{L}_2$. To prove the other inclusion, we use Proposition 1 and then notice that there is no way for two hyperedges, one of which is labeled by $R_i(p)$ and the other one is labeled by $R_j(q)$, to be attached to the same node in the antecedent (to prove this, it suffices to analyze variants of how the rule $(\div \to)$ can be applied).

5.3 Regular graphs

A less trivial example of a hypergraph language generated by a HL-grammar and by no HRGs is the language of regular binary graphs.

Definition 17. A binary graph H is *regular* if there is an integer $k \geq 1$ such that the indegree and the outdegree of each node equals k.

Let $\mathscr{L}_3 \subseteq \mathscr{L}_1$ be the language of all regular binary graphs (without the empty graph).

Theorem 7. $\mathscr{L}_3$ *can be generated by some HL-grammar.*

To prove this theorem, we need the following result proved in [13]:

Theorem 8. *If $L_1, \ldots, L_k$ are languages generated by some HRGs, then $L_1 \cap \ldots \cap L_k$ can be generated by some HL-grammar.*

Less formally, this means that HL-grammars can generate finite intersections of languages generated by HRGs.

Definition 18. Let $MS_1, \ldots, MS_n$, $n \geq 1$ be some multisets with elements from $C' \subseteq C$ (that is, they are multisets of labels from C'). Let $b \in C \setminus C'$ be some symbol with $rank(b) = 2$. We denote $MS_i = \{a_i^1, \ldots, a_i^{k_i}\}$, and $rank(a_i^j) = t_i^j > 0$. A *flowerbed* $\mathfrak{F}(MS_1, \ldots, MS_n, b)$ over C' is the hypergraph $\langle V, E, att, lab, ext \rangle$ where

1. $V = \{v_i^{jk} \mid i = 1, \ldots, n, \ j = 1, \ldots, k_i, \ k = 1, \ldots, t_i^j - 1\} \cup \{u_1, \ldots, u_n\}$;

2. $E = \{e_i^j \mid i = 1, \ldots, n, \ j = 1, \ldots, k_i\} \cup \{f_1, \ldots, f_{n-1}\}$;

3. (a) $att(e_i^j) = u_i v_i^{j1} v_i^{j2} \ldots v_i^{j(t_i^j - 1)}$ (if $t_i^j = 1$, then $att(e_i^j) = u_i$);
    (b) $att(f_k) = v_k v_{k+1}$;

4. (a) $lab(e_i^j) = a_i^j$;
    (b) $lab(f_k) = b$;

5. $ext = \Lambda$.

Informally, a flowerbed is a string graph $\mathrm{SG}(b \ldots b)$ (b repeated n times) without external nodes such that several hyperedges of different ranks can be additionally attached to its nodes (but only the first attachment node of a hyperedge belongs to this string graph; the remaining nodes must be attached only to this hyperedge).

Definition 19. Given a multiset MS, $|MS|^a$ denotes the number of occurences of a in MS.

Proof (of Theorem 7). Let $C' = \{a, z\}$ with $rank(a) = 1$, $rank(z) = 3$. We set $\Sigma = \{a, z, b\}$ with $rank(b) = 2$. Let us introduce the following languages:

- L_1 is the set of all flowerbeds over C' of the form $\mathfrak{F}(MS_1, \ldots, MS_n, b)$ such that $|MS_{2k}|^a = = |MS_{2k+1}|^a = |MS_{2k}|^z = |MS_{2k+1}|^z$ for $k = 1, \ldots, \lfloor \frac{n-1}{2} \rfloor$

- L_2 is the set of all flowerbeds over C' of the form $\mathfrak{F}(MS_1, \ldots, MS_n, b)$ such that $|MS_{2k-1}|^a = |MS_{2k}|^a = |MS_{2k-1}|^z = |MS_{2k}|^z$ for $k = 1, \ldots, \lfloor \frac{n}{2} \rfloor$

It is left as an exercise to prove that L_1 and L_2 can be generated by some HRGs; thus, according to Theorem 8, $L = L_1 \cap L_2$ can be generated by an HL-grammar. Let us denote such a grammar $HGr = \langle \Sigma, S, \triangleright \rangle$: $L(HGr) = L$. Note that L is the set of all flowerbeds over C' of the form $\mathfrak{F}(MS_1, \ldots, MS_n, b)$ such that $|MS_k|^a = |MS_{k+1}|^a = |MS_k|^z = |MS_{k+1}|^z$ for $k = 1, 2, \ldots, n-1$.

