Hypergraph Basic Categorial Grammars

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
tpshenitsyn@lpcs.math.msu.su

Abstract. This work is an attempt to generalize categorial grammars, which deal with string languages, to hypergraphs. We consider a particular approach called basic categorial grammar (BCG) and introduce its natural extension to hypergraphs — hypergraph basic categorial grammar (HBCG). We show that BCGs can be naturally embedded in HBCGs. It turns out that, as BCGs are equivalent to context-free grammars, HBCGs are equivalent to hyperedge replacement grammars in generalized Greibach normal form. We also present several structural properties of HBCGs. From practical point of view, we show that HBCGs can be used to describe semantics of sentences of natural languages. We incorporate the lambda semantics into the new mechanism in the same way as it is done for BCGs and show that such an embedding allows one to describe semantics of sentences with cross-serial dependencies.

1 Introduction

Formal mechanisms serving to describe formal (string) languages include two large classes: generative grammars and categorial grammars. The former generate strings using rewriting rules (productions): a string is correct if it can be produced by a grammar. The most well-known example of such a formalism is context-free grammar (CFG). Categorial grammars, in opposite, take the whole string at first and then check whether it is correct as follows: there is a set of types and a uniform mechanism which defines what sequences of types are correct; a particular grammar contains a lexicon, i.e. a correspondence between symbols in an alphabet and types of the system. In order to check whether a string \( a_1 \ldots a_n \) is correct one chooses types \( T_1, \ldots, T_n \) such that \( a_i \) corresponds to \( T_i \) in the grammar and then checks if \( T_1, \ldots, T_n \) is correct with respect to uniform rules of the formalism.

One of the most fundamental examples of categorial grammars is basic categorial grammar (BCG). It is introduced in works of Ajdukiewicz [2] and Bar-Hillel [3]. Types in BCGs are built of primitive types \( Pr \) using left and right divisions \( \backslash, / \). There are two uniform rules of interaction between types: given \( A, (A \backslash B) \) or \( (B / A) \), \( A \) standing nearby each other within a sequence of types

The study was funded by RFBR, project number 20-01-00670.

© Springer Nature Switzerland AG 2020
F. Gadducci and T. Kehrer (Eds.): ICGT 2020, LNCS 12150, pp. 146–162, 2020.
https://doi.org/10.1007/978-3-030-51372-6_9
one can replace them by \( B \). The sequence of types is said to be correct iff it can be reduced to some distinguished \( s \in Pr \). It is proved by Gaifman [4] that this approach has the same generating power as context-free grammars.

BCGs can serve to describe natural languages. E.g. the sentence Tim thinks Helen is smart corresponds to the sequence of types \( NP, (NP\backslash S)/S, NP, (NP\backslash S)/ADJ, ADJ \), which can be reduced to \( S \); thus this sentence is grammatically correct\(^1\). Moreover, it is possible to combine BCGs with the \( \lambda \)-calculus and to model semantics of this sentence. Namely, if the \( \lambda \)-term \( \lambda x.\text{smart}(x) \) is assigned to the adjective \( \text{smart} \), the \( \lambda \)-term \( \lambda s.\lambda x.\text{think}(s)(x) \) is assigned to the verb \( \text{thinks} \), and the \( \lambda \)-term \( \lambda f.\lambda x.f(x) \) is assigned to \( \text{is} \), then the reductions that are done in order to obtain \( S \) from the sequence above can be treated as applications in the \( \lambda \)-calculus; hence, the sentence above is described by the meaning \( \text{think} (\text{smart}(\text{Helen}))(\text{Tim}) \).

Let us return to generative grammars. The principles underlying them can be extended to graphs; a class of resulting formalisms is called graph grammars. In this paper we focus on a particular approach to generating graphs named hyperedge replacement grammar (HRG in short). An overview on HRGs can be found in [9]. We are interested in HRG because it is closely related to CFG: definitions of these formalisms are similar to each other; consequently, they share many crucial properties, e.g. the pumping lemma and the fixed-point theorem. Moreover, HRGs represent a natural extension of CFGs, since strings can be represented by string graphs and CFGs can be modeled using HRGs.

The question we are going to discuss in this paper is how to generalize basic categorial grammars to hypergraphs and to obtain a categorial mechanism related to HRGs. We present such a generalization — hypergraph basic categorial grammars. We extend notions of types, of reduction laws, and of semantics to hypergraphs. As expected, the resulting mechanism is closely related both to BCGs and HRGs, which is shown in Sects. 5 and 6. In Sect. 7 several structural properties of HBCGs are studied. In Sect. 8 we show how to enrich our mechanism with the lambda semantics. In Sect. 9 we show an application of our theory to linguistics.

2 Basic Categorial Grammars (for Strings)

The survey of categorial grammars including basic categorial grammars can be found in [8]. Here we introduce the main definitions to show connections with the new formalism.

Let us fix a countable set \( Pr = \{p_i\}_{i=1}^{\infty} \) of primitive types.

**Definition 2.1.** The set \( Tp \) of types is defined inductively as follows: it is the least set such that \( Pr \subseteq Tp \) and for each \( A, B \in Tp \) \( B \backslash A, A/B \) are also in \( Tp \).

\(^1\) In the types considered, \( NP \) stands for noun phrases, \( ADJ \) stands for adjectives, and \( S \) stands for sentences.
Throughout this paper small letters $p, q, \ldots$ and strings composed of them (e.g. $np, cp$) range over primitive types. Capital letters $A, B, \ldots$ usually range over types (however, graphs are often referred to as $G$ and $H$).

There are two rules of BCGs:

1. $\Gamma, A, A \backslash B, \Delta \mapsto \Gamma, B, \Delta$
2. $\Gamma, B / A, A, \Delta \mapsto \Gamma, B, \Delta$

Here $\Gamma, \Delta$ are finite (possibly empty) sequences of types. Thus $\mapsto$ is a relation on $Tp^+ \times Tp^+$. We denote by $\overset{*}{\mapsto}$ its reflexive transitive closure. $\Gamma \overset{k}{\mapsto} \Delta$ denotes that $\Delta$ is obtained from $\Gamma$ in $k$ steps (the same notation is used for all the relations in this work).

**Definition 2.2.** A basic categorial grammar is a tuple $Gr = (\Sigma, s, \triangleright)$ where $\Sigma$ is a finite set (alphabet), $s$ is a distinguished primitive type, and $\triangleright \subseteq \Sigma \times Tp$ is a finite binary relation, i.e. it assigns a finite number of types to each symbol in the alphabet.

