Second-Order Abstract Categorial Grammars

Makoto Kanazawa

July 28, 2009

1 Some "Context-Free" Grammar Formalisms

1.1 Top-Down Rewriting and Bottom-Up Deduction

When Chomsky first introduced the *context-free grammar* (CFG), it was defined to be a restricted type of *semi-Thue system* or *unrestricted rewriting system*. An unrestricted rewriting system has rules of the form:

$$\alpha \to \beta$$
,

where α and β are strings of terminal and nonterminal symbols. Such a rule is interpreted as a permission to rewrite a string $\gamma \alpha \delta$ to $\gamma \beta \delta$. The language of a rewriting system is then defined to be the set $\{w \in \Sigma^* \mid S \Rightarrow^* w\}$, where Σ is the terminal alphabet, S is the designated start nonterminal of the rewriting system, and \Rightarrow is the relation of one-step rewriting. It is important to note that even though the definition of the language generated by a rewriting system only refers to the rewriting relation between nonterminals and strings of terminals

$$B \Rightarrow^* w,$$

it is necessary to define the rewriting relation between strings of terminals and nonterminals in general:

$$\alpha \Rightarrow^* \beta,$$

because it is not possible to define the former directly by induction.

The standard presentation of a context-free grammar is the same, except for the restriction placed on the form of rules. All rules of a context-free grammar must be of the form

$$B \to \alpha$$
,

where B is a nonterminal and α is a string of terminals and nonterminals. The definition of the relation \Rightarrow can be given in the same way as in the unrestricted case. An important fact about context-free grammars, however, is that the relation

$$B \Rightarrow^* w$$

between nonterminals and strings of terminals can be defined by direct induction, completely bypassing the definition of the binary relation \Rightarrow^* on $(\mathcal{N} \cup \Sigma)^*$:¹

$$B \Rightarrow^* w$$
 iff for some rule $B \to v_0 B_1 v_1 \dots v_{n-1} B_n v_n$,
it holds that $B_i \Rightarrow^* w_i$ for $i = 1, \dots, n$ and
 $w = v_0 w_1 v_1 \dots v_{n-1} w_n v_n$.

In order to distinguish the above direct definition of the relation $B \Rightarrow^* w$ from the standard one, let us use a different piece of notation

to express the same relation. A rule of a context-free grammar can now be represented as a Horn clause:

$$B(v_0x_1v_1...v_{n-1}x_nv_n) := B_1(x_1), \ldots, B_n(x_n),$$

and the generated language can be defined as

$$L(G) = \{ w \in \Sigma^* \mid \vdash_G S(w) \},\$$

where $\vdash_G B(w)$ means that B(w) can be deduced from the set of rules of G viewed as a Horn clause program.

The standard rewriting view of the CFG is very close to the nondeterminisitc pushdown automaton model equivalent to it, while the above alternative "deductive" view leads to an *alternating* Turing machine (operating in logarithmic space).² Another way to express the difference is that the rewriting view is sequential, while the deductive view is parallel. The former gives rise to an exponential-time, backtracking, stack-based recognition algorithm, and the latter naturally leads to a polynomial-time, tabular

 $^{^{1}}$ I use calligraphic \mathcal{N} to denote the set of nonterminals.

²An alternating Turing machine is a generalization of a nondeterministic Turing machine that has universal and existential states. If a configuration is in an existential state, at least one of its successor configurations must lead to acceptance, while if a configuration is in a universal state, all of its successor configurations must lead to acceptance. In establishing B(w), the choice of a rule $B \rightarrow v_0 B_1 v_1 \dots v_{n-1} B_n v_n$ and the choice of substrings w_i of the given string w constitute existential (nondeterministic) moves, while checking all of $B_i(w_i)$ is realized as a universal move.

$$B \rightarrow \begin{bmatrix} D \\ B \\ D \end{bmatrix} = \begin{bmatrix} X \\ F \\ Y \end{bmatrix} = D(X), D(Y), E(Z)$$

Figure 1: The top-down rewriting view (left) and the bottom-up deductive view (right) of a context-free rule.

recognition algorithm. Or rather, the deductive view lies behind the simple bottom-up tabular recognition algorithm for CFGs. The derivation tree of $w \in L(G)$ is, with subtle points aside, just the derivation tree of $\vdash_G S(w)$. The standard notion of a derivation $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n = w$ need not play any role in the definition of context-free grammars; it was just a historical accident that the CFG was introduced as a restriction of the semi-Thue system.³