Let us denote all types corresponding to a via $\triangleright$ as A_i (i.e. $a \triangleright A_i$), all types corresponding to z as Z_j, and all types corresponding to b as B_k. Let

$$
T_{ij} := \times \left(\begin{array}{cc} \boxed{A_i} & \boxed{Z_j} \\ {}_1\!\!\downarrow & {}_1\!\!\downarrow \\ \bullet & \bullet \\ {}_{(1)} & {}_{(2)} \end{array} \right), \qquad
T_{ijk} := \times \left(\begin{array}{cc} \boxed{A_i} \quad\quad \boxed{Z_j} \\ {}_1\!\bullet \xrightarrow{\;\;B_k\;\;} {}_1\!\bullet \\ {}_{(1)} \quad\quad\quad {}_{(2)} \end{array} \right).
$$

Using these types we define an HL-grammar $\widetilde{HGr} := \langle \{*\}, S, \widetilde{\triangleright} \rangle$ as follows: $* \widetilde{\triangleright} T_{ij}, T_{ijk}$ for all possible i, j, k. We argue that $L(\widetilde{HGr}) = L_3$. Indeed, $H \in L(\widetilde{HGr})$ if and only if there exists a relabeling f_H such that $lab_H(e) \widetilde{\triangleright} f_H(e)$, and $\text{HL} \vdash f_H(H) \to S$. Labels in $f_H(H)$ are types T_{ij} and T_{ijk}, which are of the form $\times(M)$. Using Proposition 1, we draw the conclusion that $\text{HL} \vdash f_H(H) \to S$ if and only if $\text{HL} \vdash \hat{H} \to S$ where $\hat{H}$ is obtained from $f_H(H)$ by replacing each hyperedge labeled by a type of the form $\times(M)$ by M. $\hat{H}$ is labeled by types A_i, Z_j, and B_k. Note that for all i, j, k $A_i \ne Z_j$, $Z_j \ne B_k$, $A_i \ne B_k$ since $type(A_i) = 1$, $type(B_k) = 2$, $type(Z_j) = 3$. Let $g : E_{\hat{H}} \to \Sigma$ be such a function that $g(e) = a$ if $lab_{\hat{H}}(e) = A_i$, $g(e) = b$ if $lab_{\hat{H}}(e) = B_k$, and $g(e) = z$ if $lab_{\hat{H}}(e) = Z_j$. Since $\text{HL} \vdash \hat{H} \to S$, $g(\hat{H})$ belongs to $L(HGr) = L$. To complete the proof, observe that the number of a-labeled edges attached to a node in $g(\hat{H})$ equals the outdegree of this node in H, and the number of z-labeled edges attached to a node in $g(\hat{H})$ equals the indegree of this node in H; according to the definition of L, this number is the same for all nodes.

Formally, in the above reasonings we made a one-way transition when we introduced g; hence, we only proved that $\widetilde{HGr}$ generates regular binary graphs. However, given a regular binary graph H, we can construct graphs of the form $f_H(H)$, $\hat{H}$, and $g(\hat{H})$ corresponding to it and then repeat the above reasonings. $\square$

Remark. Consider the language L_3 of all flowerbeds over C' of the form $\mathfrak{F}(MS_1, \ldots, MS_n, b)$ such that $|MS_1|^a = n - 1$. This language can also be generated by some HRG (this is left as an exercise to the reader). If we defined L in the above proof as $L_1 \cap L_2 \cap L_3$, then $L(\widetilde{HGr})$ would consist of all regular binary graphs with n nodes such that the indegree and the outdegree of each node equals $n - 1$. Note that numbers of edges in graphs of $L(\widetilde{HGr})$ in such a case form the set $\left\{ \frac{n(n-1)}{2} \mid n \ge 2 \right\}$, which grows with the pace $O(n^2)$ (this violates the Linear-Growth theorem, see [5], Chapter IV.2).

6   Conclusion

Hypergraph Lambek grammars are a logical formalism, which extend hyperedge replacement grammars. They deal with hypergraph types and sequents, which have a model-theoretic semantics of hypergraph languages. We showed that HL-grammars are more powerful than HRGs; in particular, they violate the pumping lemma and the Linear-Growth theorem. Note that, since they generate the language of all graphs, they are able to generate languages of unbounded treewidth. This can be considered as a disadvantage since we cannot use algorithms for languages of bounded treewidth. However, our goal was rather to show that HL-grammars are much more powerful than HRGs, which cannot be obtained for free.

Note that the membership problem for HL-grammars is NP-complete: this follows from the fact that, if H belongs to $L(HGr)$ for $HGr = \langle \Sigma, S, \triangleright \rangle$, then this can be certified by a function f_H and by a derivation of $f_H(H) \to S$. Description of f_H and the derivation have polynomial size with respect to H and HGr, hence the problem is in NP. It is NP-complete since HL-grammars can generate an NP-complete language

(which can be generated by some HRG without isolated nodes). Therefore, being equal in complexity to HRGs, HL-grammars represent a promising instrument for generating hypergraph languages.

As is often the case, there are more questions than answers. Some of them are listed below:

1. Is it true that, if $\overline{w}(H \rightarrow A)$ is true for all valuations, then $\mathrm{HL} \vdash H \rightarrow A$ (for sequents that include types with $\times$)?

