

# Second-Order Abstract Categorical Grammars as Hyperedge Replacement Grammars

Makoto Kanazawa\*

National Institute of Informatics  
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan

**Abstract.** Second-order *abstract categorical grammars* (de Groote 2001) and *hyperedge replacement grammars* (Bauderon and Courcelle 1987, Habel and Kreowski 1987) are two natural ways of generalizing “context-free” grammar formalisms for string and tree languages. It is known that the string generating power of both formalisms is equivalent to (non-erasing) *multiple context-free grammars* (Seki et al. 1991) or *linear context-free rewriting systems* (Weir 1988). In this paper, we give a simple, direct proof of the fact that second-order ACGs are simulated by hyperedge replacement grammars, which implies that the string and tree generating power of the former is included in that of the latter. The normal form for tree-generating hyperedge replacement grammars given by Engelfriet and Maneth (2000) can then be used to show that the tree generating power of second-order ACGs is exactly the same as that of hyperedge replacement grammars.

## 1 Introduction

Many types of grammars have been devised for string and tree languages that are like ordinary context-free grammars in that each grammar is associated with a *local* set of derivation trees, but have more complex types of derived objects. Table 1 is a classification of some of these so-called “context-free” grammar formalisms according to what kind of derived object derivation trees yield. This paper compares two grammar formalisms that generalize these different grammars, namely, second-order *abstract categorical grammars* and *hyperedge replacement grammars*.

In *abstract categorical grammars* introduced by de Groote 2001, both grammatical derivations and derived objects (strings, trees, logical formulae, etc.) are represented by linear  $\lambda$ -terms. These two sets of linear  $\lambda$ -terms associated with an ACG are known as its *abstract language* and *object language*. When the constants of the abstract vocabulary have at most second-order types,  $\lambda$ -terms in the abstract language do not contain any  $\lambda$ -abstraction and can hence be regarded as trees. Since in this case the abstract language is a *local* tree language, second-order ACGs may be called “context-free” grammars on linear  $\lambda$ -terms.<sup>1</sup> De Groote and Pogodalla (2004) showed

---

\* This work was supported by the Japan Society for the Promotion of Science under the Grant-in-Aid for Scientific Research (18-06739).

<sup>1</sup> We use the term “context-free” informally in this paper. See Courcelle 1987 for a precise definition of an abstract notion of context-free grammar.

**Table 1.** Context-free grammars on ...

	strings	trees	unary tree contexts $C[y]$	$n$ -ary tree contexts $C[y_1, \dots, y_n]$
single	CFG	RTG	TAG (monadic LCFTG)	LCFTG
multiple	MCFG	MRTG	MCTAG	MLCFTG

RTG: regular tree grammar

LCFTG: linear non-deleting context-free tree grammar (Rounds 1970, Engelfriet and Schmidt 1977, Kepser and Mönnich 2006)

MCFG: multiple context-free grammar (Seki et al. 1991)

MRTG: multiple regular tree grammar (Raoult 1997, Engelfriet 1997)

MCTAG: (set-local) multi-component tree-adjoining grammar (Weir 1988)

MLCFTG: multiple linear non-deleting context-free tree grammar (see Section 5)

that non-erasing *multiple context-free grammars* (Seki et al. 1991), or string-based *linear context-free rewriting systems* (Weir 1988), and linear non-deleting *context-free tree grammars* (Rounds 1970, Engelfriet and Schmidt 1977, Kepser and Mönnich) can be faithfully encoded by second-order ACGs. Subsequently, Salvati (2007) showed that the string generating power of second-order ACGs exactly matches that of LCFRSs using the fact that the latter coincides with the class of output languages of *deterministic tree-walking transducers* (Weir 1992). The ability of second-order ACGs to capture the above-mentioned “context-free” grammar formalisms is explained by the fact that linear  $\lambda$ -terms generalize both strings and trees and can express “linear and non-deleting” operations on them, such as concatenation and (linear and non-deleting) second-order tree substitution (i.e., tree homomorphism).

Another very general “context-free” grammar formalism, encompassing both string grammars and tree grammars, is that of *hyperedge replacement grammars* (Bauderon and Courcelle 1987, Habel and Kreowski 1987), which generate sets of *hypergraphs*. Hypergraphs are a generalization of graphs that allows edges to be incident on any number of nodes, and can represent strings and trees in a straightforward way. The operation of *hyperedge replacement* can express many operations on strings and trees, including concatenation and (linear and non-deleting) second-order tree substitution. It is known that the string generating power of hyperedge replacement grammars coincides with the class of output languages of deterministic tree-walking transducers (Engelfriet and Heyker 1991). By Salvati’s (2007) result, this implies the equivalence of second-order ACGs and hyperedge replacement grammars in string generating power.

Since second-order ACGs and hyperedge replacement grammars are two natural ways of generalizing “context-free” string grammars and tree grammars, it is interesting to know whether their tree generating power is also the same. In this paper, we give a very simple proof of the fact that second-order ACGs are “simulated” by hyperedge replacement grammars, which implies that the generating power of the former is included in that of the latter both in the case of strings and in the case of trees. The main idea here is that typed linear  $\lambda$ -terms employed by ACGs can be represented by hypergraphs, and  $\lambda$ -term substitution  $M[x := N]$  corresponds to hyperedge replacement. To show the

converse direction in the case of trees, we can use the normal form for tree-generating hyperedge replacement grammars given by Engelfriet and Maneth (2000), thus completing the proof that the tree generating power of the two formalisms is exactly the same.

## 2 Preliminaries

### 2.1 Types and $\lambda$ -terms

Given a finite set  $A$  of *atomic types*, the set  $\mathcal{T}(A)$  of *types* built upon  $A$  is defined to be the closure of  $A$  under the rule:  $\alpha, \beta \in \mathcal{T}(A)$  implies  $(\alpha \rightarrow \beta) \in \mathcal{T}(A)$ . We omit outermost parentheses, write  $\alpha \rightarrow \beta \rightarrow \gamma$  for  $\alpha \rightarrow (\beta \rightarrow \gamma)$ , and  $\alpha^k \rightarrow \beta$  for  $\alpha \rightarrow \dots \rightarrow \alpha \rightarrow \beta$  with “ $\alpha \rightarrow$ ” repeated  $k$  times. The *order* of an atomic type  $p$  is  $\text{ord}(p) = 1$ , and  $\text{ord}(\alpha \rightarrow \beta)$  is  $\max(\text{ord}(\alpha) + 1, \text{ord}(\beta))$ .

A *higher-order signature* is  $\Sigma = (A, C, \tau)$ , where  $A$  is a finite set of atomic types,  $C$  is a finite set of *constants*, and  $\tau: C \rightarrow \mathcal{T}(A)$  is an assignment of types to constants. The *order* of  $\Sigma$  is  $\max\{\text{ord}(\tau(c)) \mid c \in C\}$ . Fix a countably infinite set  $X$  of variables. The set  $\Lambda(\Sigma)$  of  $\lambda$ -terms over  $\Sigma$  is the closure of  $C \cup X$  under the rules: (i)  $M, N \in \Lambda(\Sigma)$  implies  $(MN) \in \Lambda(\Sigma)$ ; and (ii)  $x \in X, M \in \Lambda(\Sigma)$  implies  $(\lambda x.M) \in \Lambda(\Sigma)$ . We omit outermost parentheses and write  $MNP$  for  $(MN)P$ ,  $\lambda x.MN$  for  $\lambda x.(MN)$ , and  $\lambda x_1 \dots x_n.M$  for  $\lambda x_1.(\dots(\lambda x_n.M)\dots)$ . The set of *free variables* of a  $\lambda$ -term  $M$ ,  $\text{FV}(M)$  in symbols, is defined in the usual way. If  $\text{FV}(M) = \emptyset$ , then  $M$  is *closed*, and if  $M$  contains no constant,  $M$  is *pure*. We take for granted the notions of *substitution* (of a  $\lambda$ -term for a free variable in a  $\lambda$ -term),  $\beta$ -redex,  $\beta$ -reduction,  $\beta$ -normal form, etc. (See Hindley 1997 or Sørensen and Urzyczyn 2006 for these and other standard notions in simply typed  $\lambda$ -calculus.) We write  $M[x := N]$  for the result of substituting  $N$  for  $x$  in  $M$ , write  $\rightarrow_\beta$  for  $\beta$ -reduction, and let  $|M|_\beta$  denote the  $\beta$ -normal form of  $M$ .

A  $\lambda$ -term  $M$  over  $\Sigma$  with  $\text{FV}(M) = \{x_1, \dots, x_n\}$  may be assigned a type  $\alpha$  under a *type environment*  $\{x_1 : \alpha_1, \dots, x_n : \alpha_n\}$ , or in symbols:

$$x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash_\Sigma M : \alpha,$$

according to the following rules:<sup>2</sup>

(cons)  $\vdash_\Sigma c : \tau(c)$  where  $c \in C$ ,

(var)  $x : \alpha \vdash_\Sigma x : \alpha$  where  $x \in X$  and  $\alpha \in \mathcal{T}(A)$ ,

(abs)  $\frac{\Gamma \vdash_\Sigma M : \beta}{\Gamma - \{x : \alpha\} \vdash_\Sigma \lambda x.M : \alpha \rightarrow \beta}$  provided  $\Gamma \cup \{x : \alpha\}$  is a type environment,

(app)  $\frac{\Gamma \vdash_\Sigma M : \alpha \rightarrow \beta \quad \Delta \vdash_\Sigma N : \alpha}{\Gamma \cup \Delta \vdash_\Sigma MN : \beta}$  provided  $\Gamma \cup \Delta$  is a type environment.

The proviso in (abs) means that either  $x : \alpha$  is in  $\Gamma$  or  $x$  does not appear in  $\Gamma$ .

<sup>2</sup> Our use of the symbol  $\vdash$  corresponds to  $\mapsto$  in Hindley 1997 and  $\Rightarrow$  in Mints 2000. Although  $x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash_\Sigma M : \alpha$  implies  $\text{FV}(M) = \{x_1, \dots, x_n\}$ , Weakening is derivable in this system in the sense that  $\Gamma \vdash_\Sigma M : \alpha$  implies  $\Gamma, x : \beta \vdash_\Sigma (\lambda y.M)x : \alpha$ , where  $x, y \notin \text{FV}(M)$ .

An expression  $\Gamma \vdash_{\Sigma} M : \alpha$  is called a *judgment*, and when it is derivable using the above rules, it is called a *typing* of  $M$ . A  $\lambda$ -term  $M$  is said to be *typable* if it has a typing. We write  $\Gamma \vdash M : \alpha$  when  $M$  is a pure  $\lambda$ -term, omitting reference to  $\Sigma$ . The tree-like figure showing how the inference rules are used to derive a judgment  $\Gamma \vdash_{\Sigma} M : \alpha$  is called its *deduction*. When  $M$  is  $\beta$ -normal, every typing of  $M$  has a unique deduction.