The language $L(Gr)$ generated by $Gr$ is the set of all strings $a_1 \ldots a_n$ for which there are types $T_1, \ldots, T_n$ such that $a_i \triangleright T_i$, and $T_1, \ldots, T_n \overset{*}{\mapsto} s$.

## 3 Hypergraphs and Operations on Them

This section is concerned with definitions related to hypergraphs. All the notions except for compression are well known and widely accepted (see [9]). Note that we use a slightly different notation from that in [9].

### 3.1 Hypergraphs, Sub-hypergraphs

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

Let $C$ be some fixed set of labels for whom the function $\text{type} : C \rightarrow \mathbb{N}$ is considered.

**Definition 3.1.** A hypergraph $G$ over $C$ is a tuple $G = (V, E, att, lab, ext)$ where $V$ is the set of nodes, $E$ is the set of hyperedges, $att : E \rightarrow V^{\otimes}$ assigns a string (i.e. an ordered set) of attachment nodes to each edge, $lab : E \rightarrow C$ labels each edge by some element of $C$ in such a way that $\text{type}(\text{lab}(e)) = |\text{att}(e)|$ whenever $e \in E$, and $\text{ext} \in V^{\otimes}$ 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 simply called graphs, and hyperedges are simply called edges. The set of all graphs with labels from $C$ is denoted by $\mathcal{H}(C)$. In drawings of graphs black dots correspond to nodes, labeled squares correspond to edges, $\text{att}$ is represented with numbered lines, and external nodes are depicted by numbers in brackets. If an edge has exactly two attachment nodes, it can be denoted by an arrow (which goes from the first attachment node to the second one).

**Definition 3.2.** The function $\text{type}$ (or $\text{type}_G$ to be exact) returns the number of nodes attached to some edge in a graph $G$: $\text{type}_G(e) := |\text{att}_G(e)|$. If $G$ is a graph, then $\text{type}(G) := |\text{ext}_G|$.

**Example 3.1.** There are two hypergraphs on Fig. 1, to the left and right of the symbol $\Rightarrow$. E.g., the graph on the left has 5 edges; there are three edges within it for which $\text{type}$ equals 2, one edge with $\text{type}$ equal to 1, and an edge with $\text{type}$ equal to 3.

**Definition 3.3.** 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 $e \in E_H$ $\text{att}_H(e) = \text{att}_G(e)$, $\text{lab}_H(e) = \text{lab}_G(e)$.

**Definition 3.4.** If $H = \langle \{v_i\}_{i=1}^n, \{e_0\}, \text{att}, \text{lab}, v_1 \ldots v_n \rangle$, $\text{att}(e_0) = v_1 \ldots v_n$ and $\text{lab}(e_0) = a$, then $H$ is called a handle. It is denoted by $\odot(a)$.

**Definition 3.5.** Let $H \in \mathcal{H}(C)$ be a graph, and let $f : E_H \rightarrow C$ be a relabeling function. Then $f(H) = \langle V_H, E_H, \text{att}_H, \text{lab}_H, \text{ext}_H \rangle$ where $\text{lab}_f(H)(e) = f(e)$ for all $e \in E_H$. It is required that $\text{type}(\text{lab}_H(e)) = \text{type}(f(e))$ for $e \in E_H$.

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

**Definition 3.6.** An isomorphism between graphs $G$ and $H$ is a pair of bijective functions $\mathcal{E} : E_G \rightarrow E_H$, $\mathcal{B} : V_G \rightarrow V_H$ such that $\text{att}_H \circ \mathcal{E} = \mathcal{B} \circ \text{att}_G$, $\text{lab}_G = \text{lab}_H \circ \mathcal{E}$, $\mathcal{B}(\text{ext}_G) = \text{ext}_H$. In this work, we do not distinguish between isomorphic graphs.

### 3.2 Operations on Graphs

In graph formalisms certain graph transformation are in use. To generalize categorial grammars we present the following operation called compression.

**Compression.** Let $G$ be a graph, and let $H$ be a subgraph of $G$. Compression of $H$ into an $a$-labeled edge within $G$ is a procedure of transformation of $G$, which can be done under the following conditions:

(a) For each $v \in V_H$, if $v$ is attached to some edge $e \in E_G \setminus E_H$ (i.e. $v \in [\text{att}(e)]$), then $v$ has to be external in $H$ ($v \in [\text{ext}_H]$).
(b) If \( v \in V_H \) is external in \( G \), then it is external in \( H \) \( ([\text{ext}_G] \cap V_H \subseteq [\text{ext}_H]) \).
(c) \( \text{type}(H) = \text{type}(a) \).

Then the procedure is the following:
1. Remove all nodes of \( V_H \) except for those of \( \text{ext}_H \) from \( V_G \);
2. Remove \( E_H \) from \( E_G \);
3. Add a new edge \( \tilde{e} \);
4. Set \( \text{att}(\tilde{e}) = \text{ext}_H, \text{lab}(\tilde{e}) = a \).

Let \( G[a/H] \) (or \( G[a, \tilde{e}/H] \)) denote the resulting graph. Formally, \( G[a/H] = \langle V', E', \text{att}', \text{lab}', \text{ext}_G \rangle \), where \( V' = V_G \setminus (V_H \setminus \text{ext}_H) \), \( E' = \{\tilde{e}\} \cup (E_G \setminus E_H) \), \( \text{att}'(e) = \text{att}_G(e) \), \( \text{lab}'(e) = \text{lab}_G(e) \) for \( e \neq \tilde{e} \), and \( \text{att}'(\tilde{e}) = \text{ext}_H, \text{lab}'(\tilde{e}) = a \).

**Replacement.** This procedure is defined in [9]. In short, the replacement of an edge \( e_0 \) in \( G \) with a graph \( H \) can be done if \( \text{type}(e_0) = \text{type}(H) \) as follows:
1. Remove \( e_0 \);
2. Insert an isomorphic copy of \( H \) (namely, \( H \) 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 attachment node of \( e_0 \).

To be more precise, the set of edges in the resulting graph is \((E_G \setminus \{e_0\}) \cup E_H\), and the set of nodes is \( V_G \cup (V_H \setminus \text{ext}_H) \). The result is denoted by \( G[H/e_0] \).