The notion of a context-free grammar can be generalized to various objects other than strings. (See Figure 1.) Under the top-down rewriting view, the right-hand side of a rule is a complex object made up of terminal and nonterminal symbols. Under the bottom-up deductive view, the argument to the nonterminal on the left-hand side is a complex object made up of terminal symbols and variables appearing on the right-hand side (ranging over complex objects made up of terminal symbols). An important notion, under both views, is the operation of substitution of complex objects for nonterminals or variables, in the free algebra of objects of an appropriate type (e.g., strings, trees, tree contexts, etc.). If M is a complex object made up of terminal symbols and variables X_1, \ldots, X_n , and P_1, \ldots, P_n are complex objects made up of terminal symbols, then we write $M[X_1 := P_1, \ldots, X_n := P_n]$ for the result of substituting P_1, \ldots, P_n for X_1, \ldots, X_n , respectively, in M. If

$$B(M) := B_1(X_1), \ldots, B_n(X_n)$$

is a rule in a context-free grammar G on an appropriate type of object, then

³The Horn-clause representation of a CFG is a special case of an *elementary formal* system, which Smullyan (1961) introduced to develop recursion theory on the basis of strings, rather than natural numbers. Elementary formal systems were later rediscovered by Groenink (1997), who called them *literal movement grammars*. In a better possible world, Smullyan's dissertation would have appeared a few years earlier and Chomsky would have based his theory of formal grammars on Smullyan's work, rather than on the work of Thue and Post.

the interpretation of this rule is the implication from

$$\vdash_G B_1(P_1),\ldots,\vdash_G B_n(P_n)$$

 to

$$\vdash_G B(M[X_1 := P_1, \ldots, X_n := P_n]),$$

for all objects P_1, \ldots, P_n of the appropriate type. The *language* of G (i.e., set of objects of the appropriate type generated by G) is defined to be $\{M \mid \vdash_G S(M)\}$, where S is the distinguished nonterminal (start symbol) of G.

1.2 Multiple Context-Free Grammars

One big advantage of the bottom-up deductive presentation is that it allows for a straightforward generalization to grammars that generate tuples of objects, rather than single objects, such as *multiple context-free grammars* (MCFGs). An MCFG is a context-free grammar on tuples of strings. A rule of an MCFG is a Horn clause of the form

$$B(t_1,\ldots,t_r) := B_1(x_{1,1},\ldots,x_{1,r_1}),\ldots,B_n(x_{n,1},\ldots,x_{n,r_n}),$$

where B, B_1, \ldots, B_n are nonterminals, $x_{1,1}, \ldots, x_{n,r_n}$ are pairwise distinct variables, and t_1, \ldots, t_r are strings made up of terminals and variables on the right-hand side. It is required that each $x_{i,j}$ appear exactly once on the left-hand side, or, in symbols:

$$|t_1 \dots t_r|_{x_{i,i}} = 1$$
 for all $i \in [1, n]$ and $j \in [1, r_i]$,

where $|w|_c$ denotes the number of occurrences of c in w.⁴ Each nonterminal is interpreted as an r-ary predicate on terminal strings, the arity r depending on the nonterminal. The arity of the start nonterminal is 1. If the maximal arity of nonterminals is $\leq m$, the grammar is called an m-MCFG. Note that 1-MCFGs are just CFGs.

Formally, an *m*-MCFG is a quintuple $G = (\mathcal{N}, \Sigma, \alpha, \mathcal{P}, S)$, where \mathcal{N} is a set of nonterminals, Σ is a set of terminals, α is a function from \mathcal{N} to $\{1, \ldots, m\}$, \mathcal{P} is a set of rules, and $S \in \mathcal{N}$ has $\alpha(S) = 1$. The arity of a nonterminal B is given by $\alpha(B)$.

⁴In Seki et al.'s (1991) original definition, the requirement was that each $x_{i,j}$ appear at most once on the left-hand side.