A  $\lambda$ -term  $M$  is in  *$\eta$ -long form* relative to a deduction  $\mathcal{D}$  of  $\Gamma \vdash_{\Sigma} M : \alpha$  if whenever a judgment of the form

$$\Gamma' \vdash_{\Sigma} N : \beta \rightarrow \gamma$$

occurs in  $\mathcal{D}$ , it is either the conclusion of (abs) or the left premise of (app). We call a  $\lambda$ -term  $\eta$ -long relative to a typing when it is  $\eta$ -long relative to some deduction of the typing. We also speak of a closed  $\lambda$ -term being in  $\eta$ -long form relative to a type, with the obvious meaning. Let  $M \in \Lambda(\Sigma)$  be any typable  $\lambda$ -term. For every deduction  $\mathcal{D}$  of a typing  $\Gamma \vdash_{\Sigma} M : \alpha$  of  $M$ ,  $M$  can be turned into a  $\lambda$ -term  $M'$  in  $\eta$ -long form relative to a deduction  $\mathcal{D}'$  of  $\Gamma \vdash_{\Sigma} M' : \alpha$  by repeated application of  *$\eta$ -expansion*, which replaces any occurrence of a judgment

$$\Gamma' \vdash_{\Sigma} N : \beta \rightarrow \gamma$$

which is neither the conclusion of (abs) nor the left premise of (app) by

$$\frac{\frac{\Gamma' \vdash_{\Sigma} N : \beta \rightarrow \gamma \quad x : \beta \vdash_{\Sigma} x : \beta}{\Gamma', x : \beta \vdash_{\Sigma} Nx : \gamma}}{\Gamma' \vdash_{\Sigma} \lambda x. Nx : \beta \rightarrow \gamma}$$

where  $x$  is a variable not in  $\text{FV}(N)$ .

Every typable  $\lambda$ -term  $M \in \Lambda(\Sigma)$  has a most general typing from which all other typings may be obtained by type substitution; it is called a *principal typing* of  $M$ . We call  $M$  in  $\eta$ -long form if it is  $\eta$ -long relative to its principal typing. When  $M$  is  $\eta$ -long relative to a typing  $\Gamma \vdash_{\Sigma} M : \alpha$ , this typing can be obtained from a principal typing of  $M$  by a type substitution that maps atomic types to atomic types.

Let  $M \rightarrow_{\beta} M'$  in one step by contraction of a  $\beta$ -redex  $(\lambda x.N)P$ . This  $\beta$ -reduction step is *non-erasing* if  $x \in \text{FV}(N)$ , and *non-duplicating* if  $x$  occurs free in  $N$  at most once. The  $\beta$ -reduction from  $M$  to  $M'$  is non-erasing (non-duplicating) if it consists entirely of non-erasing (non-duplicating)  $\beta$ -reduction steps. The following are two important and well-known properties of the type assignment system which we will make use of in this paper (see Hindley 1997):

**Subject Reduction Theorem.** *If  $\Gamma \vdash_{\Sigma} M : \alpha$  and  $M \rightarrow_{\beta} M'$ , then  $\Gamma' \vdash_{\Sigma} M' : \alpha$ , where  $\Gamma'$  is the restriction of  $\Gamma$  to  $\text{FV}(M')$ .*

**Subject Expansion Theorem.** *If  $\Gamma \vdash_{\Sigma} M' : \alpha$  and  $M \rightarrow_{\beta} M'$  by non-erasing non-duplicating  $\beta$ -reduction, then  $\Gamma \vdash_{\Sigma} M : \alpha$ .*

A  $\lambda$ -term is said to be *linear* if no variable occurs free more than once in any subterm, and for every subterm of the form  $\lambda x.N$ ,  $x \in \text{FV}(N)$ . We denote the set of linear  $\lambda$ -terms over  $\Sigma$  by  $\Lambda_{\text{lin}}(\Sigma)$ . The set  $\Lambda_{\text{lin}}(\Sigma)$  is closed under  $\beta$ -reduction, and any  $\beta$ -reduction starting with a linear  $\lambda$ -term is non-erasing and non-duplicating. Every typing of a linear  $\lambda$ -term  $M$  has a unique deduction, whether or not  $M$  is  $\beta$ -normal.

## 2.2 Trees and Strings as $\lambda$ -terms

Recall that a *ranked alphabet* is a finite set  $\Delta = \bigcup_{n \in \mathbb{N}} \Delta^{(n)}$ , where  $\Delta^{(m)} \cap \Delta^{(n)} = \emptyset$  if  $m \neq n$ . If  $f \in \Delta^{(n)}$ ,  $n$  is the *rank* of  $f$ , written  $\text{rank}_\Delta(f)$ . A *tree* over  $\Delta$  is an expression  $fT_1 \dots T_n$ , where  $\text{rank}_\Delta(f) = n$  and  $T_1, \dots, T_n$  are trees over  $\Delta$ . The set of trees over  $\Delta$  is denoted  $\mathbb{T}_\Delta$ . Let  $Y_k = \{y_1, \dots, y_k\}$  be a set of  $k$  variables. The notation  $\mathbb{T}_\Delta(Y_k)$  denotes the set  $\mathbb{T}_{\Delta \cup Y_k}$ , where  $\text{rank}_{\Delta \cup Y_k}(y_i) = 0$  for all  $y_i \in Y_k$ .

If  $\Delta$  is a ranked alphabet,  $\text{inc}(\Delta)$  is the ranked alphabet such that  $(\text{inc}(\Delta))^{(0)} = \emptyset$  and  $(\text{inc}(\Delta))^{(n+1)} = \Delta^{(n)}$ . If  $\Delta$  is a ranked alphabet with  $\Delta^{(0)} = \emptyset$ , then  $\text{dec}(\Delta)$  is the ranked alphabet such that  $(\text{dec}(\Delta))^{(n)} = \Delta^{(n+1)}$ .

A ranked alphabet  $\Delta$  can be represented by a second-order signature  $\Sigma_\Delta^{\text{tree}} = (\{o\}, \Delta, \tau_\Delta)$ , where for each  $f \in \Delta^{(n)}$ ,  $\tau_\Delta(f) = o^n \rightarrow o$ . A tree in  $\mathbb{T}_\Delta(Y_k)$  can be identified with a  $\beta$ -normal  $\lambda$ -term  $T$  over  $\Sigma_\Delta^{\text{tree}}$  such that  $\text{FV}(T) \subseteq Y_k$  and  $\{y_i : o \mid y_i \in \text{FV}(T)\} \vdash_{\Sigma_\Delta^{\text{tree}}} T : o$ . A *simple tree*  $T \in \mathbb{T}_\Delta(Y_k)$ , i.e., a tree in which each  $y_i \in Y_k$  occurs exactly once, is a linear  $\lambda$ -term according to this identification. We call  $\Sigma_\Delta^{\text{tree}}$  a *tree signature*.

A string  $a_1 \dots a_n$  over an alphabet  $U$  can be represented by the linear  $\lambda$ -term  $/a_1 \dots a_n/ = \lambda y. a_1(\dots(a_n y) \dots)$  over the signature  $\Sigma_U^{\text{string}} = (\{o\}, U, \tau)$ , where  $\tau(a) = o \rightarrow o$  for all  $a \in U$ . For every string  $v \in U^*$ , we have  $\vdash_{\Sigma_U^{\text{string}}} /v/ : o \rightarrow o$ . We call  $\Sigma_U^{\text{string}}$  a *string signature*. Under this representation of strings, string concatenation is represented by the combinator  $\mathbf{B} = \lambda x y z. x(yz)$ .

An  $n$ -tuple of (simple) trees or strings can be represented by a linear  $\lambda$ -term of the form  $\lambda w. w M_1 \dots M_n$ , where the  $M_i$  represent individual trees or strings. Many operations on such tuples may be expressed as linear  $\lambda$ -terms. For example, the operation  $(u_1, u_2), (v_1, v_2) \mapsto u_1 v_1 u_2 v_2$ , which takes two pairs of strings and returns a string, is represented by

$$\lambda x_1 x_2 y. x_1(\lambda u_1 u_2. x_2(\lambda v_1 v_2. u_1(v_1(u_2(v_2 y))))).$$

## 2.3 Abstract Categorical Grammars

An *abstract categorical grammar* (de Groote 2001) is a quadruple  $\mathcal{G} = (\Sigma, \Sigma', \mathcal{L}, s)$ , where  $\Sigma = (A, C, \tau)$  (*abstract vocabulary*) and  $\Sigma' = (A', C', \tau')$  (*object vocabulary*) are higher-order signatures, the *lexicon*  $\mathcal{L}$  maps each atomic type in  $A$  to a type built upon  $A'$  and each abstract constant  $c$  in  $C$  to a closed linear  $\lambda$ -term  $\mathcal{L}(c)$  over  $\Sigma'$  such that  $\vdash_{\Sigma'} \mathcal{L}(c) : \mathcal{L}(\tau(c))$ , and  $s$  is an atomic type in  $A$ . The *abstract language* of  $\mathcal{G}$ , written  $\mathcal{A}(\mathcal{G})$ , is defined as  $\mathcal{A}(\mathcal{G}) = \{P \in \Lambda_{\text{lin}}(\Sigma) \mid \vdash_{\Sigma'} P : s\}$ . The *object language* of  $\mathcal{G}$  is  $\mathcal{O}(\mathcal{G}) = \{|\mathcal{L}(P)|_\beta \mid P \in \mathcal{A}(\mathcal{G})\}$ . An ACG is  *$m$ -th order* if the *order* of the abstract vocabulary does not exceed  $m$ . For  $m, n \geq 1$ , the notation  $\mathbf{G}(m, n)$  denotes the class of  $m$ -th order ACGs whose lexicon maps each atomic abstract type to a type of order  $\leq n$ . We let  $\mathbf{G}(m)$  stand for the class of  $m$ -th order ACGs in this paper (i.e.,  $\mathbf{G}(m) = \bigcup_{n \geq 1} \mathbf{G}(m, n)$ ).

If  $\mathcal{G} = (\Sigma, \Sigma', \mathcal{L}, s)$  is a second-order ACG, where  $\Sigma = (A, C, \tau)$ , and  $p \in A$ , then a  $\beta$ -normal  $\lambda$ -term  $M \in \Lambda(\Sigma)$  with  $\vdash_{\Sigma'} M : p$  is a tree and the set of all such trees is a *local tree language*, or equivalently, the set of derivation trees of some context-free grammar.

Let  $\mathcal{G} = (\Sigma, \Sigma', \mathcal{L}, s)$  be an ACG. If  $\Sigma'$  is a tree signature and  $\mathcal{L}(s) = o$ , then  $\mathcal{O}(\mathcal{G})$  is a tree language, and we call  $\mathcal{G}$  a *tree-generating ACG*. If  $\Sigma'$  is a string signature and

$\mathcal{L}(s) = o \rightarrow o$ , then we call  $\mathcal{G}$  a *string-generating ACG*, and say  $\mathcal{G}$  generates a string language  $L$  if  $O(\mathcal{G}) = \{ /w/ \mid w \in L \}$ . If  $\mathbf{G}$  is a class of ACGs, we let  $TR(\mathbf{G})$  and  $STR(\mathbf{G})$  stand for the class of tree languages and the class of string languages, respectively, generated by ACGs in  $\mathbf{G}$ .