**Proposition 3.1.** Compression and replacement are opposite:
1. \( G[a, e/H][H/e] \equiv G \) (for a subgraph \( H \) of \( G \) satisfying conditions (a) and (b); \( a \) is an arbitrary label);
2. \( G[H/e][a, e/H] \equiv G \) (provided \( e : \text{type}(e) = \text{type}(H) \) and \( a = \text{lab}_G(e) \)).

### 3.3 Hyperedge Replacement Grammars

**Definition 3.7.** A hyperedge replacement grammar is a tuple \( Gr = \langle N, \Sigma, P, S \rangle \), where \( N \) is a finite alphabet of nonterminal symbols, \( \Sigma \) is a finite alphabet of terminal symbols \( (N \cap \Sigma = \emptyset) \), \( P \) is a set of productions, and \( S \in N \). Each production is of the form \( A \rightarrow H \) where \( A \in N, H \in H(N \cup \Sigma) \) and \( \text{type}(A) = \text{type}(H) \).

Edges labeled by terminal (nonterminal) symbols are called terminal (nonterminal) edges.

If \( G \) is a graph, \( e_0 \in E_G \), \( \text{lab}(e_0) = A \) and \( A \rightarrow H \in P \), then \( G \) directly derives \( G[H/e_0] \) (denote \( G \Rightarrow G[H/e_0] \)). The transitive reflexive closure of \( \Rightarrow \) is denoted by \( \Rightarrow^\ast \). If \( G \Rightarrow^\ast H \), then \( G \) is said to derive \( H \). The corresponding sequence of production applications is called a derivation.

**Definition 3.8.** The language generated by an HRG \( \langle N, \Sigma, P, S \rangle \) is the set of graphs \( H \in H(\Sigma) \) such that \( \odot(S) \Rightarrow^\ast H \). Two grammars are said to be equivalent iff they generate the same language.

Further we simply write \( A \Rightarrow G \) instead of \( \odot(A) \Rightarrow G \).
4 Hypergraph Basic Categorial Grammars

In this section, we present definitions needed to extend BCGs to graphs. Firstly, we introduce the notion of a type; then we define a rewriting rule, which operates on graphs labeled by types; finally, we introduce the definitions of a hypergraph basic categorial grammar and of a language generated by it.

4.1 Types

We fix a countable set $Pr$ of primitive types and a function $\text{type} : Pr \to \mathbb{N}$ such that for each $n \in \mathbb{N}$ there are infinitely many $p \in Pr$ for which $\text{type}(p) = n$. Types are constructed from primitive types using division. Simultaneously, we define the function $\text{type}$ on types.

Let us fix some symbol $\$ $ that is not included in all the sets considered. NB! This symbol is allowed to label edges with different number of attachment nodes. To be consistent with Definition 3.1 one can assume that there are countably many symbols $\$ _n such that $\text{type}(\$ _n) = n$.

**Definition 4.1.** The set $Tp_\chi$ of types is the least set satisfying the following conditions:

1. $Pr \subseteq Tp_\chi$;
2. Let $N$ ("numerator") be in $Tp_\chi$. Let $D$ ("denominator") be a graph such that exactly one of its edges (call it $e_0$) is labeled by $\$$, and other edges (possibly, there are none of them) are labeled by elements of $Tp_\chi$; let also $\text{type}(N) = \text{type}(D)$. Then $T = \div(N/D)$ also belongs to $Tp_\chi$, and $\text{type}(T) := \text{type}_D(e_0)$.

In types, $\$$ serves to "connect" a denominator and a numerator.

**Example 4.1.** The following structure is a type:

$$E_0 = \div \left( \begin{array}{c} q \\ \$ \end{array} \right) \left( \begin{array}{c} 2 & 3 \\ 1 & \end{array} \right).$$

Here $p, q$ belong to $Pr$, $\text{type}(p) = 2$, $\text{type}(q) = 3$; $\text{type}(E_0) = 2$.

4.2 D-Isomorphism

In order to generalize the rules $A/B, B \rightarrow A$ and $B, B \backslash A \rightarrow A$, denominators of types are going to be "overlaid" on subgraphs of graphs. This idea is formalized by the notion of a d-isomorphism.
Definition 4.2. A graph $D$ that has exactly one edge labeled by $\$ \text{ while other ones are labeled by elements of } Tp_{\chi}$, is called d-formed. The only edge of $D$ labeled by $\$ \text{ is denoted } \text{head}(D)$.

Definition 4.3. A graph-edge pair is a pair $(H;e_0)$ where $H$ is a graph, and $e_0 \in E_H$.

Definition 4.4. A d-isomorphism between a d-formed graph $D$ and an graph-edge pair $(H;e_0)$ is a pair of functions $(\epsilon;\beta)$ such that

- $\epsilon : E_D \rightarrow E_H$, $\beta : V_D \rightarrow V_H$ are bijections;
- $\text{att}_H \circ \epsilon = \beta \circ \text{att}_D$;
- $\epsilon(\text{head}(D)) = e_0$;
- For all $e \in E_D \setminus \{\text{head}(D)\}$ $\text{lab}_D(e) = \text{lab}_H(\epsilon(e))$;
- $\beta(\text{ext}_D) = \text{ext}_H$.

4.3 Rule ($\div$)

The concept of hypergraph basic categorial grammars (HBCGs) is based on the mechanism of reduction of hypergraphs labeled by types. There is an inference rule, which is denoted by ($\div$), generalizing two rules for BCGs presented earlier.

The following dramatis personae participate in the rule ($\div$):

- $G \in \mathcal{H}(Tp_{\chi})$;
- $H$ — a subgraph of $G$, and $(H;e_0)$, which is a graph-edge pair;
- $\text{lab}(e_0) = \div(N/D)$;
- $(\epsilon;\beta)$ — a d-isomorphism between $D$ and $(H;e_0)$.

The rule ($\div$) can be applied to $(H;e_0)$ if the following conditions are fulfilled:

1. If $v \in [\text{att}(e)]$ for some $e \in E_G \setminus E_H$ and $v \in V_H$, then $v \in [\text{ext}_H]$;
2. $[\text{ext}_G] \cap V_H \subseteq [\text{ext}_H]$.

Then the rule is of the form

$G \xrightarrow{\chi} G[N/H \div \div] \hspace{1cm} (\div).$

Requirements 1 and 2 guarantee that compression is possible.