The following 2-MCFG generates $\texttt{RESP}=\{\,a_1^ma_2^mb_1^nb_2^na_3^ma_4^mb_3^nb_4^n\mid m,n\geq 0\,\}$:

$$S(x_1y_1x_2y_2) := P(x_1, x_2), Q(y_1, y_2).$$

$$P(\epsilon, \epsilon).$$

$$P(a_1x_1a_2, a_3x_2a_4) := P(x_1, x_2).$$

$$Q(\epsilon, \epsilon).$$

$$Q(b_1y_1b_2, b_3y_2b_4) := Q(y_1, y_2).$$

The language of an MCFG is a *multiple context-free language* (MCFL). There are many other formalisms characterizing the class of MCFLs (see, e.g., Kanazawa 200x).

1.3 Regular Tree Grammars and Multiple Regular Tree Grammars

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. The set \mathbb{T}_{Δ} of trees over a ranked alphabet Δ is the smallest set satisfying the following condition:⁵

If
$$f \in \Delta^{(n)}$$
 and $T_1, \ldots, T_n \in \mathbb{T}_{\Delta}$, then $(fT_1 \ldots T_n) \in \mathbb{T}_{\Delta}$.

Let **X** be a finite set of variables, disjoint from Δ . The notation $\mathbb{T}_{\Delta}(\mathbf{X})$ denotes the set $\mathbb{T}_{\Delta\cup\mathbf{X}}$, where $(\Delta\cup\mathbf{X})^{(0)} = \Delta^{(0)} \cup \mathbf{X}$ and $(\Delta\cup\mathbf{X})^{(n)} = \Delta^{(n)}$ for all $n \geq 1$. Let $\mathbf{X}_k = \{x_1, \ldots, x_k\}$. If $U_1, \ldots, U_k \in \mathbb{T}_\Delta$ and $T \in \mathbb{T}_\Delta(\mathbf{X}_k)$, then $T[x_1 := U_1, \ldots, x_k := U_k]$ is the result of substituting U_1, \ldots, U_k for x_1, \ldots, x_k , respectively, in T, defined recursively as follows:

$$(fT_1 \dots T_n)[x_1 := U_1, \dots, x_k := U_k] = (fT_1[x_1 := U_1, \dots, x_k := U_k] \dots T_n[x_1 := U_1, \dots, x_k := U_k]) \text{ if } f \in \Delta^{(n)}, x_i[x_1 := U_1, \dots, x_k := U_k] = U_i.$$

A tree $T \in \mathbb{T}_{\Delta}(\mathbf{X})$ is a *simple* tree if each variable in **X** appears exactly once in T, or, in symbols,

$$|T|_x = 1$$
 for all x in **X**.

A regular tree grammar (RTG) is a context-free grammar on trees over some ranked alphabet Δ . Symbols of Δ are terminal symbols. Nonterminals

⁵A common alternative notation for $(fT_1 \ldots T_n)$ is $f(T_1, \ldots, T_n)$.

of the grammar all have arity 1, as in the case of ordinary context-free grammars. Rules of an RTG have the following form:

$$B(T):-B_1(x_1),\ldots,B_n(x_n),$$

where T is a simple tree in $\mathbb{T}_{\Delta}(\mathbf{X}_n)$.

Formally, an RTG is a quadruple $G = (\mathcal{N}, \Delta, \mathcal{P}, S)$, where \mathcal{N} is a set of nonterminals, Δ is a ranked alphabet of terminals, \mathcal{P} is a set of rules, and $S \in \mathcal{N}$.

Here is an example of an RTG:

$$\begin{split} S(fxy) &:= P(x), Q(y).\\ P(e).\\ P(gaxb) &:= P(x).\\ Q(e).\\ Q(gcyd) &:= Q(y) \end{split}$$

Here, $\Delta = \Delta^{(0)} \cup \Delta^{(2)} \cup \Delta^{(3)} = \{a, b, c, d, e\} \cup \{f\} \cup \{g\}$. The language of this RTG consists of all trees of the form

$$f(\underbrace{ga(\dots(ga e \underbrace{b})\dots)b}_{n \text{ times}})(\underbrace{gc(\dots(gc e \underbrace{d})\dots)d}_{m \text{ times}}).$$

The language of an RTG is called a *regular tree language* (RTL). The regular tree languages coincide with the languages recognizable by finite tree automata.

The function yield from \mathbb{T}_{Δ} to $(\Delta^{(0)})^*$ is defined recursively as follows:

yield
$$(f) = f$$
 if $f \in \Delta^{(0)}$,
yield $(fT_1 \dots T_n) =$ yield $(T_1) \dots$ yield (T_n) if $f \in \Delta^{(n)}$ and $n \ge 1$.