*Example 1.* The following ACG  $\mathcal{G} = (\Sigma, \Sigma_U^{\text{string}}, \mathcal{L}, s)$ , where  $U = \{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4\}$ , belongs to  $\mathbf{G}(2, 4)$  and generates  $\text{RESP} = \{ a_1^m a_2^m b_1^n b_2^n a_3^m a_4^m b_3^n b_4^n \mid m, n \geq 0 \}$ :

$$\begin{aligned} \Sigma &= (\{p, q, s\}, \{c_1, c_2, c_3, c_4, c_5\}, \tau) \\ \tau(c_1) &= p \rightarrow q \rightarrow s, \quad \tau(c_2) = p, \quad \tau(c_3) = q, \quad \tau(c_4) = p \rightarrow p, \quad \tau(c_5) = q \rightarrow q, \\ \mathcal{L}(p) &= \mathcal{L}(q) = ((o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o) \rightarrow o, \quad \mathcal{L}(s) = o \rightarrow o, \\ \mathcal{L}(c_1) &= \lambda x_1 x_2 y. x_1 (\lambda u_1 u_2. x_2 (\lambda v_1 v_2. u_1 (v_1 (u_2 (v_2 y)))))), \\ \mathcal{L}(c_2) &= \mathcal{L}(c_3) = \lambda w. w (\lambda y. y) (\lambda y. y), \\ \mathcal{L}(c_4) &= \lambda x w. x (\lambda v_1 v_2. w (\lambda y. a_1 (v_1 (a_2 y)))) (\lambda y. a_3 (v_2 (a_4 y))), \\ \mathcal{L}(c_5) &= \lambda x w. x (\lambda v_1 v_2. w (\lambda y. b_1 (v_1 (b_2 y)))) (\lambda y. b_3 (v_2 (b_4 y))). \end{aligned}$$

For example,  $c_1(c_4(c_4c_2))(c_5c_3) \in \mathcal{A}(\mathcal{G})$ , and we have

$$\begin{aligned} \mathcal{L}(c_4c_2) &\rightarrow_{\beta} \lambda w. w /a_1 a_2 / / a_3 a_4 /, \\ \mathcal{L}(c_4(c_4c_2)) &\rightarrow_{\beta} \lambda w. w /a_1 a_1 a_2 a_2 / / a_3 a_3 a_4 a_4 /, \\ \mathcal{L}(c_5c_3) &\rightarrow_{\beta} \lambda w. w /b_1 b_2 / / b_3 b_4 /, \\ \mathcal{L}(c_1(c_4(c_4c_2))(c_5c_3)) &\rightarrow_{\beta} /a_1 a_1 a_2 a_2 b_1 b_2 a_3 a_3 a_4 a_4 b_3 b_4 / \end{aligned}$$

It was shown by de Groote and Pogodalla (2004) that the class MCFL of multiple context-free languages (or, equivalently, LCFRS languages) is included in  $STR(\mathbf{G}(2, 4))$ . Salvati (2007) in turn showed  $STR(\mathbf{G}(2)) \subseteq \text{MCFL}$  using Weir's (1992) result that MCFL coincides with the class of output languages of deterministic tree-walking transducers. Thus,  $STR(\mathbf{G}(2, 4)) = STR(\mathbf{G}(2)) = \text{MCFL}$ , and the hierarchy  $STR(\mathbf{G}(2, m))$  collapses for  $m \geq 4$ . Salvati (2005) also showed that for any  $G \in \mathbf{G}(2)$ ,  $O(G)$  belongs to the class P, while Kanazawa (2007) tightened the complexity bound to LOGCFL.

Turning to third-order ACGs, Salvati (2005) exhibited an NP-complete tree language belonging to  $TR(\mathbf{G}(3, 1))$ , while Yoshinaka and Kanazawa (2005) showed that  $STR(\mathbf{G}(3, 2))$  includes languages that are not semilinear. No non-trivial complexity upper bound is known for languages generated by  $m$ -th order ACGs for  $m \geq 3$ .

Kanazawa (2006) showed that for  $m, n \geq 2$ ,  $STR(\mathbf{G}(m, n))$  is a substitution-closed full abstract family of languages. In the same paper, it was shown that for  $m \geq 2, n \geq 1$ ,  $TR(\mathbf{G}(m, n))$  is closed under union, tree concatenation, linear non-deleting tree homomorphism, and intersection with regular tree languages, and  $TR(\mathbf{G}(2, n))$  is also closed under the tree analogue of the Kleene star operation.<sup>3</sup>

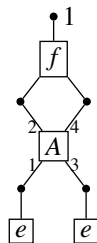
<sup>3</sup> The tree versions of concatenation and Kleene star are called "concatenation" and "closure" in Comon et al. 2007 and "z-product" and "z-iteration" in Gécseg and Steinby 1997.

## 2.4 Hyperedge Replacement Grammars

For the most part, our presentation of hyperedge replacement grammars follows Engelfriet and Heyker (1991) and Engelfriet (1997).<sup>4</sup>

For any set  $V$ , we let  $V^*$  denote the set of finite sequences of elements of  $V$ . The length of a finite sequence  $\mathbf{v} = (v_1, \dots, v_n)$  is  $|\mathbf{v}| = n$ . We use the symbol  $\hat{\phantom{v}}$  for concatenation of sequences: if  $\mathbf{v} = (v_1, \dots, v_n)$  and  $\mathbf{u} = (u_1, \dots, u_m)$ , then  $\mathbf{v}\hat{\mathbf{u}} = (v_1, \dots, v_n, u_1, \dots, u_m)$ . A *hypergraph* over a ranked alphabet  $\mathcal{A}$  is a tuple  $H = (V, E, \text{nod}, \text{lab}, \text{ext})$ , where  $V$  and  $E$  are disjoint finite sets,  $\text{nod}: E \rightarrow V^*$ ,  $\text{lab}: E \rightarrow \mathcal{A}$ , and  $\text{ext} \in V^*$ . Elements of  $V$  and  $E$  are nodes and *hyperedges*, respectively,  $\text{nod}$  is the *incidence function*,  $\text{lab}$  is the *labeling function*, and  $\text{ext}$  is the sequence of *external nodes*. It is required that for every  $e \in E$ ,  $\text{rank}_{\mathcal{A}}(\text{lab}(e)) = |\text{nod}(e)|$ . If  $\text{nod}(e) = (v_1, \dots, v_m)$ , then we write  $\text{nod}(e, i)$  for  $v_i$ , and if  $\text{ext} = (v_1, \dots, v_m)$ , we write  $\text{ext}(i)$  for  $v_i$ . The *type* of  $H$  is defined as  $\text{type}(H) = |\text{ext}|$ . We often refer to the components of  $H$  as  $V_H, E_H, \text{nod}_H, \text{lab}_H, \text{ext}_H$ . It is customary to identify hypergraphs that are isomorphic to each other.

We depict hypergraphs as in the following diagram:



In a diagram like this, dots represent nodes, and dots with numbers attached are external nodes. Hyperedges are represented by boxes with labels inside and *tentacles* connecting them to the nodes that they are incident on. We adopt the convention that the tentacles are ordered counterclockwise starting from the 9 o'clock position, unless otherwise indicated, as with the  $A$ -labeled hyperedge in this example, where the order is indicated by numbers in small type. The above example is a hypergraph over a ranked alphabet  $\mathcal{A}$ , where  $\text{rank}_{\mathcal{A}}(f) = 3$ ,  $\text{rank}_{\mathcal{A}}(A) = 4$ ,  $\text{rank}_{\mathcal{A}}(e) = 1$ .

The *string graph* of a string  $a_1 \dots a_n$  is  $\text{sgr}(a_1 \dots a_n) = (V, E, \text{nod}, \text{lab}, \text{ext})$ , where

$$\begin{aligned} V &= \{0, \dots, n\}, \\ E &= \{e_1, \dots, e_n\}, \\ \text{nod}(e_i) &= (i-1, i) \quad \text{for } i = 1, \dots, n, \\ \text{lab}(e_i) &= a_i, \\ \text{ext} &= (0, n). \end{aligned}$$

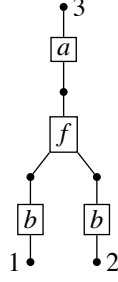
<sup>4</sup> Another introductory article on hyperedge replacement grammars is Drewes et al. 1997. Bauderon and Courcelle (1987) define hyperedge replacement grammars in an alternative style using systems of equations.

A tree  $fT_1 \dots T_n \in \mathbb{T}_A$  is represented by a *tree graph* of type 1, namely,  $\text{gr}(fT_1 \dots T_n) = (V, E, \text{nod}, \text{lab}, \text{ext})$ , where

$$\begin{aligned} V &= \{v_0\} \cup \bigcup_{i=1}^n V_{\text{gr}(T_i)}, \\ E &= \{e_0\} \cup \bigcup_{i=1}^n E_{\text{gr}(T_i)}, \\ \text{nod}(e) &= \begin{cases} (v_1, \dots, v_n, v_0) & \text{if } e = e_0, \text{ where } \text{ext}_{\text{gr}(T_i)} = (v_i) \text{ for } i = 1, \dots, n, \\ \text{nod}_{\text{gr}(T_i)}(e) & \text{if } e \in E_{\text{gr}(T_i)}, \end{cases} \\ \text{lab}(e) &= \begin{cases} f & \text{if } e = e_0, \\ \text{lab}_{\text{gr}(T_i)}(e) & \text{if } e \in E_{\text{gr}(T_i)}, \end{cases} \\ \text{ext} &= (v_0). \end{aligned}$$

In the definition of  $V$  and  $E$ , the union is assumed to be over  $n + 1$  disjoint sets. If  $T \in \mathbb{T}_A$ , then  $\text{gr}(T)$  is a hypergraph over  $\text{inc}(A)$ .

The tree graph  $\text{gr}(T)$  of type  $k + 1$  representing a simple tree  $T$  in  $\mathbb{T}_A(Y_k)$  is defined similarly, except that there are no edges corresponding to the nodes labeled with  $y_1, \dots, y_k$ , and these nodes followed by the root node constitute the list of external nodes. For example, the tree  $a(f(by_1)(by_2))$  is represented by the following hypergraph:



Let  $H = (V_H, E_H, \text{nod}_H, \text{lab}_H, \text{ext}_H)$  be a hypergraph and  $R \subseteq V_H \times V_H$ . The result of identifying every pair of nodes  $(v, v') \in R$  in  $H$  is defined as  $H/R = (V_H/\equiv_R, E_H, \text{nod}, \text{lab}_H, \text{ext})$ , where  $\equiv_R$  is the smallest equivalence relation on  $V_H$  extending  $R$ ,  $\text{nod}(e, i) = [\text{nod}_H(e, i)]_{\equiv_R}$ , and  $\text{ext}(i) = [\text{ext}_H(i)]_{\equiv_R}$ . Note that if we think of the nodes in  $V_H$  as drawn from some universal set  $U$  and a mapping  $\sigma$  on  $U$  is a most general unifier of the pairs in  $R$ , then  $H/R$  is isomorphic to  $H' = (V', E_H, \text{nod}', \text{lab}_H, \text{ext}')$ , where  $V' = \{v\sigma \mid v \in V_H\}$ ,  $\text{nod}'(e) = \text{nod}_H(e)\sigma$  and  $\text{ext}' = \text{ext}_H\sigma$ .

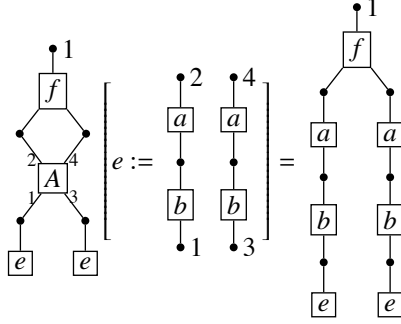
Let  $K = (V_K, E_K, \text{nod}_K, \text{lab}_K, \text{ext}_K)$  and  $H = (V_H, E_H, \text{nod}_H, \text{lab}_H, \text{ext}_H)$  be hypergraphs and let  $e \in E_K$  be such that  $|\text{nod}_K(e)| = \text{type}(H)$ . Then the result of substituting  $H$  for  $e$  in  $K$  is  $K[e := H] = \bar{K}/R$ , where

$$\begin{aligned} \bar{K} &= (V, E, \text{nod}, \text{lab}, \text{ext}), \\ V &= V_K \cup V_H, \\ E &= (E_K - \{e\}) \cup E_H, \end{aligned}$$

$$\begin{aligned}
\text{nod} &= (\text{nod}_K \cup \text{nod}_H) \upharpoonright E, \\
\text{lab} &= (\text{lab}_K \cup \text{lab}_H) \upharpoonright E, \\
\text{ext} &= \text{ext}_K, \\
R &= \{ (\text{nod}_K(e, i), \text{ext}_H(i)) \mid 1 \leq i \leq \text{type}(H) \}.
\end{aligned}$$

In the definition of  $V$  and  $E$ , it is assumed that  $V_K \cap V_H = E_K \cap E_H = \emptyset$  (taking isomorphic copies when necessary).