If $G \xrightarrow{\star \chi} H$, then $G$ is said to be reducible to $H$ (as usual, $\xrightarrow{\star \chi}$ is the reflexive transitive closure of $\xrightarrow{\chi}$).

![Diagram](image-url)

Fig. 1. Example of the application of ($\div$). Here $E_0$ is from Example 4.1. Note that the order of attachment node of the $q$-labeled edge is changed w.r.t. to $E_0$. 


4.4 Hypergraph Basic Categorial Grammar

**Definition 4.5.** A hypergraph basic categorial grammar (HBCG for short) \( Gr \) is a tuple \( Gr = \langle \Sigma, s, \triangleright \rangle \) where \( \Sigma \) is a finite alphabet, \( s \) is a primitive type, and \( \triangleright \subseteq \Sigma \times Tp_x \) is a binary relation (called a lexicon) which assigns a finite number of types to each symbol in the alphabet. Additionally, we require that the function \( \text{type} \) is defined on \( \Sigma \) such that \( a \triangleright T \) implies \( \text{type}(a) = \text{type}(T) \).

The set \( \text{dict}(Gr) = \{ T \in Tp_x | \exists a \in \Sigma : a \triangleright T \} \cup \{ s \} \) is called a dictionary.

**Definition 4.6.** The language \( L(Gr) \) generated by an HBCG \( Gr = \langle \Sigma, s, \triangleright \rangle \) is the set of all hypergraphs \( G \in \mathcal{H}(\Sigma) \) for which a function \( f_G : E_G \rightarrow Tp_x \) exists such that:

1. \( \text{lab}_G(e) \triangleright f_G(e) \) whenever \( e \in E_G \);
2. \( f_G(G) \xrightarrow{s} \triangleright \circ(s) \).

All the definitions presented above are slightly more complicated and technical than that of BCGs; however, the concept of HBCGs is closely related both to HRGs and BCGs.

**Example 4.2.** Let us consider an example of an HRG from [11] (a little bit modified) generating abstract meaning representations, which contains four rules:

\[
\begin{align*}
S &\rightarrow X \\
X &\rightarrow Y \\
Y &\rightarrow X \\
I &\rightarrow i \\
\end{align*}
\]

Here \( S \) is the initial symbol. This HRG can be converted into an equivalent HBCG as follows. Let \( s, x, y, i, a_0, a_1 \) be primitive types. Then the following lexicon defines an HBCG that generates the same language as the HRG above:

\[
\begin{align*}
\text{go} &\triangleright \div \left( \begin{array}{c}
\begin{array}{c}
\text{\$} \\
1 \\
a_0 \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
x \\
2 \\
i \\
\end{array}
\end{array}
\right); \text{want} \triangleright \div \left( \begin{array}{c}
\begin{array}{c}
\text{\$} \\
1 \\
a_0 \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
x \\
2 \\
\end{array}
\end{array}
\right); \text{arg}_0 \triangleright \div a_0; \\
\text{need} \triangleright \div \left( \begin{array}{c}
\begin{array}{c}
\text{\$} \\
1 \\
a_1 \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
x \\
2 \\
\end{array}
\end{array}
\right); \text{need} \triangleright \div \left( \begin{array}{c}
\begin{array}{c}
\text{\$} \\
1 \\
a_1 \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
x \\
2 \\
\end{array}
\end{array}
\right); \text{arg}_1 \triangleright \div a_1; \\
I \triangleright i. \\
\end{align*}
\]
To be more precise, the HBCG is of the form \( \langle \{ \text{want, need, go, I, arg}_0, arg_1 \}, s, \triangleright \rangle \).
All the primitive types have type being equal to 2.

The conversion can be done since there is a terminal edge in the right-hand side of each production (see more in Sect. 6). A more thorough example of an HBCG is given in Sect. 9.

5 Embedding of BCG

To justify that HBCGs appropriately extend BCGs, we present an embedding of the latter into the former in a natural and a simple way.

A function \( tr : Tp \to Tp_\chi \) presented below embeds string types into graph types:

- \( tr(p) := p, p \in Pr, \text{type}(p) = 2; \)
- \( tr(A/B) := \frac{1}{A/B} \begin{array}{c}
(1) \bullet \to \bullet \to (2) \\
\end{array} \)
- \( tr(B \setminus A) := \frac{1}{B \setminus A} \begin{array}{c}
(1) \bullet \to \bullet \to (2) \\
\end{array} \)

Recall that a string graph induced by a word \( w = a_1 \ldots a_n \) is a graph of the form \( \langle \{ v_i \}_{i=0}^n, \{ e_i \}_{i=1}^n, \text{att}, \text{lab}, v_0v_n \rangle \) where \( \text{att}(e_i) = v_{i-1}v_i, \text{lab}(e_i) = a_i \). This graph is denoted by \( w^*. \)

\( tr(Tp) \) denotes the set of translations of all types. If \( \Gamma = T_1, \ldots, T_n \) is a sequence of types, then \( tr(\Gamma) := (tr(T_1) \ldots tr(T_n))^*. \) If \( Gr = \langle \Sigma, s, \triangleright \rangle \) is a BCG, then \( tr(Gr) \) is the HBCG \( \langle \Sigma, s, \triangleright_\chi \rangle \) where \( a \triangleright_\chi T \Leftrightarrow T = tr(A) \) and \( a \triangleright A \).

Two propositions below establish connection between BCGs and HBCGs.

**Proposition 5.1.** If \( G \in \mathcal{H}(tr(Tp)), C \in Tp \) and \( G \xrightarrow{\chi^*} (tr(C)) \), then \( G = tr(\Gamma) \) for some \( \Gamma \) such that \( \Gamma \xrightarrow{\chi^*} C \).

**Proof.** Proof by induction on the number of steps in the derivation \( G \xrightarrow{n} \chi (tr(C)) \).

**Basis.** If \( n = 0 \), then \( G = \chi (tr(C)) \) and \( C \xrightarrow{0} C \). **Step.** Let the first reduction be applied to \( (H; e_0) \) within \( G \). If \( lab(e_0) = tr(A/B) \), then \( H = \langle \{ v_0, v_1, v_2 \}, \{ e_0, e_1 \}, \text{att}, \text{lab}, v_0v_n \rangle \) where \( \text{att}(e_i) = v_{i-1}v_i, i = 0, 1 \) and \( \text{lab}(e_1) = tr(B) \). Since \( G[e, tr(A)/H] \xrightarrow{\chi^*} (tr(C)) \), by the induction hypothesis we obtain \( G[e, tr(A)/H] = tr(\Delta) \) such that \( \Delta = \Phi, A, \Psi \xrightarrow{\chi^*} C \). Then \( G = G[e, tr(A)/H][H/e] = tr(\Phi, A/B, B, \Psi) \) and \( \Phi, A/B, B, \Psi \xrightarrow{\chi^*} C \).