If $L \subseteq \mathbb{T}_{\Delta}(\mathbf{X})$, we write yield(L) for { yield(T) | $T \in L$ }. If L is a regular tree language, yield(L) is a context-free (string) language. Conversely, every context-free language is yield(L) for some regular tree language L.

A multiple regular tree grammar (MRTG) is a context-free grammar on tuples of trees. Nonterminals of an MRTG have variying arity, and rules are of the form:

$$B(T_1,\ldots,T_r):=B_1(x_{1,1},\ldots,x_{1,r_1}),\ldots,B_n(x_{n,1},\ldots,x_{n,r_n}),$$

where T_1, \ldots, T_r are trees in $\mathbb{T}_{\Delta}(\{x_{i,j} \mid i \in [1, n], j \in [1, r_i]\})$ such that

$$\sum_{k=1}^{r} |T_k|_{x_{i,j}} = 1 \text{ for all } i \in [1, n] \text{ and } j \in [1, r_i].$$

The arity of the start nonterminal is 1.

Formally, an *m*-MRTG is a quintuple $G = (\mathcal{N}, \Delta, \alpha, \mathcal{P}, S)$, where α is a function from \mathcal{N} to $\{0, \ldots, m\}$ and $\alpha(S) = 1$.

Here is an example of a 2-MRTG:

$$S(fx_1x_2) := P(x_1, x_2).$$

 $P(e, e).$
 $P(gax_1b, gcx_2d) := P(x_1, x_2).$

Here, $\Delta = \Delta^{(0)} \cup \Delta^{(2)} \cup \Delta^{(3)} = \{a, b, c, d, e\} \cup \{f\} \cup \{g\}$. The language of this MRTG consists of all trees of the form

$$f(\underbrace{ga(\dots(ga e \underbrace{b})\dots)b}_{n \text{ times}})(\underbrace{gc(\dots(gc e \underbrace{d})\dots)d}_{n \text{ times}}).$$

The language of an MRTG is a multiple regular tree language (MRTL). If L is an MRTL, then yield(L) is an MCFL. In fact, if

$$B(T_1,\ldots,T_r) := B_1(x_{1,1},\ldots,x_{1,r_1}),\ldots,B_n(x_{n,1},\ldots,x_{n,r_n})$$

is an MRTG rule, then

$$B(\text{yield}(T_1), \dots, \text{yield}(T_r)) := B_1(x_{1,1}, \dots, x_{1,r_1}), \dots, B_n(x_{n,1}, \dots, x_{n,r_n})$$

is an MCFG rule. Conversely, every MCFL is yield(L) for some MRTL L.

1.4 Linear Context-Free Tree Grammars and Multiple Linear Context-Free Tree Grammars

An *n*-ary tree context is an expression $f(x) = \frac{1}{2} \int dx \, dx$

$$\lambda x_1 \dots x_n . T,$$

where T is a simple tree in $\mathbb{T}_{\Delta}(\mathbf{X}_n)$. The *application* of an *n*-ary tree context $U = \lambda x_1 \dots x_n T$ to trees T_1, \dots, T_n is defined as

$$app(U, T_1, \dots, T_n) = T[x_1 := T_1, \dots, x_n := T_n].$$

A *tree context* is an n-ary tree context for some n. Note that a 0-ary tree context is just a tree.

Let \mathbf{Y} be a ranked alphabet disjoint from Δ . We let $\Delta \cup \mathbf{Y}$ be the ranked alphabet such that $(\Delta \cup \mathbf{Y})^{(n)} = \Delta^{(n)} \cup \mathbf{Y}^{(n)}$. For $i \in [1, l]$, let $U_i = \lambda x_1 \dots x_{n_i} V_i$ be an n_i -ary tree context over Δ . Let $y_i \in \mathbf{Y}^{(n_i)}$ and let $T \in \mathbb{T}_{\Delta \cup \mathbf{Y}}(\mathbf{X})$. The result of substituting U_1, \dots, U_l for y_1, \dots, y_l in T, $T[y_1 := U_1, \dots, y_l := U_l]$ in symbols, is defined recursively as follows:

$$\begin{aligned} (fT_1 \dots T_n)[y_1 &:= U_1, \dots, y_l := U_l] &= \\ fT_1[y_1 &:= U_1, \dots, y_l := U_l] \dots T_n[y_1 &:= U_1, \dots, y_l := U_l] & \text{if } f \in \Delta^{(n)}, \\ x[y_1 &:= U_1, \dots, y_l := U_l] &= x & \text{if } x \in \mathbf{X}, \\ (y_iT_1 \dots T_{n_i})[y_1 &:= U_1, \dots, y_l := U_l] &= \\ & \operatorname{app}(U_i, T_1[y_1 &:= U_1, \dots, y_l := U_l], \dots, T_{n_i}[y_1 &:= U_1, \dots, y_l := U_l]) \\ (yT_1 \dots T_n)[y_1 &:= U_1, \dots, y_l := U_l] &= \\ & yT_1[y_1 &:= U_1, \dots, y_l := U_l] \dots T_n[y_1 &:= U_1, \dots, y_l := U_l] & \text{if } y \notin \{y_1, \dots, y_l\} \end{aligned}$$

A linear context-free tree grammar (LCFTG) is a context-free grammar on tree contexts.⁶ Each nonterminal is a (unary) predicate on *n*-ary tree contexts for some fixed *n*, which is the nonterminal's *rank*. The rank of the start symbol *S* is 0. We write $\rho(B)$ for the rank of nonterminal *B*. To express a rule of an LCFTG, we use a set **Y** of ranked variables ranging over tree contexts. A variable of rank *n* ranges over *n*-ary tree contexts. A rule of an LCFTG is of the following form:⁷

$$B(\lambda x_1 \dots x_n T) := B_1(y_1), \dots, B_l(y_l),$$

where $n = \rho(B)$, $y_i \in \mathbf{Y}^{(\rho(B_i))}$ for each $i \in [1, l]$, and T is a simple tree in $\mathbb{T}_{\Delta \cup \mathbf{Y}}(\mathbf{X}_n)$ such that for all $y \in \mathbf{Y}$,

$$|T|_{y} = \begin{cases} 1 & \text{if } y = y_{i} \text{ for some } i \in [1, l], \\ 0 & \text{otherwise.} \end{cases}$$

$$Bx_1 \ldots x_n \rightarrow T[y_1 := B_1, \ldots, y_l := B_l].$$

 $^{^{6}}$ This is actually what is called a linear *non-deleting* context-free tree grammar in the literature on context-free tree grammars.

⁷The usual formulation of a context-free tree grammar uses the top-down rewriting view of rules, and does not use λ :

Formally, an LCFTG of rank m is a quaituple $G = (\mathcal{N}, \Delta, \rho, \mathcal{P}, S)$ such that ρ is a function from \mathcal{N} to $\{0, \ldots, m\}$ and $\rho(S) = 0$. Note that an LCFTG of rank 0 is just an RTG.

Here is an example of an LCFTG:

$$\begin{split} S(yee) &:= P(y). \\ P(\lambda x_1 x_2.f x_1 x_2). \\ P(\lambda x_1 x_2.a(y(bx_1)(bx_2))) &:= P(y). \end{split}$$

Here, $\Delta = \Delta^{(0)} \cup \Delta^{(1)} \cup \Delta^{(2)} = \{e\} \cup \{a, b\} \cup \{f\}$. The language of this grammar consists of all trees of the form:

$$\underbrace{a(\dots(a)(f(\underbrace{b(\dots(b\ e\)\dots)}_{n\ \text{times}})(\underbrace{b(\dots(b\ e\)\dots)}_{n\ \text{times}}))(\underbrace{b(\dots(b\ e\)\dots)}_{n\ \text{times}})))}_{n\ \text{times}}))_{n\ \text{times}},$$

or, in abbreviated form:

 $a^n(f(b^n e)(b^n e)).$

The yield images of the languages of LCFTGs coincide with the languages of *non-duplicating macro grammars* and with the languages of *coupled context-free tree grammars*.⁸