For example, we have



where  $e$  is the  $A$ -labeled hyperedge in the leftmost hypergraph.

If  $e_1$  and  $e_2$  are distinct hyperedges of  $K$ , one can show that  $K[e_1 := H_1][e_2 := H_2] = K[e_2 := H_2][e_1 := H_1]$ . The simultaneous substitution  $K[e_1 := H_1, \dots, e_m := H_m]$  is defined as  $K[e_1 := H_1] \dots [e_m := H_m]$ .

A *hyperedge replacement grammar* is a tuple  $G = (N, \Delta, P, S)$ , where  $N$  and  $\Delta$  are disjoint ranked alphabets,  $S \in N$ , and  $P$  is a finite set of *productions*, each of the form

$$B \rightarrow H,$$

where  $B \in N$ ,  $H$  is a hypergraph over  $N \cup \Delta$ , and  $\text{type}(H) = \text{rank}_N(B)$ . Elements of  $N$  and  $\Delta$  are called *nonterminals* and *terminals*, respectively, and  $S$  is called the *initial nonterminal*. If  $H$  is a hypergraph over  $N \cup \Delta$ ,  $\text{nont}(H)$  is the list consisting of the hyperedges in  $H$  labeled with nonterminals, in some fixed order.

We associate with  $G$  a second-order signature  $\Sigma_G = (N, P, \tau)$ , where  $\tau(\pi) = B_1 \rightarrow \dots \rightarrow B_n \rightarrow B$  if  $\pi = B \rightarrow H$  and  $\text{nont}(H) = (e_1, \dots, e_n)$  with  $\text{lab}(e_i) = B_i$ .<sup>5</sup> A *derivation tree* of  $G$  is a  $\beta$ -normal closed  $\lambda$ -term over  $\Sigma_G$  of atomic type. A derivation tree  $T$  is *complete* if it is of type  $S$ . If  $\pi = B \rightarrow H$ ,  $\text{nont}(H) = (e_1, \dots, e_n)$ , and  $T = \pi T_1 \dots T_n$  is a derivation tree of  $G$ , then

$$\text{yield}_G(T) = H[e_1 := \text{yield}_G(T_1), \dots, e_n := \text{yield}_G(T_n)].$$

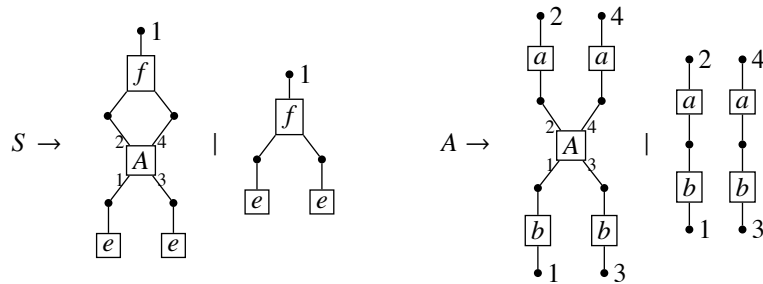
The language of  $G$  is defined as

$$L(G) = \{ \text{yield}_G(T) \mid T \text{ is a complete derivation tree of } G \}.$$

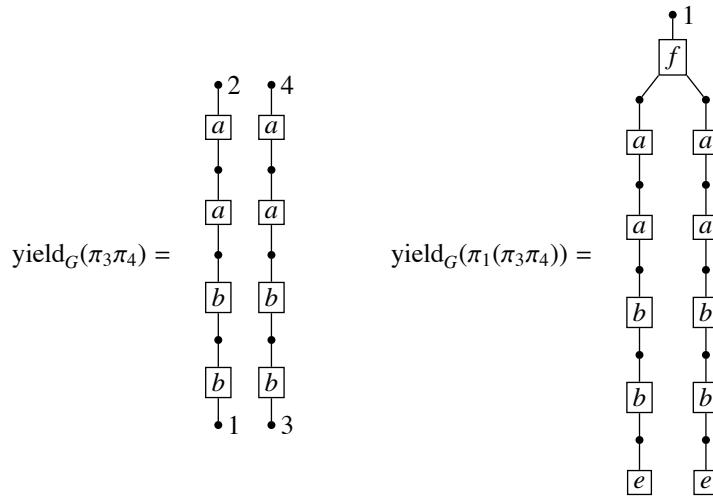
<sup>5</sup> A more standard treatment uses a context-free grammar rather than a second-order signature, associating  $\pi \in P$  with a CFG production  $B \rightarrow B_1 \dots B_n$ .

The notation  $STR(HR)$  denotes the class of all string languages  $L$  such that there exists a hyperedge replacement grammar that generates  $\{sgr(w) \mid w \in L\}$ . Similarly,  $TR(HR)$  is the class of tree languages  $L$  such that there exists a hyperedge replacement grammar that generates  $\{gr(T) \mid T \in L\}$ .

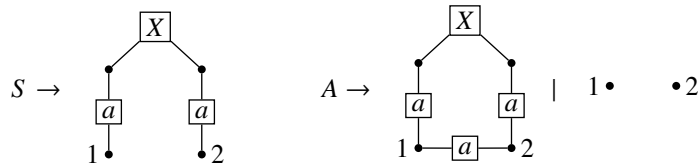
*Example 2.* The following grammar  $G$ , taken from Engelfriet and Maneth (2000), with the ranked alphabet of nonterminals  $N = N^{(1)} \cup N^{(4)} = \{S\} \cup \{A\}$ , generates  $\{gr(f(a^n(b^n e))(a^n(b^n e))) \mid n \geq 0\}$ . Vertical bars in productions separate alternative right-hand sides, as is customary with context-free grammars.



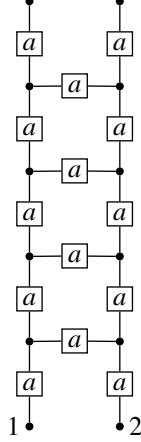
Let us name the four productions  $\pi_1, \pi_2, \pi_3, \pi_4$  (from left to right). Then, for example,  $\pi_1(\pi_3\pi_4)$  is a complete derivation tree, and we have



*Example 3.* Consider the following grammar (from Engelfriet and Heyker 1991):



This grammar generates hypergraphs like the following (“ladders”):



The language generated by a hyperedge replacement grammar always belongs to NP, and is sometimes NP-complete. For a certain special class of hyperedge replacement grammars, however, the generated language belongs to LOGCFL (Laute- mann 1990). This includes the case where the generated language consists of connected hypergraphs of bounded degree, which implies that both  $STR(HR)$  and  $TR(HR)$  lie in- side LOGCFL.

Engelfriet and Heyker (1991) showed that  $STR(HR)$  equals the class of output lan- guages of deterministic tree-walking transducers, and Engelfriet and Maneth (2000) characterized  $TR(HR)$  as the class of output tree languages of finite-copying macro tree transducers which are linear and non-deleting in the parameters.

### 3 Linear $\lambda$ -Terms as Hypergraphs

The key to encoding second-order ACGs with hyperedge replacement grammars is the representation of linear  $\lambda$ -terms as hypergraphs.

If  $\alpha$  is a type, we let  $\bar{\alpha}$  denote the sequence of atomic types occurring in  $\alpha$ , listed from left to right (with repetition). Given a  $\lambda$ -term  $M$  over  $\Sigma = (A, C, \tau)$ , the notation  $\overrightarrow{\text{Con}}(M)$  denotes the sequence  $(d_1, \dots, d_m)$  of constants occurring in  $M$ ;  $\widehat{M}[z_1, \dots, z_m]$  is the pure  $\lambda$ -term with  $z_1, \dots, z_m$  among its free variables such that  $\widehat{M}[d_1, \dots, d_m] = M$ . Note that  $\Gamma \vdash_{\Sigma} M : \alpha$  if and only if  $z_1 : \tau(d_1), \dots, z_m : \tau(d_m), \Gamma \vdash \widehat{M}[z_1, \dots, z_m] : \alpha$ .

Let  $M$  be a  $\lambda$ -term over  $\Sigma = (A, C, \tau)$  in  $\eta$ -long form relative to a typing  $x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash_{\Sigma} M : \alpha$ . Let  $\Delta = C \cup \{x_1, \dots, x_n\}$  be the ranked alphabet with  $\text{rank}_{\Delta}(d) = |\overline{\tau(d)}|$  for  $d \in C$ , and  $\text{rank}_{\Delta}(x_i) = |\bar{\alpha}_i|$ . We associate with  $M$  a hypergraph  $\text{graph}(M)$  over  $\Delta$ . Assume that  $\overrightarrow{\text{Con}}(M) = (d_1, \dots, d_m)$ , and

$$z_1 : \beta_1, \dots, z_m : \beta_m, x_1 : \gamma_1, \dots, x_n : \gamma_n \vdash \widehat{M}[z_1, \dots, z_m] : \gamma$$

is a principal typing of  $\widehat{M}[z_1, \dots, z_m]$ . Then  $\text{graph}(M)$  is defined as follows:

$$\text{graph}(M) = (V, E, \text{nod}, \text{lab}, \text{ext}),$$

where

$$\begin{aligned}
V &= \text{the set of atomic types in } \beta_1, \dots, \beta_m, \gamma_1, \dots, \gamma_n, \gamma, \\
E &= \{f_1, \dots, f_m, e_1, \dots, e_n\}, \\
\text{nod}(f_i) &= \overline{\beta}_i, \\
\text{nod}(e_i) &= \overline{\gamma}_i, \\
\text{lab}(f_i) &= d_i, \\
\text{lab}(e_i) &= x_i, \\
\text{ext} &= \overline{\gamma}.
\end{aligned}$$

Since  $M$  is in  $\eta$ -long form relative to  $x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash_{\Sigma} M : \alpha$ , the pure  $\lambda$ -term  $\widehat{M}[z_1, \dots, z_m]$  is  $\eta$ -long relative to  $z_1 : \tau(d_1), \dots, z_m : \tau(d_m), x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash \widehat{M}[z_1, \dots, z_m] : \alpha$ , and it follows that  $|\overline{\beta}_i| = |\tau(d_i)|$  and  $|\overline{\gamma}_i| = |\overline{\alpha}_i|$ ; hence  $\text{graph}(M)$  is indeed a hypergraph over  $\Delta$ .<sup>6</sup>

*Example 4.* Let  $U$  be as in Example 1. Consider a  $\lambda$ -term

$$M = \lambda w.x(\lambda v_1 v_2.w(\lambda y.a_1(v_1(a_2y)))(\lambda y.a_3(v_2(a_4y))))$$

over  $\Sigma_U^{\text{string}}$ . We have

$$\begin{aligned}
\text{FV}(M) &= \{x\}, \\
\overrightarrow{\text{Con}}(M) &= (a_1, a_2, a_3, a_4), \\
x : (((o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o) \rightarrow o) &\vdash_{\Sigma_U^{\text{string}}} M : ((o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o) \rightarrow o.
\end{aligned}$$