The case \( lab(e_0) = tr(B \setminus A) \) is treated similarly.

**Proposition 5.2.** If \( \Gamma \xrightarrow{\chi^*} C \), then \( tr(\Gamma) \xrightarrow{\chi^*} (tr(C)) \).
It is proved by a straightforward conversion of the reduction process for strings into the reduction process for graphs. These propositions yield

**Theorem 5.1.** If \( Gr \) is a BCG, then \( L(\text{tr}(Gr)) = \{w^*|w \in L(Gr)\} \).

### 6 Equivalence of HRGs and HBCGs

It is well known that CFGs and BCGs are equivalent; one of the simplest proofs involves Greibach normal form for CFGs. In this section, we show that this proof can be generalized to a wide class of graph grammars in a natural way.

#### 6.1 Greibach Normal Form for HRGs

Firstly, one has to extend the notion of the (weak) Greibach normal form. There are a few works in which variants of such extension are introduced, see [10,12]. However, normal forms presented in these works are more strict than it is needed for our purposes. In this paper, we use the following

**Definition 6.1.** The HRG \( Gr \) is in the weak Greibach normal form (WGNF) iff there is exactly one terminal edge in the right-hand side of each production. Formally, \( \forall (X \rightarrow H) \in P_{Gr} \exists! e_0 \in E_H : \text{lab}_H(e_0) \in \Sigma_{Gr} \) and for \( e \neq e_0 \) \( \text{lab}_H(e) \in N_{Gr} \).

Note that not each language generated by some HRG can be generated by an HRG in GNF. It follows from

**Example 6.1.** Consider an HRG \( Gr = \langle \{S\}, \{a\}, P, S \rangle \) where \( P \) contains two productions (for \( \text{type}(S) = \text{type}(a) = 0 \)):

- \( S \rightarrow (\{v_0\}, \{e_0\}, \text{att}, \text{lab}, \varepsilon), \text{lab}(e_0) = S, \text{att}(e_0) = \varepsilon; \)
- \( S \rightarrow \bullet(a). \)

This grammar produces graphs that have exactly one edge labeled by \( a \) and arbitrarily many isolated nodes. If there is an equivalent \( Gr' = \langle N, \{a\}, P', S' \rangle \) in GNF, then each right-hand side of each production in \( P' \) contains exactly one terminal edge. Note that if \( S' \xrightarrow{k} G, G \in \mathcal{H}(\{a\}) \) in \( Gr' \), then \( G \) has \( k \) terminal edges; hence \( k \) has to equal 1 and therefore \( S' \rightarrow G \in P' \). However, there are infinitely many graphs in \( L(Gr) \) while \( |P'| < \infty \). \( \square \)

The characterization of languages generated by HRGs in the WGNF is a subject of the further study.

It turns out that HBCGs generate the same class of languages as HRGs in the normal form presented. This is proved below.
6.2 The Equivalence Theorems

**Definition 6.2.** The set \( \text{st}(T) \) of subtypes of a type \( T \) is defined inductively as follows:

1. \( \text{st}(p) = \{ p \}, p \in Pr \);
2. \( \text{st}(\vdash (N/D)) = \{ \vdash (N/D) \} \cup \text{st}(N) \cup \left( \bigcup_{e \in E_G \setminus \{ \text{head}(D) \}} \text{st}(\text{lab}_D(e)) \right) \).

**Theorem 6.1.** For each HBCG an equivalent HRG in GNF exists.

*Proof.* Let \( Gr = (\Sigma, s, \triangleright) \) be an HBCG. We denote by \( N \) the set of subtypes of all types contained in the dictionary (formally, \( N = \{ R \in Tp_{\chi} | R \in \text{st}(T), T \in \text{dict}(Gr) \} \)). The set \( P \) contains the following productions:

1. If \( a \triangleright T \), then \( T \rightarrow \ominus(a) \in P \).
2. If \( a \triangleright (N/D) \), then \( N \rightarrow D[\text{head}(D) := a] \in P \).

It is argued that \( Gr' = (N, \Sigma, P, s) \) generates the same language as \( Gr \). It suffices to show that \( T \not\rightarrow \chi(\ominus(a)) \) for \( G \in \mathcal{H}(\Sigma), T \in N \) if and only if \( f(G) \not\rightarrow _\chi \ominus(T) \) for some \( f : E_G \rightarrow Tp_{\chi} \) such that \( \text{lab}(e) \triangleright f(e) \). It is done in a straightforward way by induction on the number of steps in the derivation:

“Only if” part. Let \( T \not\rightarrow_k G \). Induction on \( k \). **Basis:** if \( k = 1 \), then either \( G = \ominus(a) \) and \( a \triangleright T \) or \( G = D[\text{head}(D) := a] \) is obtained in such a way that \( a \triangleright \vdash (T/D) \) and \( |E_D| = 1 \). Then \( D[\text{head}(D) := \vdash (T/D)] \not\rightarrow _\chi \ominus(T) \). **Step** \((k > 1)\): let the first production applied be of the form \( T \rightarrow D[\text{head}(D) := a] \) for \( a \triangleright \vdash (T/D) \) and let \( D \setminus \{ \text{head}(D) \} = \{ e_1, \ldots, e_n \}, n \geq 1 \). Let \( G_i \in \mathcal{H}(\Sigma) \) be a graph that is obtained from \( T_i = \text{lab}(e_i) \) in the derivation process \((i = 1, \ldots, n) \). Note that \( G_i \) is a subgraph of \( G \). By induction hypothesis, \( f_i(G_i) \not\rightarrow _\chi \ominus(T_i) \) for some appropriate \( f_i : E_{G_i} \rightarrow Tp_{\chi} \). Then \( f_i \) can be combined into a single function \( f \) as follows: \( f(e) := f_i(e) \) whenever \( e \in G_i \) and \( f(\text{head}(D)) := \vdash (T/D) \). Thus \( f(G) \) can be reduced to \( \tilde{G} = D[\text{head}(D) := \vdash (T/D)] \) by induction hypothesis (its \( n \) subgraphs is compressed into \( n \) edges), and then \( \vdash (T/D) \) can be applied to \( \vdash (N/D) \) in such a way that \( \tilde{G} \not\rightarrow _\chi \ominus(T) \).