2. Do HL-grammars generate the language of (a) complete binary graphs? (b) planar binary graphs? (c) directed acyclic binary graphs?

3. What string languages can be generated by HL-grammars?

4. Is the class of languages generated by HL-grammars closed under intersections?

5. What nontrivial necessary properties (like the pumping lemma for HRGs) exist for languages generated by HL-grammars?

6. Can HL-grammars be embedded in some known kind of graph grammars?

We are interested in further and deeper study of generalizations of logical approaches and concepts to as graphs; we think that this allows one to better understand the nature of the considered notions.

Acknowledgments

References

[1] Daniel Bauer & Owen Rambow (2016): *Hyperedge Replacement and Nonprojective Dependency Structures*. In David Chiang & Alexander Koller, editors: *Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12), June 29 - July 1, 2016, Heinrich Heine University, Düsseldorf, Germany*, The Association for Computer Linguistics, pp. 103–111. Available at `https://www.aclweb.org/anthology/W16-3311/`.

[2] Frank Drewes & Anna Jonsson (2017): *Contextual Hyperedge Replacement Grammars for Abstract Meaning Representations*. In Marco Kuhlmann & Tatjana Scheffler, editors: *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms, TAG 2017, Umeå, Sweden, September 4-6, 2017*, Association for Computational Linguistics, pp. 102–111. Available at `https://www.aclweb.org/anthology/W17-6211/`.

[3] Frank Drewes, Hans-Jörg Kreowski & Annegret Habel (1997): *Hyperedge Replacement Graph Grammars*. In Grzegorz Rozenberg, editor: *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, World Scientific, pp. 95–162, doi:10.1142/9789812384720_0002.

[4] Sorcha Gilroy, Adam Lopez & Sebastian Maneth (2017): *Parsing Graphs with Regular Graph Grammars*. In Nancy Ide, Aurélie Herbelot & Lluís Màrquez, editors: *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics, *SEM @ACM 2017, Vancouver, Canada, August 3-4, 2017*, Association for Computational Linguistics, pp. 199–208, doi:10.18653/v1/S17-1024.

[5] Annegret Habel (1992): *Hyperedge Replacement: Grammars and Languages*. *Lecture Notes in Computer Science* 643, Springer, doi:10.1007/BFb0013875.

[6] Bevan K. Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann & Kevin Knight (2012): *Semantics-Based Machine Translation with Hyperedge Replacement Grammars*. In Martin Kay & Christian Boitet, editors: *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the*

Conference: Technical Papers, 8-15 December 2012, Mumbai, India, Indian Institute of Technology Bombay, pp. 1359–1376. Available at `https://www.aclweb.org/anthology/C12-1083/`.

[7] Joachim Lambek (1958):   *The Mathematics of Sentence Structure*.   *The American Mathematical Monthly* 65(3),   pp. 154–170,   doi:10.1080/00029890.1958.11989160.   Available at `http://www.jstor.org/stable/2310058`.

[8] Mati Pentus (1993): *Lambek Grammars Are Context Free*. In: *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, IEEE Computer Society, pp. 429–433, doi:10.1109/LICS.1993.287565.

[9] Mati Pentus (1995): *Models for the Lambek Calculus*.   *Ann. Pure Appl. Log.* 75(1-2), pp. 179–213, doi:10.1016/0168-0072(94)00063-9.

[10] Tikhon Pshenitsyn (2020): *Hypergraph Basic Categorial Grammars*. In Fabio Gadducci & Timo Kehrer, editors: *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Online, June 25-26, 2020, Proceedings, Lecture Notes in Computer Science* 12150, Springer, pp. 146–162, doi:10.1007/978-3-030-51372-6_9.

[11] Tikhon Pshenitsyn (2020): *Weak Greibach Normal Form for Hyperedge Replacement Grammars*.   In Berthold Hoffmann & Mark Minas, editors: *Proceedings of the Eleventh International Workshop on Graph Computation Models, GCM@STAF 2020, Online-Workshop, 24th June 2020, EPTCS* 330, pp. 108–125, doi:10.4204/EPTCS.330.7.

[12] Tikhon Pshenitsyn (2021): *Introduction to a Hypergraph Logic Unifying Different Variants of the Lambek Calculus*. Available at `https://arxiv.org/abs/2103.01199`.

[13] Tikhon Pshenitsyn (2021): *Powerful and NP-Complete: Hypergraph Lambek Grammars*. In Fabio Gadducci & Timo Kehrer, editors: *Graph Transformation - 14th International Conference, ICGT 2021, Held as Part of STAF 2021, Virtual Event, June 24-25, 2021, Proceedings, Lecture Notes in Computer Science* 12741, Springer, pp. 102–121, doi:10.1007/978-3-030-78946-6_6.