Since  $a_1, a_2, a_3, a_4$  all have type  $o \rightarrow o$  in  $\Sigma_U^{\text{string}}$ ,

$$\begin{aligned}
z_1 : o \rightarrow o, z_2 : o \rightarrow o, z_3 : o \rightarrow o, z_4 : o \rightarrow o, x : (((o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o) \rightarrow o) \vdash \\
\widehat{M}[z_1, z_2, z_3, z_4] : ((o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o) \rightarrow o,
\end{aligned}$$

and a principal typing of  $\widehat{M}[z_1, z_2, z_3, z_4]$  is

$$\begin{aligned}
z_1 : q_4 \rightarrow q_3, z_2 : q_6 \rightarrow q_5, z_3 : q_8 \rightarrow q_7, z_4 : q_{10} \rightarrow q_9, x : (((q_5 \rightarrow q_4) \rightarrow (q_9 \rightarrow q_8) \rightarrow q_2) \rightarrow q_1) \vdash \\
\widehat{M}[z_1, z_2, z_3, z_4] : ((q_6 \rightarrow q_3) \rightarrow (q_{10} \rightarrow q_7) \rightarrow q_2) \rightarrow q_1,
\end{aligned}$$

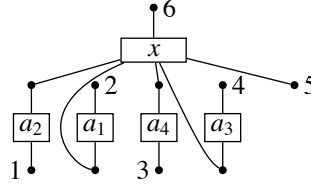
where  $q_1, \dots, q_{10}$  are distinct atomic types. Thus,  $\text{graph}(M) = (V, E, \text{nod}, \text{lab}, \text{ext})$ , where

$$\begin{aligned}
V &= \{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}\}, \\
E &= \{f_1, f_2, f_3, f_4, e\}, \\
\text{nod}(f_1) &= (q_4, q_3),
\end{aligned}$$

<sup>6</sup> This means that the definition of  $\text{graph}(M)$  does not really depend on the given typing of  $M$ , since one can always take a principal typing of  $M$ .

$$\begin{aligned}
\text{nod}(f_2) &= (q_6, q_5), \\
\text{nod}(f_3) &= (q_8, q_7), \\
\text{nod}(f_4) &= (q_{10}, q_9), \\
\text{nod}(e) &= (q_5, q_4, q_9, q_8, q_2, q_1), \\
\text{lab}(f_1) &= a_1, \\
\text{lab}(f_2) &= a_2, \\
\text{lab}(f_3) &= a_3, \\
\text{lab}(f_4) &= a_4, \\
\text{lab}(e) &= x, \\
\text{ext} &= (q_6, q_3, q_{10}, q_7, q_2, q_1).
\end{aligned}$$

The following is a diagram representing  $\text{graph}(M)$ :



If  $M$  is a  $\beta$ -normal linear  $\lambda$ -term,  $\text{graph}(M)$  has a simple inductive definition. Let  $\Sigma = (A, C, \tau)$ .

- If  $M = cM_1 \dots M_n$  is in  $\eta$ -long form relative to  $\Gamma \vdash_{\Sigma} cM_1 \dots M_n : p$  for some  $p \in A$ , then  $\text{graph}(cM_1 \dots M_n) = (V, E, \text{nod}, \text{lab}, \text{ext})$ , where

$$\begin{aligned}
V &= \{v_0\} \cup \bigcup_{i=1}^n V_{\text{graph}(M_i)}, \\
E &= \{e_0\} \cup \bigcup_{i=1}^n E_{\text{graph}(M_i)}, \\
\text{nod}(e) &= \begin{cases} \text{ext}_{\text{graph}(M_1)} \hat{\ } \dots \hat{\ } \text{ext}_{\text{graph}(M_n)} \hat{\ } (v_0) & \text{if } e = e_0, \\ \text{nod}_{\text{graph}(M_i)}(e) & \text{if } e \in E_{\text{graph}(M_i)}, \end{cases} \\
\text{lab}(e) &= \begin{cases} c & \text{if } e = e_0, \\ \text{lab}_{\text{graph}(M_i)}(e) & \text{if } e \in E_{\text{graph}(M_i)}, \end{cases} \\
\text{ext} &= (v_0).
\end{aligned}$$

(In the definitions of  $V$  and  $E$ , the union is over disjoint sets.)

- If  $M = xM_1 \dots M_n$  is in  $\eta$ -long form relative to  $\Gamma \vdash_{\Sigma} xM_1 \dots M_n : p$  for some  $p \in A$ , then  $\text{graph}(xM_1 \dots M_n) = (V, E, \text{nod}, \text{lab}, \text{ext})$ , where

$$V = \{v_0\} \cup \bigcup_{i=1}^n V_{\text{graph}(M_i)},$$

$$\begin{aligned}
E &= \{e_0\} \cup \bigcup_{i=1}^n E_{\text{graph}(M_i)}, \\
\text{nod}(e) &= \begin{cases} \text{ext}_{\text{graph}(M_1)} \hat{\ } \dots \hat{\ } \text{ext}_{\text{graph}(M_n)} \hat{\ } (v_0) & \text{if } e = e_0, \\ \text{nod}_{\text{graph}(M_i)}(e) & \text{if } e \in E_{\text{graph}(M_i)}, \end{cases} \\
\text{lab}(e) &= \begin{cases} x & \text{if } e = e_0, \\ \text{lab}_{\text{graph}(M_i)}(e) & \text{if } e \in E_{\text{graph}(M_i)}, \end{cases} \\
\text{ext} &= (v_0).
\end{aligned}$$

(In the definitions of  $V$  and  $E$ , the union is over disjoint sets.)

- If  $M = \lambda x.M_1$  is in  $\eta$ -long form relative to  $\Gamma \vdash_{\Sigma} \lambda x.M_1 : \alpha \rightarrow \beta$  and  $e$  is the unique hyperedge in  $\text{graph}(M_1)$  with label  $x$ , then  $\text{graph}(\lambda x.M_1) = (V, E, \text{nod}, \text{lab}, \text{ext})$ , where

$$\begin{aligned}
V &= V_{\text{graph}(M_1)}, \\
E &= E_{\text{graph}(M_1)} - \{e\}, \\
\text{nod} &= \text{nod}_{\text{graph}(M_1)} \upharpoonright E, \\
\text{lab} &= \text{lab}_{\text{graph}(M_1)} \upharpoonright E, \\
\text{ext} &= \text{nod}_{\text{graph}(M_1)}(e) \hat{\ } \text{ext}_{\text{graph}(M_1)}.
\end{aligned}$$

It is easy to see that if  $T$  is a tree in  $\mathbb{T}_{\Delta}$ , then  $\text{gr}(T)$  is isomorphic to  $\text{graph}(T)$ , where  $T$  is viewed as a closed  $\lambda$ -term over  $\Sigma_{\Delta}^{\text{tree}}$ . If  $T$  is a simple tree in  $\mathbb{T}_{\Delta}(Y_k)$ , then  $\text{gr}(T)$  is isomorphic to  $\text{graph}(\lambda y_1 \dots y_k.T)$ . Also, for every string  $a_1 \dots a_n$ ,  $\text{graph}(/a_1 \dots a_n/)$  is the same as  $\text{sgr}(a_n \dots a_1)$ , the string graph of the reversal of  $a_1 \dots a_n$ . Thus, the function  $\text{graph}(\cdot)$  is one-one on  $\lambda$ -terms representing trees and strings. More generally, it is one-one on linear  $\lambda$ -terms in  $\eta$ -long  $\beta$ -normal form, allowing  $\text{graph}(M)$  to be viewed as a representation of  $M$  for such  $\lambda$ -terms. This is a consequence of the *Coherence Theorem* (see Mints 2000).

**Lemma 1.** *Let  $M$  be a linear  $\lambda$ -term in  $\eta$ -long form relative to  $\Gamma \vdash_{\Sigma} M : \alpha$ . If  $M \rightarrow_{\beta} M'$ , then  $M'$  is in  $\eta$ -long form relative to  $\Gamma \vdash_{\Sigma} M' : \alpha$ , and  $\text{graph}(M)$  and  $\text{graph}(M')$  are isomorphic.*

*Proof.* It is easy to see that  $\eta$ -long form is preserved under  $\beta$ -reduction. Let  $(d_1, \dots, d_m) = \overrightarrow{\text{Con}}(M)$ . Since  $M$  is linear,  $M$  reduces to  $M'$  by non-erasing non-duplicating  $\beta$ -reduction. This implies that for some permutation  $\rho$  of  $\{1, \dots, m\}$ ,  $\overrightarrow{\text{Con}}(M') = (d_{\rho(1)}, \dots, d_{\rho(m)})$  and  $\widehat{M}[z_1, \dots, z_m] \rightarrow_{\beta} \widehat{M}'[z_{\rho(1)}, \dots, z_{\rho(m)}]$  by non-erasing non-duplicating  $\beta$ -reduction. By the Subject Reduction and Subject Expansion Theorems,  $\widehat{M}[z_1, \dots, z_m]$  and  $\widehat{M}'[z_{\rho(1)}, \dots, z_{\rho(m)}]$  share the same principal typing. It follows that  $\text{graph}(M)$  and  $\text{graph}(M')$  are isomorphic.  $\square$

**Lemma 2.** *Let  $M, N$  be linear  $\lambda$ -terms over  $\Sigma$  with  $\text{FV}(M) \cap \text{FV}(N) = \emptyset$ , and suppose that  $M$  and  $N$  are in  $\eta$ -long form relative to  $\Gamma, x : \beta \vdash_{\Sigma} M : \alpha$  and  $\Delta \vdash_{\Sigma} N : \beta$ , respectively. Then  $M[x := N]$  is in  $\eta$ -long form relative to  $\Gamma, \Delta \vdash M[x := N] : \alpha$ , and*

$$\text{graph}(M[x := N]) = \text{graph}(M)[e := \text{graph}(N)],$$

where  $e$  is the unique hyperedge of  $\text{graph}(M)$  with label  $x$ .

*Proof.* Again, it is easy to see that  $P = M[x := N]$  is in  $\eta$ -long form. Let

$$\begin{aligned} \Phi, x : \delta \vdash \widehat{M}[z] : \gamma \\ \Psi \vdash \widehat{N}[z'] : \delta' \end{aligned}$$

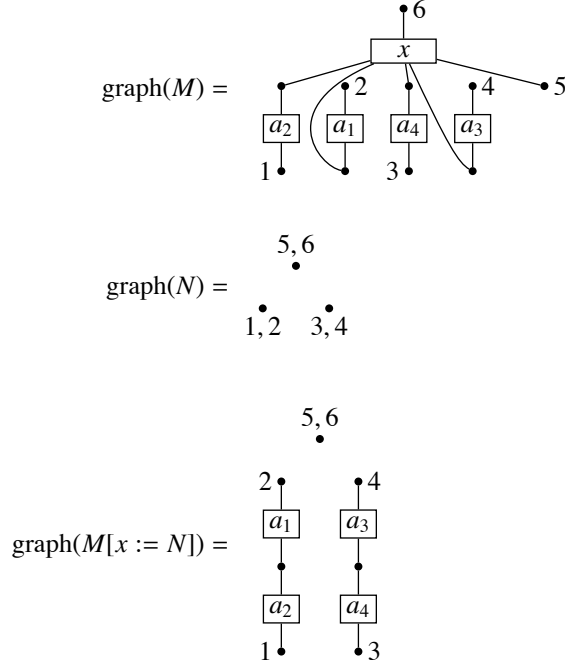
be principal typings of  $\widehat{M}[z]$  and  $\widehat{N}[z']$ , respectively. Then it is clear that

$$(\Phi, \Psi)\sigma \vdash \widehat{P}[z''] : \gamma\sigma$$

is a principal typing of  $\widehat{P}[z'']$ , where  $\sigma$  is a most general unifier of the pair  $(\delta, \delta')$  and  $z''$  is some permutation of  $z \widehat{z}'$ . Then the lemma is immediate from the definition of hyperedge replacement.  $\square$

*Example 5.* Let  $M$  be as in Example 4, and let  $N = \lambda w.w(\lambda y.y)(\lambda y.y)$ . We have

$$M[x := N] = \lambda w.w(\lambda y.a_1(a_2y))(\lambda y.a_3(a_4y))$$



We see that it indeed holds that  $\text{graph}(M[x := N]) = \text{graph}(M)[e := \text{graph}(N)]$ , where  $e$  is the  $x$ -labeled hyperedge of  $\text{graph}(M)$ .