The “if” part is proved similarly: one has to transform applications of \( \vdash (T/D) \) into productions in \( Gr' \).

Example 4.2 provides an example of application of the theorem above.

**Theorem 6.2.** Each HRG \( Gr \) in GNF is equivalent to some HBCG.

*Proof.* Let \( Gr = (N, \Sigma, P, S) \). Consider elements of \( N \) as elements of \( Pr \) with the same function type defined on them. Since \( Gr \) is in GNF, each production in \( P \) is of the form \( \pi = X \rightarrow G \) where \( G \) contains exactly one terminal edge \( e_0 \) (say \( \text{lab}_G(e_0) = a \in \Sigma \)). We convert this production into the type \( T_\pi := \vdash (X/G[e_0 := S]) \). Then we introduce the HBCG \( Gr' = (\Sigma, S, \triangleright) \) where \( \triangleright \) is defined as follows: \( a \triangleright T_\pi \). Finally, note that, if one applies the transformation described in Theorem 6.1 to \( Gr' \), he obtains \( Gr \), which implies that \( L(Gr) = L(Gr') \).
Corollary 6.1. The problem of whether a given graph belongs to the language generated by a given HBCG is NP-complete.

Proof. This problem is in NP since, if the answer is “YES”, there is a certificate of polynomial size that justifies this; namely, this is a sequence of applications of (÷) (a derivation). Another explanation is that an HBCG can be converted into an equivalent HRG in polynomial time for whom the membership problem is in NP. In [9], an NP-complete graph language generated by some HRG ERG is introduced. One notices that there is at least one terminal edge in each production in ERG; by adding nonterminal symbols corresponding to terminal ones one transforms ERG into an equivalent one in GNF, and then — to an HBCG using Theorem 6.2 (it all takes polynomial time). ⊓⊔

7 Structural Properties

In this section, we study some structural properties of HBCGs.

7.1 HBCGs with One Primitive Type

The set of primitive types Pr is countably infinite; i.e. we are allowed to use as many primitive types as we want. However, the following theorem shows that it suffices to have one primitive type only.

Theorem 7.1. For each HBCG Gr = ⟨Σ, s, ⊧⟩ an equivalent one ⟨Σ, s, ⊧′⟩ exists such that types in its dictionary do not have primitive subtypes except for s.

Proof. We consider the following substitution of primitive types. Let s, p₁, ..., pₙ be all the primitive types occurring in dict(Gr) and let T be the set of all subtypes of all types in dict(Gr). Denote M := max{type(T)|T ∈ T}. Then we define F(pₖ) = ÷(s/Dₖ) (k = 1, ..., n):

- Vₖ = {v₁, v₂, ..., vₖ, w₁, w₂, ..., wₘ+k} (for tₖ = type(pₖ)).
- Eₖ = {e₀, e₁}.
- attₖ(e₀) = v₁v₂...vₖ, attₖ(e₁) = w₁w₂...wₘ+k.
- lab(e₀) = $, lab(e₁) = T_k where
  - T_k = ÷(s/Dₖ);
  - Dₖ = ⟨{u₁, ..., uₘ+k}, e', att', lab', u₁ ... uₖ⟩ (for t = type(s));
  - att'(e') = u₁ ... uₘ+k, lab'(e') = $.
- extₖ = w₁w₂...wₖ.

Behind this definition a simple idea stands: F(pₖ) has a huge edge in the denominator, which is larger than any edge existing in the lexicon.

Let F(T) stand for a type obtained from T ∈ T by substituting each pₖ with F(pₖ) (we do not change s). Now, if a > T, then let a >' F(T). No more relations in >' exist. We argue that Gr' = ⟨Σ, s, >'⟩ is a desired grammar. It follows from its definition that it contains only s as a primitive type. Clearly,
Let \( L(G_r) \subseteq L(G_r') \): each derivation in \( G_r \) can be remade in \( G_r' \), if one considers \( F(p_k) \) as an atomic, indivisible type corresponding to \( p_k \).

To prove the reverse inclusion let us consider the set \( T' = \{ F(T) | T \in T \} \). Assume that for a graph \( G \in \mathcal{H}(T) \) the graph \( F(G) \) is reducible to \( \otimes(s) \) (here \( F(G) \) is obtained from \( G \) by changing each label \( a \) by the label \( F(a) \)). At each step of the reduction process the rule \((\div)\) is applied either to a type of the form \( F(\div(N/D)) \), \((N/D) \in T \) or to \( F(p_k) \). However, note that no edges in \( F(G) \) have type exceeding \( M \), whereas \( D_k \) requires to be overlaid on the edge of the type \( M+k>M \). Consequently, \((\div)\) cannot be applied to \( F(p_k) \), and \( G \not\rightarrow_{\otimes(s)} \).

If \( H \) belongs to \( L(G_r') \), then there is a function \( f : E_H \rightarrow T' \) such that \( \text{lab}(e) \triangleright f(e) \) whenever \( e \in E_H \) and \( f(H) \not\rightarrow_{\otimes(s)} \). Since \( f(H) = F(G) \) for some \( G \in \mathcal{H}(T) \), \( G \) is also reducible to \( \otimes(s) \). Finally, note that \( g = F^{-1} \circ f \) satisfies the condition \( \text{lab}(e) \triangleright g(e) \) for \( e \in E_H \) and \( G = g(H) \not\rightarrow_{\otimes(s)} \). Therefore, \( H \) belongs to \( L(G_r) \).

\[ \square \]

### 7.2 Counters

One of the features HBCG's inherit from BCG's is so-called counters.

**Definition 7.1.** Let \( f : Pr \rightarrow \mathbb{Z} \) be some function. An \( f \)-counter \( \#_f : Tp_{\lambda} \rightarrow \mathbb{Z} \) is defined as follows:

- \( \#_f(p) = f(p) \);
- If \( T = \div(N/D) \) and \( E_D = \{ \text{head}(D), e_1, \ldots, e_n \} \), then

\[
\#_f(T) = \#_f(N) - \sum_{i=1}^{n} \#_f(\text{lab}(e_i)).
\]

If \( G \) is labeled by types and \( E_G = \{ e_1, \ldots, e_n \} \), then \( \#_f(G) := \sum_{i=1}^{n} \#_f(\text{lab}(e_i)) \).

**Proposition 7.1.** If \( G \not\rightarrow_{\lambda} G' \), then \( \#_f(G) = \#_f(G') \) for each \( f \).

**Proof.** Induction on \( k \). Basis: if \( k = 0 \), then \( G = G' \).

**Step:** let \( G \not\rightarrow_{\lambda} G[N/H] \not\rightarrow_{\lambda} G' \) where \( \div(N/D) \) is the type involved in the first rule \((\div)\). Let us denote all the participants of the compression \( G \not\rightarrow_{\lambda} G[N/H] \) similarly to those in Sect. 4.3. Then, since \( D \) and \((H;e_0)\) are \( d \)-isomorphic, \( \#_f(\text{lab}(e)) = \#_f(\text{lab}(e(e))) \) for \( e \in E_D, e \neq \text{head}(D) \). Thus \( \#_f(G) = \#_f(G[N/H]) \). The induction hypothesis completes the proof. \( \square \)

Counters can be used to check whether a graph \( G \) can be reduced to another one \( G' \): if \( \#_f(G) \neq \#_f(G') \) for some \( f \), then \( G \not\rightarrow_{\lambda} G' \).

**Example 7.1.** We provide two specific examples of counters:

- \( f = g_q, q \in Pr : g_q(p) = 1 \) whenever \( p = q \) and \( g_q(p) = 0 \) otherwise. E.g. for the graph \( E_0 \) from Example 4.1 \( \#_{g_q}(E_0) = -1, \#_{g_q}(E_0) = 0 \).
- \( f = h_m, m \in \mathbb{N} : h_m(p) = 1 \) whenever \( \text{type}(p) = m \) and \( h_m(p) = 0 \) otherwise.
8 Lambda Semantics

The \( \lambda \)-calculus is a formal tool, which has a number of applications in functional programming and in formal semantics. In this paper, we do not provide the definitions of this mechanism and refer the reader to the paper [5], which is an overview of the \( \lambda \)-calculus.

In basic categorial grammars, one can assign \( \lambda \)-terms to types. \( \alpha : A \) denotes a \( \lambda \)-term \( \alpha \) assigned to a type \( A \) (i.e. this is the pair \((\alpha; A)\)). The rules of reduction then have the following form:

1. \( \Gamma, \alpha : A, \beta : A \setminus B, \Delta \mapsto \Gamma, \beta \alpha : B, \Delta \)
2. \( \Gamma, \beta : B / A, \alpha : A, \Delta \mapsto \Gamma, \beta \alpha : B, \Delta \)

Here \( \beta \alpha \) stands for the application of \( \beta \) to \( \alpha \). A linguistic example that shows how \( \lambda \)-terms describe semantics of a natural language was given in Sect. 1.

This approach can be generalized to hypergraphs and HBCGs. Let \( T \) be a type, i.e. belong to \( T \in \chi \). By \( \tau : T \) we denote a pair containing a \( \lambda \)-term \( \tau \).

Now we are going to incorporate the \( \lambda \)-calculus into the rule \((\div)\). Let objects involved in this rule be denoted as in Sect. 4.3. We additionally require that edges in \( E_D \) are numbered: \( E_D = \{e_0, e_1, \ldots, e_k\} \) (and this numbering is fixed for a given type). If a \( \lambda \)-term \( \tau \) is assigned to \( \div (N/D) \) and for \( i > 0 \) \( \text{lab}(e_i) = \alpha_i : T_i \), then the rule \((\div)\) so ft h ef o rm \( G \mapsto G[\tau \alpha_1 \alpha_2 \ldots \alpha_k : N/H] \) \((\div)\).

Here \( \tau \alpha_1 \alpha_2 \ldots \alpha_k = ((\tau \alpha_1) \alpha_2) \ldots \alpha_k \). This means that \( \lambda \)-terms written on edges that are consumed by the denominator \( D \) are treated as arguments of the \( \lambda \)-term assigned to \( \div (N/D) \).

An example of an application of HBCGs enriched with the \( \lambda \)-calculus to linguistics is presented in the next section.

9 Cross-serial Dependencies

It is well known that context-free languages in the usual sense fail to describe certain linguistic phenomena. One of them is so-called cross-serial dependencies (CSD) — a class of phenomena that can be described by the language \( \{ww | w \in \Sigma^*\} \) of reduplicated strings.

We focus here on the following example of CSD from the Russian language:

\begin{center}
\begin{tabular}{l}
| Olya, Petya, Vasja финишировали первой, вторым, третьим. |
| Olya, Petya and Vasja finished the first, the second and the third. |
\end{tabular}
\end{center}

The meaning of this sentence is: \textit{Olya was the first who finished, Petya was the second who finished, and Vasya was the third who finished} (e.g. when speaking about a competition). In Russian, the ordinal numerals \textit{the first, the second, the third} agree with nouns in gender which leads to CSD (note that \textit{Olya} is a
female name, and Petya, Vasya are male ones). Below we show how to generate such Russian sentences using an HBCG and how to model their semantics using the λ-calculus. In order to simplify the example, we ignore some features of the Russian language. We denote by \(pd_m\) (\(pd_f\)) a primitive type which stands for masculine (feminine resp.) predicate phrases in singular form in the instrumental case (such as ordinal numbers, e.g. первый); we denote by \(np_m\) (\(np_f\)) a primitive type corresponding to masculine (feminine) noun phrases in singular form in the nominative case (such as proper nouns, e.g. Петя); \(np_p\) (\(pd_p\)) denotes nouns (predicate phrases resp.) in a plural form; \(s\) denotes sentences (it is a distinguished type). Then the grammar generating sentences of the above form is the following:

Вася, Петя (masculine) \(\triangleright\div \left( Z/\begin{array}{c} s \\ (1) \\ pd_m \\ (2) \\ (3) \\ (4) \end{array} \right) \div \left( Z/\begin{array}{c} s \\ (1) \\ pd_m \\ (2) \\ (3) \\ (4) \end{array} \right)\)

Оля, Маша (feminine) \(\triangleright\div \left( Z/\begin{array}{c} s \\ (1) \\ pd_f \\ (2) \\ (3) \\ (4) \end{array} \right) \div \left( Z/\begin{array}{c} s \\ (1) \\ pd_f \\ (2) \\ (3) \\ (4) \end{array} \right)\)

первым, вторым, третьим, ... (masculine) \(\triangleright pd_m\);
первой, второй, третьей, ... (feminine) \(\triangleright pd_f\);

finisphirali \(\triangleright T_0 = tr((np_p/s)/pd_p)\).