## 4 From Second-Order ACGs to Hyperedge Replacement Grammars

In this section, we show that any second-order ACG can be simulated by a hyperedge replacement grammar, using the lemmas from the previous section.

Let  $\mathcal{G} = (\Sigma, \Sigma', \mathcal{L}, s)$  be a second-order ACG with  $\Sigma = (A, C, \tau)$  and  $\Sigma' = (A', C', \tau')$ . We assume that for all  $c \in C$ ,  $\mathcal{L}(c)$  is in  $\eta$ -long form relative to  $\mathcal{L}(\tau(c))$ , so that every  $M \in \mathcal{O}(\mathcal{G})$  is in  $\eta$ -long  $\beta$ -normal form relative to  $\mathcal{L}(s)$ . In addition, we assume that for all  $c, c' \in C$ ,  $\tau(c) = \tau(c')$  and  $\mathcal{L}(c) = \mathcal{L}(c')$  imply  $c = c'$ .

We associate with  $\mathcal{G}$  a certain hyperedge replacement grammar  $\text{hr}(\mathcal{G})$ , where each abstract constant  $c$  of  $\mathcal{G}$  corresponds to a production  $\pi_c$  of  $\text{hr}(\mathcal{G})$ . If  $\tau(c) = p_1 \rightarrow \dots \rightarrow p_n \rightarrow p_0$ , then we define

$$M_c = \mathcal{L}(c)p_1 \dots p_n. \quad (1)$$

Note that  $M_c$  is a closed  $\lambda$ -term over  $\Sigma'' = (A', C' \cup A, \tau'')$ , where  $\tau''(c) = \tau'(c)$  for  $c \in C'$  and  $\tau''(p) = \mathcal{L}(p)$  for  $p \in A$ . Since  $\mathcal{L}(c)$  is in  $\eta$ -long form relative to  $\mathcal{L}(\tau(c))$ , it is easy to see that  $|M_c|_\beta$  is in  $\eta$ -long form relative to  $\mathcal{L}(p_0)$ , and  $\text{graph}(|M_c|_\beta)$  is a hypergraph over the ranked alphabet  $C' \cup A$ , where the rank of  $d \in C'$  is  $|\tau'(d)|$  and the rank of  $p \in A$  is  $|\mathcal{L}(p)|$ . Let  $\pi_c$  be the production

$$p_0 \rightarrow \text{graph}(|M_c|_\beta).$$

The hyperedge replacement grammar associated with  $\mathcal{G}$  is  $\text{hr}(\mathcal{G}) = (A, C', P, s)$ , where  $C \cup A'$  is the ranked alphabet defined above, and

$$P = \{\pi_c \mid c \in C\}.$$

Note that the mapping  $c \mapsto \pi_c$  induces an isomorphism from  $\Sigma$  to  $\Sigma_{\text{hr}(\mathcal{G})}$ . Thus, we shall identify the closed  $\beta$ -normal  $\lambda$ -terms over  $\Sigma$  of atomic type with the derivation trees of  $\text{hr}(\mathcal{G})$ , and write  $c$  for  $\pi_c$  in derivation trees of  $\text{hr}(\mathcal{G})$ . In particular, the elements of the abstract language of  $\mathcal{G}$  are identical to the complete derivation trees of  $\text{hr}(\mathcal{G})$  under this convention.

*Example 6.* If we apply the above construction to the second-order ACG in Example 1, we obtain the following hyperedge replacement grammar:

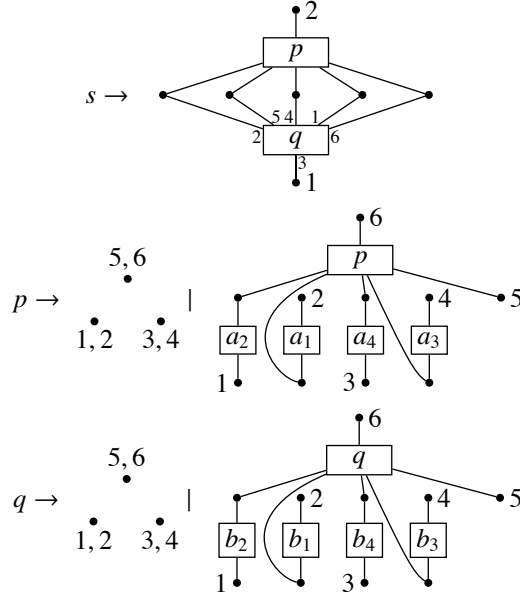


Figure 1 shows a complete derivation tree of this hyperedge replacement grammar. This derivation tree corresponds to the element  $c_1(c_4(c_4c_2))(c_5c_3)$  of the abstract language of the original ACG and yields  $\text{sgr}(b_4b_3a_4a_4a_3a_3b_2b_1a_2a_2a_1a_1) = \text{graph}(/a_1a_1a_2a_2b_1b_2a_3a_3a_4a_4b_3b_4/)$ .

**Lemma 3.** *Let  $\mathcal{G} = (\Sigma, \Sigma', \mathcal{L}, \tau)$  be a second-order ACG. For every closed  $\lambda$ -term  $T$  over  $\Sigma$  of atomic type,  $\text{yield}_{\text{hr}(\mathcal{G})}(T) = \text{graph}(\mathcal{L}(T))$ .*

*Proof.* Induction on  $T$ . Suppose that  $T = cT_1 \dots T_n$ ,  $\tau(c) = p_1 \rightarrow \dots \rightarrow p_n \rightarrow p$ , and  $\text{yield}_{\text{hr}(\mathcal{G})}(T_i) = \text{graph}(\mathcal{L}(T_i))$  for  $i = 1, \dots, n$ . Since  $\mathcal{L}(c)$  is in  $\eta$ -long form relative to  $\mathcal{L}(p_1 \rightarrow \dots \rightarrow p_n \rightarrow p)$ ,

$$\mathcal{L}(c) = \lambda x_1 \dots x_n. N_c$$

for some  $N_c \in \Lambda_{\text{lin}}(\Sigma')$ . Then

$$\mathcal{L}(T) \rightarrow_{\beta} N_c[x_1 := \mathcal{L}(T_1), \dots, x_n := \mathcal{L}(T_n)]$$

and by Lemma 1,

$$\text{graph}(\mathcal{L}(T)) = \text{graph}(N_c[x_1 := \mathcal{L}(T_1), \dots, x_n := \mathcal{L}(T_n)]). \quad (2)$$

Note that  $\text{graph}(|M_c|_{\beta})$  and  $\text{graph}(N_c)$  are isomorphic, except for the labels  $p_1, \dots, p_n$  vs.  $x_1, \dots, x_n$ . For  $i = 1, \dots, n$ , let  $e'_i$  be the unique hyperedge in  $\text{graph}(N_c)$  with label  $x_i$ , and let  $e_i$  be the corresponding hyperedge in  $\text{graph}(|M_c|_{\beta})$ . We get

$$\begin{aligned} \text{graph}(\mathcal{L}(T)) &= \text{graph}(N_c[e'_1 := \text{graph}(\mathcal{L}(T_1)), \dots, e'_n := \text{graph}(\mathcal{L}(T_n))]) \\ &\quad \text{by (2) and Lemma 2,} \\ &= \text{graph}(|M_c|_{\beta}[e_1 := \text{graph}(\mathcal{L}(T_1)), \dots, e_n := \text{graph}(\mathcal{L}(T_n))]) \\ &= \text{graph}(|M_c|_{\beta}[e_1 := \text{yield}_{\text{hr}(\mathcal{G})}(T_1), \dots, e_n := \text{yield}_{\text{hr}(\mathcal{G})}(T_n)]) \\ &\quad \text{by induction hypothesis,} \\ &= \text{yield}_{\text{hr}(\mathcal{G})}(cT_1 \dots T_n) \\ &\quad \text{by the definition of } \text{hr}(\mathcal{G}), \\ &= \text{yield}_{\text{hr}(\mathcal{G})}(T). \quad \square \end{aligned}$$

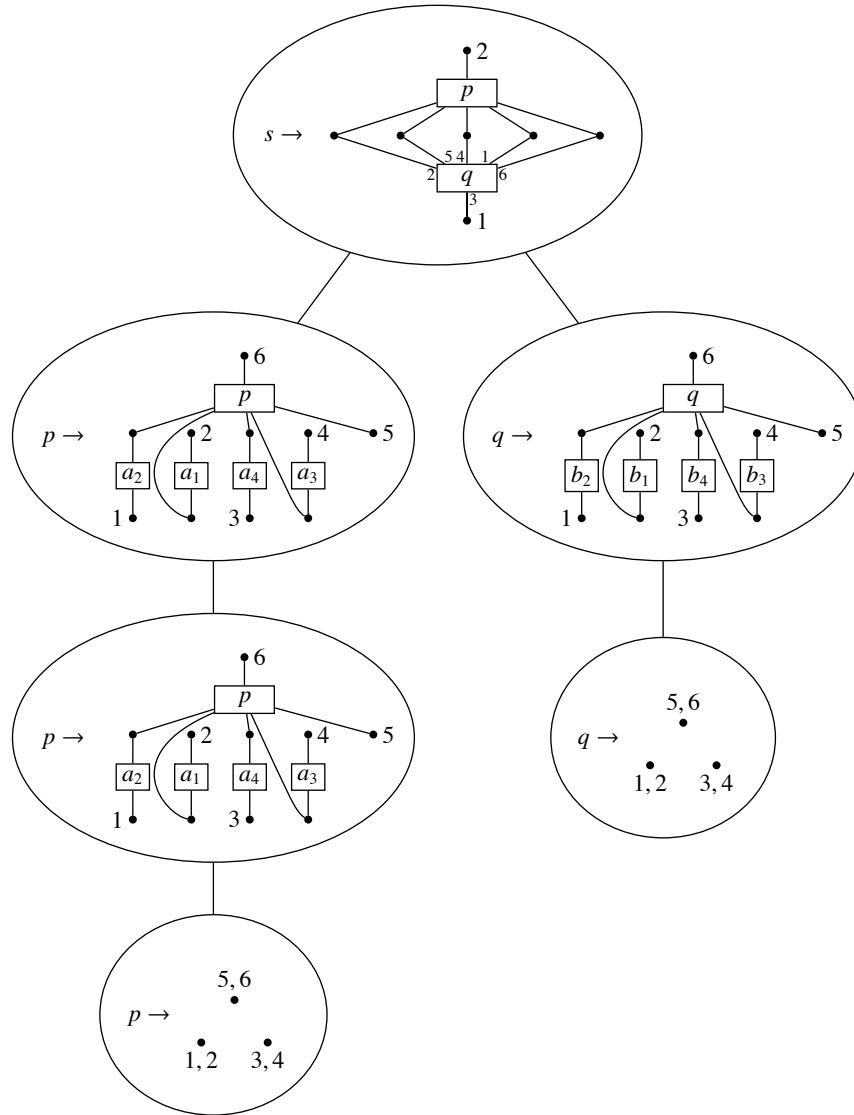
**Theorem 4.** *For every second-order ACG  $\mathcal{G}$ , we have*

$$L(\text{hr}(\mathcal{G})) = \{ \text{graph}(M) \mid M \in \mathcal{O}(\mathcal{G}) \}.$$

*Proof.* By Lemma 3,  $L(\text{hr}(\mathcal{G})) = \{ \text{graph}(\mathcal{L}(T)) \mid T \in \mathcal{A}(\mathcal{G}) \}$ . If  $M = | \mathcal{L}(T) |_{\beta}$ , then by Lemma 1,  $\text{graph}(M)$  is isomorphic to  $\text{graph}(\mathcal{L}(T))$ .  $\square$

**Corollary 5.** (i)  $TR(\mathbf{G}(2)) \subseteq TR(HR)$ .  
(ii)  $STR(\mathbf{G}(2)) \subseteq STR(HR)$ .

*Proof.* For (i), note that if  $T$  is a tree,  $\text{graph}(T)$  is isomorphic to  $\text{gr}(T)$ . For (ii), if  $w$  is a string,  $\text{graph}(/w/)$  is isomorphic to  $\text{sgr}(w^{\text{R}})$ , where  $w^{\text{R}}$  is the reversal of  $w$ . Since it is easy to see that  $STR(HR)$  is closed under reversal, the statement follows.  $\square$



**Fig. 1.** A complete derivation tree of the hyperedge replacement grammar constructed from the ACG in Example 1.

## 5 From Hyperedge Replacement Grammars to Second-Order ACGs

It is known that  $STR(HR)$  exactly equals the class of multiple context-free languages (or, equivalently, LCFRS languages). This follows from the results by Engelfriet and Heyker (1991) and by Weir (1992) concerning deterministic tree-walking transducers, but an outline of a more direct proof is given by Engelfriet (1997). Since de Groote and Pogodalla (2004) show that the LCFRSs can be encoded by second-order ACGs in  $\mathbf{G}(2, 4)$ , we have

**Theorem 6.**  $STR(HR) \subseteq STR(\mathbf{G}(2, 4))$ .

By Corollary 5 (ii) and Theorem 6, we obtain

**Corollary 7.**  $STR(\mathbf{G}(2)) = STR(HR)$ .

**Corollary 8 (Salvati).**  $STR(\mathbf{G}(2)) = STR(\mathbf{G}(2, 4))$ .

The equivalence in string generating power between second-order ACGs and LCFRSs, and consequently Corollary 8, were first obtained by Salvati (2007), using Weir’s (1992) result. The proof provided here avoids detour through deterministic tree-walking transducers, using a simple and direct translation from second-order ACGs to hyperedge replacement grammars.<sup>7</sup>

As for tree generating power, we can use the tree generating normal form for hyperedge replacement grammars due to Engelfriet and Maneth (2000).

A *forest* (i.e., sequence of trees)  $T_1, \dots, T_k$ , where each  $T_i$  is a simple tree in  $\mathbb{T}_\Delta(Y_{r_i})$  for some  $r_i$ , is represented by the *forest graph*  $\text{gr}(T_1, \dots, T_k) = \text{gr}(T_1) \oplus \dots \oplus \text{gr}(T_k)$  of type  $\sum_{i=1}^k r_i$ , where  $\oplus$  denotes disjoint union, concatenating the sequences of external nodes. A *linked ranked alphabet* is a ranked alphabet  $\Delta$  together with a mapping  $f \in \Delta^{(k)} \mapsto \text{link}(f) = (r_1, \dots, r_n)$  such that  $\sum_{i=1}^n r_i = k$ . If  $H$  is a hypergraph over a linked ranked alphabet  $\Delta$ ,  $\text{cut}(H)$  is the result of replacing every hyperedge  $e$  with label  $f$ , where  $\text{link}(f) = (r_1, \dots, r_n)$ , by distinct new hyperedges  $e_1, \dots, e_n$  such that  $e_j$  is incident on the  $(\sum_{i=1}^{j-1} r_i) + 1$ -th up to the  $\sum_{i=1}^j r_i$ -th node of  $e$  and has any label with rank  $r_j$ . A hypergraph  $H$  over a linked alphabet  $\Delta$  is a *linked forest* of type  $(r_1, \dots, r_k)$  if  $\text{cut}(H) = H_1 \oplus \dots \oplus H_k$  for some tree graphs  $H_1, \dots, H_k$  of type  $r_1, \dots, r_k$ , respectively. A hyperedge replacement grammar  $G = (N, \Delta, S, P)$  is in *tree generating normal form* if (i)  $N \cup \Delta$  is a linked ranked alphabet such that  $S \in N^{(1)}$  and for every  $f \in \Delta^{(k)}$ ,  $\text{link}(f) = (k)$ , and (ii) for every production  $B \rightarrow H$  in  $P$ ,  $H$  is a linked forest over  $N \cup \Delta$  of type  $\text{link}(B)$ . The grammar in Example 2 is a hyperedge replacement grammar in tree generating normal form, where  $\text{link}(A) = (2, 2)$ .

**Theorem 9 (Engelfriet and Maneth).** *For every tree generating hyperedge replacement grammar  $G$ , there is a hyperedge replacement grammar  $G'$  in tree generating normal form such that  $L(G') = L(G)$ .*

<sup>7</sup> The point here is that LCFRSs, second-order ACGs, and hyperedge replacement grammars are all “context-free” formalisms, and the translations from LCFRSs to second-order ACGs and then to hyperedge replacement grammars are very straightforward. The only hard work is in the direction from hyperedge replacement grammars to LCFRSs, which uses a transformation of string generating hyperedge replacement grammars due to Habel (1992).

A hyperedge replacement grammar in tree generating normal form may be viewed as representing a “context-free” grammar such that each derivation tree yields a tuple of tree contexts (i.e., a tuple of simple trees in  $\mathbb{T}_\Delta(Y_k)$ ). Such a grammar might be called a *multiple (linear non-deleting) context-free tree grammar*, in analogy with multiple context-free grammars. A *(set-local) multi-component tree-adjoining grammar* (Weir 1988) is roughly the monadic restriction of such a grammar, which only deals with unary tree contexts (as does the grammar in Example 2).

Let  $G = (N, \Delta, P, S)$  be a hyperedge replacement grammar in tree generating normal form. Let  $H$  be a linked forest over  $N \cup \Delta$  of type  $(r_1, \dots, r_k)$  with  $\text{cut}(H) = H_1 \oplus \dots \oplus H_k$ . Let  $\text{nont}(H) = (e_1, \dots, e_n)$  and let  $e_{i,1}, \dots, e_{i,l_i}$  be the hyperedges of  $H_1 \oplus \dots \oplus H_k$  that come from  $e_i$ . Assume that the labels of  $e_{i,j}$  in  $H_1 \oplus \dots \oplus H_k$  are distinct variables  $z_{i,j}$ , and let  $Z$  be the ranked alphabet consisting of all the  $z_{i,j}$ . For  $i = 1, \dots, k$ , let  $T_i \in \mathbb{T}_{\text{dec}(\Delta \cup Z)}(Y_{r_i-1})$  be the simple tree such that  $\text{gr}(T_i) = H_i$ . Define

$$\begin{aligned} M_H &= \lambda x_1 x_2 \dots x_n w. \\ &\quad x_1(\lambda z_{1,1} \dots z_{1,l_1}. \\ &\quad x_2(\lambda z_{2,1} \dots z_{2,l_2}. \\ &\quad \vdots \\ &\quad x_n(\lambda z_{n,1} \dots z_{n,l_n}. w(\lambda y_1 \dots y_{r_1-1}. T_1) \dots (\lambda y_1 \dots y_{r_k-1}. T_k)) \dots)). \end{aligned}$$

$M_H$  is a closed linear  $\lambda$ -term over  $\Sigma_\Delta^{\text{tree}}$  and has the following type:

$$\begin{aligned} &(((o^{r_{1,1}-1} \rightarrow o) \rightarrow \dots \rightarrow (o^{r_{1,l_1}-1} \rightarrow o) \rightarrow o) \rightarrow o) \rightarrow \\ &\quad \vdots \\ &(((o^{r_{n,1}-1} \rightarrow o) \rightarrow \dots \rightarrow (o^{r_{n,l_n}-1} \rightarrow o) \rightarrow o) \rightarrow o) \rightarrow \\ &\quad ((o^{r_1-1} \rightarrow o) \rightarrow \dots \rightarrow (o^{r_k-1} \rightarrow o) \rightarrow o) \rightarrow o, \end{aligned}$$

where  $(r_{i,1}, \dots, r_{i,l_i}) = \text{link}(\text{lab}_H(e_i))$  for  $i = 1, \dots, n$ .

We now describe how to represent  $G$  by a second-order ACG  $\text{acg}(G) = (\Sigma, \Sigma_{\text{dec}(\Delta)}^{\text{tree}}, \mathcal{L}, s)$ . The construction is similar to the encoding of multiple context-free grammars by de Groote and Pogodalla (2004). The abstract vocabulary of  $\text{acg}(G)$  is  $\Sigma = (N \cup \{s\}, P \cup \{d\}, \tau)$ , where  $(N, P, \tau \upharpoonright P)$  is  $\Sigma_G$ , the second-order signature associated with  $G$ , and  $\tau(d) = S \rightarrow s$ . The lexicon  $\mathcal{L}$  is defined as follows. First,  $\mathcal{L}(s) = o$ . For a nonterminal  $B \in N$  with  $\text{link}(B) = (r_1, \dots, r_k)$ , we let

$$\mathcal{L}(B) = ((o^{r_1-1} \rightarrow o) \rightarrow \dots \rightarrow (o^{r_k-1} \rightarrow o) \rightarrow o) \rightarrow o.$$

For a production  $\pi = B \rightarrow H$  in  $P$ , we let

$$\mathcal{L}(\pi) = M_H.$$

Finally, we let  $\mathcal{L}(d) = \lambda x. x(\lambda y. y)$ . We can see that  $\mathcal{L}$  is indeed a lexicon:

$$\vdash_{\Sigma_{\text{dec}(\Delta)}^{\text{tree}}} \mathcal{L}(c) : \mathcal{L}(\tau(c))$$

for every  $c \in P \cup \{d\}$ . Note that  $\text{ord}(\mathcal{L}(B)) \leq 4$  for every  $B \in N \cup \{s\}$ , so  $\text{acg}(G) \in \mathbf{G}(2, 4)$ .

Clearly, the derivation trees of  $G$  can be identified with the closed  $\lambda$ -terms over  $\Sigma$  of type  $B \in N$ .