Here \(Z\) equals \(\triangleright s/\begin{array}{c} s \\ (1) \\ 2 \\ 3 \\ 4 \\ T_0 \end{array}\).

Note that \(tr\) is defined in Sect. 5.

Let us denote the first type in the first row above as \(I_m\), and the second one as \(T_m\); by analogue, types in the second row are denoted by \(I_f\) and \(T_f\) resp.

In addition, we assign semantic types to the syntactic ones:

- Вася \(\triangleright \lambda P.\lambda x.f(P)(Vasya) : I_m\), \(\lambda z.\lambda P.\lambda f. (f(P)(Vasya) \land zf) : T_m\); ...
- Оля \(\triangleright \lambda P.\lambda f. (P)(Olya) : I_f\), \(\lambda z.\lambda P.\lambda f. (f(P)(Olya) \land zf) : T_f\); ...
- первым: \(\text{first} : pd_m\), вторым: \(\text{second} : pd_m\), ...;
- первым: \(\text{first} : pd_f\), вторым: \(\text{second} : pd_f\), ...;
- финишериали: \(\lambda P.\lambda x.\text{finish}(P)(x) : T_0\).

Then the sentence (1) belongs to the language generated by this HBCG:
Let us track the \(\lambda\)-terms occurring in the derivation process step-by-step:

1. There is a term \(\tau_1 = \lambda f.f\) assigned to \(Z\) after the first step;
2. \(\zeta = \tau_1\) and \(P = second\) are applied to \(\lambda \zeta.\lambda P.\lambda g.(g(Petya) \land \zeta g)\) in the second step; the result is \(\lambda g.(g(second)(Petya) \land g(first)(Olya))\).
3. Similarly, after the third step the following \(\lambda\)-term is assigned to \(Z\):
   \[
   \lambda h. (h(third)(Vasya) \land (h(second)(Petya) \land h(first)(Olya))).
   \]
4. Finally, one obtains the following term:
   
   \[
   finish(third)(Vasya) \land (finish(second)(Petya) \land finish(first)(Olya)).
   \]

Thus, this HBCG not only generates sentences of the form (1) (HRGs can deal with them as well) but also provides their semantical representation which is composed of \(\lambda\)-terms assigned to types.

# 10 Conclusions and Related Work

There are a few papers devoted to combining categorial grammars with graph tools. E.g. there is a recent work of Sebastian Beschke and Wolfgang Menzel [7] where it is shown how to enrich the Lambek calculus, which is another categorial approach (see [13]), with graph semantics. In the paper [14] (which is rather linguistic than mathematical) an extension of some concepts of the Lambek calculus to graphs is presented; namely, sentences are considered to be graph structures (functor-argumentor-structures), and then categorial graph grammars are introduced, which deal with these structures. Hypergraph basic categorial grammars introduced in our work, however, do not seem to be closely related to any of these approaches. Possibly, it is because our motivation for HBCGs is rather logical and mathematical: our main purpose was to directly combine concepts of BCGs and HRGs, so the resulting mechanism satisfies these requirements. Nevertheless, we hope that HBCGs also can be used (possibly, with some further modifications) in practical applications, e.g. in linguistics.

Note that there is a work [6] where HRGs are used to describe CSD of the Dutch language. Actually, examples in Sect. 9 have a similar structure with examples in [6]. Comparing [6] with this paper, we conclude that one of the crucial features distinguishing between HRGs and HBCGs from linguistic point of view is the \(\lambda\)-semantics, which can be naturally built into the latter.

There is a number of questions that remain open; we hope to study them in future works.
We showed that the membership problem for HBCGs is NP-complete. How to restrict HBCGs in order to obtain efficient parsing algorithms?

- We introduced some applications of HBCGs to linguistics. We are interested in further developing a theory that would use HBCGs and the \( \lambda \)-calculus to model visual structures related to natural languages. Particularly, we desire to consider syntactic trees which linguists deal with from the point of view of our approach.

- How to generalize other string categorial approaches to hypergraphs?

Acknowledgments. I thank my scientific advisor prof. Mati Pentus for his careful attention to my study and anonymous reviewers for their valuable advice.

References

1. Aho, A.V., Ullman, J.D.: The Theory of Parsing, Translation, and Compiling. Prentice-Hall, Upper Saddle River (1972)
2. Ajdukiewicz, K.: Die syntaktische Konnexitat (Syntactic connexion). Studia Philosophica 1, 1–27 (1935)
3. Bar-Hillel, Y.: A quasi-arithmetical notation for syntactic description. Language 29, 47–58 (1953)
4. Bar-Hillel, Y., Gaifman, H., Shamir, E.: On categorial and phrase structure grammars. Bull. Res. Counc. Israel 9, 1–6 (1960)
5. Barendregt, H., Barendsen, E.: Introduction to lambda calculus. Nieuw archief voor wisenkunde 4, 337–372 (1984)
6. Bauer, D., Owen, R.: Hyperedge replacement and nonprojective dependency structures. In: TAG (2016)
7. Beschke, S., Menzel, W.: Graph algebraic combinatory categorial grammar, pp. 54–64 (2018)
8. Buszkowski, W.: Categorial grammars and their logics. In: Garrido, Á., Wybraniec-Skardowska, U. (eds.) The Lvov-Warsaw School. Past and Present. SUL, pp. 91–115. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-65430-0_6
9. Drewes, F., Kreowski, H.-J., Habel, A.: Hyperedge replacement graph grammars (1997)
10. Engelfriet, J.: A Greibach normal form for context-free graph grammars. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 138–149. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55719-9_70
11. Gilroy, S., Lopez, A., Maneth, S.: Parsing graphs with regular graph grammars, pp. 199–208 (2017)
12. Jansen, C., Heinen, J., Katoen, J.-P., Noll, T.: A local Greibach normal form for hyperedge replacement grammars. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 323–335. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21254-3_25
13. Lambek, J.: The mathematics of sentence structure. Am. Math. Monthly 65(3), 154–170 (1958)
14. Robering, K.: Categorial graph grammar: a direct approach to functor-argumentor-structure. Theor. Linguist. 26, 31–73 (2000)