*Example 7.* Let  $G$  be the hyperedge replacement grammar in Example 2. Then  $\text{acg}(G) = (\Sigma, \Sigma_{\mathcal{A}}^{\text{tree}}, \mathcal{L}, s)$ , where  $\mathcal{A} = \mathcal{A}^{(0)} \cup \mathcal{A}^{(1)} \cup \mathcal{A}^{(2)} = \{e\} \cup \{a, b\} \cup \{f\}$  and

$$\begin{aligned} \Sigma &= (\{S, A, s\}, \{\pi_1, \pi_2, \pi_3, \pi_4, d\}, \tau), \\ \tau(\pi_1) &= A \rightarrow S, \quad \tau(\pi_2) = S, \quad \tau(\pi_3) = A \rightarrow A, \quad \tau(\pi_4) = A, \quad \tau(d) = S \rightarrow s, \\ \mathcal{L}(S) &= (o \rightarrow o) \rightarrow o, \quad \mathcal{L}(A) = ((o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o) \rightarrow o, \quad \mathcal{L}(s) = o, \\ \mathcal{L}(\pi_1) &= \lambda x w. x(\lambda z_1 z_2. w(f(z_1 e)(z_2 e))), \\ \mathcal{L}(\pi_2) &= \lambda w. w(f e e), \\ \mathcal{L}(\pi_3) &= \lambda x w. x(\lambda z_1 z_2. w(\lambda y. a(z_1(b y)))(\lambda y. a(z_2(b y)))), \\ \mathcal{L}(\pi_4) &= \lambda w. w(\lambda y. a(b y))(\lambda y. a(b y)), \\ \mathcal{L}(d) &= \lambda x. x(\lambda y. y). \end{aligned}$$

For example,  $d(\pi_1(\pi_3\pi_4)) \in \mathcal{A}(\text{acg}(G))$ , and we have

$$\begin{aligned} \mathcal{L}(\pi_3\pi_4) &\rightarrow_{\beta} \lambda w. w(\lambda y. a(a(b(b y))))(\lambda y. a(a(b(b y))))), \\ \mathcal{L}(\pi_1(\pi_3\pi_4)) &\rightarrow_{\beta} \lambda w. w(f(a(a(b(b e))))(a(a(b(b e))))), \\ \mathcal{L}(d(\pi_1(\pi_3\pi_4))) &\rightarrow_{\beta} f(a(a(b(b e))))(a(a(b(b e)))). \end{aligned}$$

Note that if  $H$  is a hypergraph,  $H \oplus \text{gr}(y_1)$  is obtained from  $H$  by adding one new node and counting it as external nodes twice:  $H \oplus \text{gr}(y_1) = (V_H \cup \{v\}, E_H, \text{nod}_H, \text{lab}_H, \text{ext}_H^{\wedge}(v, v))$ .

**Lemma 10.** *Let  $G$  be a hyperedge replacement grammar in tree generating normal form and  $\mathcal{L}$  be the lexicon of  $\text{acg}(G)$ . If  $T$  is a derivation tree of  $G$ , then  $\text{graph}(\mathcal{L}(T)) = \text{yield}_G(T) \oplus \text{gr}(y_1)$ .*

*Proof.* We prove the lemma by induction on  $T$ . Let  $T = \pi T_1 \dots T_n$ , where  $\pi = B \rightarrow H$ ,  $\text{nont}(H) = (e_1, \dots, e_n)$ , and  $\text{lab}_H(e_i) = B_i$ . It is easy to see that  $\text{graph}(|M_{\pi}|_{\beta})$  as defined in (1) in Section 4 is isomorphic to the graph  $(V_H \cup \{v_1, \dots, v_{n+1}\}, E_H, \text{nod}, \text{lab}_H, \text{ext})$  over  $\text{inc}(\text{inc}(N)) \cup \mathcal{A}$ , where

$$\begin{aligned} \text{nod}(e) &= \text{nod}_H(e) \quad \text{if } \text{lab}_H(e) \in \mathcal{A}, \\ \text{nod}(e_i) &= \text{nod}_H(e_i)^{\wedge}(v_{i+1}, v_i), \\ \text{ext} &= \text{ext}_H^{\wedge}(v_{n+1}, v_1). \end{aligned}$$

Then

$$\begin{aligned} \text{graph}(\mathcal{L}(T)) &= \text{graph}(|M_{\pi}|_{\beta}[e_1 := \text{graph}(\mathcal{L}(T_1)), \dots, e_n := \text{graph}(\mathcal{L}(T_n))]) \\ &\quad \text{as in the proof of Lemma 3,} \\ &= \text{graph}(|M_{\pi}|_{\beta} \\ &\quad [e_1 := \text{yield}_G(T_1) \oplus \text{gr}(y_1), \dots, e_n := \text{yield}_G(T_n) \oplus \text{gr}(y_1)]) \\ &\quad \text{by induction hypothesis,} \\ &= H[e_1 := \text{yield}_G(T_1), \dots, e_n := \text{yield}_G(T_n)] \oplus \text{gr}(y_1) \\ &\quad \text{by the above characterization of } \text{graph}(|M_{\pi}|_{\beta}), \\ &= \text{yield}_G(T) \oplus \text{gr}(y_1). \quad \square \end{aligned}$$

**Theorem 11.** *Let  $G$  be a hyperedge replacement grammar in tree generating normal form. Then  $L(G) = \{ \text{graph}(M) \mid M \in \mathcal{O}(\text{acg}(G)) \}$ .*

*Proof.* The abstract language  $\mathcal{A}(\text{acg}(G))$  of  $\text{acg}(G)$  equals  $\{ dT \mid T \text{ is a complete derivation tree of } G \}$ , so it suffices to show that for every complete derivation tree  $T$  of  $G$ , we have  $\text{yield}_G(T) = \text{graph}(\downarrow_{\beta}(\mathcal{L}(dT))\downarrow_{\beta})$ . This easily follows from Lemmas 1 and 10.  $\square$

**Corollary 12.**  $TR(HR) \subseteq TR(\mathbf{G}(2, 4))$ .

By Corollaries 5 and 12, we obtain

**Corollary 13.**  $TR(\mathbf{G}(2)) = TR(HR)$ .

**Corollary 14.**  $TR(\mathbf{G}(2)) = TR(\mathbf{G}(2, 4))$ .

By Corollaries 8 and 14, the second-order hierarchy  $\mathbf{G}(2, n)$  collapses beyond  $n = 4$  for string-generating and tree-generating ACGs. It is an open question whether this holds of second-order ACGs in general.

## 6 Conclusion

Second-order abstract categorial grammars and hyperedge replacement grammars generalize string and tree grammars with “context-free” derivations in two directions, by employing data structures that encompass strings and trees as special cases, namely linear  $\lambda$ -terms and hypergraphs. The present work shows two things. Since linear  $\lambda$ -terms can be represented by hypergraphs but not all hypergraphs can be represented by linear  $\lambda$ -terms (cf. “ladders” in Example 3), second-order ACGs are less general than hyperedge replacement grammars. However, when the generated language consists of strings or trees, both formalisms are equivalent in generating power.

Past work in formal grammar theory has shown that the class of string languages generated by second-order ACGs and hyperedge replacement grammars is remarkably robust, with a large number of formalisms with equivalent power, not all of them grammars with “context-free” derivations. The class of tree languages generated by second-order ACGs and hyperedge replacement grammars may be thought of as a similarly “natural” class in the realm of tree languages. It would be interesting to find more formalisms with equivalent tree generating power.

## References

- Bauderon, Michel and Bruno Courcelle. (1987). Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20, 83–127.
- Comon, Hubert, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. (2007). *Tree Automata Techniques and Applications*. Available online at <http://tata.gforge.inria.fr/>.
- Courcelle, B. (1987). An axiomatic definition of context-free rewriting and its application to NLC graph grammars. *Theoretical Computer Science* 55, 141–181.

- Drewes, F., H.-J. Kreowski, and A. Habel. (1997). Hyperedge replacement graph grammars. In Grzegorz Rozenberg (Ed.), *Handbook of Graph Grammars and Computing by Graph Transformation* (pp. 95–162). Singapore: World Scientific.
- Engelfriet, Joost. (1997). Context-free graph grammars. In G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages, Volume 3: Beyond Words* (pp. 125–213). Berlin: Springer.
- Engelfriet, Joost and Linda Heyker. (1991). The string generating power of context-free hypergraph grammars. *Journal of Computer and System Sciences*, 43, 328–360.
- Engelfriet, Joost and Sebastian Maneth. (2000). Tree languages generated by context-free graph grammars. In H. Ehrig et al. (Eds.), *Graph Transformation*. Lecture Notes in Computer Science, (Vol. 1764, pp. 15–29). Berlin: Springer.
- Engelfriet, J. and E. M. Schmidt. (1977). IO and OI, part I. *The Journal of Computer and System Sciences*, 15, 328–353.
- Gécseg, Ferenc and Magnus Steinby. (1997). Tree languages. In Grzegorz Rozenberg and Arto Salomaa (Eds.), *Handbook of Formal Languages, Volume 3: Beyond Words* (pp. 1–68). Berlin: Springer. 1997.
- de Groote, Philippe. (2001). Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference* (pp. 148–155).
- de Groote, Philippe and Sylvain Pogodalla. (2004). On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13, 421–438.
- Habel, Annegret. (1992). *Hyperedge Replacement: Grammars and Languages*. Berlin: Springer.
- Habel, Annegret and Hans-Jörg Kreowski. (1987). Some structural aspects of hypergraph languages generated by hyperedge replacement. In G. Goos and J. Hartmanis, editors, *STACS 87: 4th Annual Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science (Vol. 247, pp. 207–219). Berlin: Springer.
- Hindley, J. Roger. (1997). *Basic Simple Type Theory*. Cambridge: Cambridge University Press.
- Kanazawa, Makoto. (2006). Abstract families of abstract categorial languages. In G. Mints and R. de Queiroz (Eds.), Proceedings of the 13th Workshop on Logic, Language, Information and Computation (WoLLIC 2006). *Electronic Notes in Theoretical Computer Science*, 165, 65–80.
- Kanazawa, Makoto. (2007). Parsing and generation as Datalog queries. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics* (pp. 176–183).
- Kepler, Stephan and Uwe Mönnich. (2006). Closure properties of linear context-free tree languages with an application to optimality theory. *Theoretical Computer Science*, 354, 82–97.
- Lautemann, Clemens. (1990). The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, 27, 399–421.
- Mints, Grigori. (2000). *A Short Introduction to Intuitionistic Logic*. New York: Kluwer Academic/Plenum Publishers.
- Raoult, Jean-Claude. (1997). Rational tree relations. *Bulletin of the Belgian Mathematical Society*, 4, 149–176.
- Rounds, William C. (1970). Mappings and grammars on trees. *Mathematical Systems Theory*, 4, 257–287.
- Salvati, Sylvain. (2005). *Problèmes de Filtrage et Problèmes d’analyse pour les Grammaires Catégorielles Abstraites*. Doctoral thesis. Institut National Polytechnique de Lorraine.
- Salvati, Sylvain. (2007). Encoding second order string ACG with deterministic tree walking transducers. In Shuly Wintner (Ed.), *Proceedings of FG 2006: The 11th conference on Formal Grammar* (pp. 143–156). FG Online Proceedings. Stanford, CA: CSLI Publications.
- Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88, 191–229.

- Sørensen, Morten Heine and Paweł Urzyczyn. (2006). *Lectures on the Curry-Howard Isomorphism*. Amsterdam: Elsevier.
- Weir, David J. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. dissertation. University of Pennsylvania.
- Weir, David. (1992). Linear context-free rewriting systems and deterministic tree-walking transducers. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics* (pp. 136–143).
- Yoshinaka, Ryo and Makoto Kanazawa. (2005). The complexity and generative capacity of lexicalized abstract categorial grammars. In Philippe Blache, Edward Stabler, Joan Busquets, and Richard Moot (Eds.), *Logical Aspects of Computational Linguistics: 5th International Conference, LACL 2005*. Lecture Notes in Computer Science (Vol. 3492, pp. 330–346). Berlin: Springer.