A multiple linear context-free tree grammar (MLCFTG) is a context-free grammar on tuples of tree contexts. Each nonterminal is associated with a vector (n_1, \ldots, n_r) of natural numbers, called its *sort*, and takes an *r*-tuple of tree contexts as arguments; the *i*-th argument is an n_i -ary tree context. We write $\sigma(B)$ for the sort of nonterminal *B*. The start symbol *S* is associated with $\sigma(S) = (0)$, i.e., *S* takes a tree as its sole argument. A rule of an MLCFTG is of the form

$$B(U_1,\ldots,U_r):=B_1(y_{1,1},\ldots,y_{1,r_1}),\ldots,B_n(y_{n,1},\ldots,y_{n,r_n}),$$

where $\sigma(B) = (n_1, \ldots, n_r)$, U_i is a n_i -ary tree context over $\Delta \cup \mathbf{Y}$ for $i \in [1, r]$, $\sigma(B_i) = (n_{i,1}, \ldots, n_{i,r_1})$ for $i \in [1, n]$, $y_{i,j} \in \mathbf{Y}^{(n_{i,j})}$, and

$$\sum_{k=1}^{r} |U_k|_y = \begin{cases} 1 & \text{if } y = y_{i,j} \text{ for some } i \in [1,n] \text{ and } j \in [1,r_i], \\ 0 & \text{otherwise.} \end{cases}$$

Formally, an MLCFTG is a quintuple $G = (\mathcal{N}, \Delta, \sigma, \mathcal{P}, S)$, where σ is a function from \mathcal{N} to \mathbb{N}^+ and $\sigma(S) = (0)$.

 $^{^8 \}mathrm{One}\ \mathrm{can}\ \mathrm{think}\ \mathrm{of}\ \mathrm{non-duplicating}\ \mathrm{macro}\ \mathrm{grammars}\ \mathrm{as}\ \mathrm{context-free}\ \mathrm{grammars}\ \mathrm{on}\ \mathrm{string}\ \mathrm{contexts}.$

	strings	trees	unary tree contexts $C[x]$	n -ary tree contexts $C[x_1, \ldots, x_n]$
single	CFG	RTG	TAG (monadic LCFTG)	LCFTG
multiple	MCFG	MRTG	MCTAG	MLCFTG

Table 1: Context-free grammars on ...

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 context-free tree grammar (cf. Engelfriet and Maneth 2000)

Here is a simple example of an MLCFTG:

$$S(f(y_1ee)(y_2ee)) := P(y_1, y_2)$$

$$P(\lambda x_1 x_2.a(y_1(bx_1)(bx_2)), \lambda x_1 x_2.a(y_2(bx_1)(bx_2))) := P(y_1, y_2)$$

$$P(\lambda x_1 x_2.f x_1 x_2, \lambda x_1 x_2.f x_1 x_2).$$

Here, the terminal alphabet is $\Delta = \Delta^{(0)} \cup \Delta^{(1)} \cup \Delta^{(2)} = \{e\} \cup \{a, b\} \cup \{f\}$ and $\sigma(S) = (0)$ and $\sigma(P) = (2, 2)$. The language of this grammar consists of all trees of the form

$$f\left(\underbrace{a(\dots(a(f\left(\underbrace{b(\dots(b\ e\)\dots)}_{n\ \text{times}})\left(\underbrace{b(\dots(b\ e\)\dots)}_{n\ \text{times}}\right)\left(\underbrace{b(\dots(b\ e\)\dots)}_{n\ \text{times}}\right))\right)\dots)}_{n\ \text{times}}\right)$$
$$\left(\underbrace{a(\dots(a(f\left(\underbrace{b(\dots(b\ e\)\dots)}_{n\ \text{times}})\left(\underbrace{b(\dots(b\ e\)\dots)}_{n\ \text{times}}\right)\left(\underbrace{b(\dots(b\ e\)\dots)}_{n\ \text{times}}\right)\right)}_{n\ \text{times}}\right),$$

or, in abbreviated form:

$$f(a^{n}(f(b^{n}e)(b^{n}e)))(a^{n}(f(b^{n}e)(b^{n}e))).$$

Table 1 contains all formalisms that have been discussed.

2 Second-Order ACGs: Context-Free Grammars on Linear λ -Terms

A second-order abstract categorial grammar is a context-free grammar on typed linear λ -terms and generalizes all preceding formalisms.

The terminal alphabet of an abstract categorial grammar (ACG) is a higher-order signature $\Sigma = (A, C, \tau)$, where A is a set of atomic types, C is a set of constants, and τ is a type assignment function from C to $\mathscr{T}(A)$. The set $\mathscr{T}(A)$ of types built on A is defined inductively as the smallest superset of A satisfying the condition that if $\alpha, \beta \in \mathscr{T}(A)$, then $\alpha \to \beta \in \mathscr{T}(A)$.

Abstract categorial grammars generate linear λ -terms over a given higher-order signature. Let **X** be a countably infinite set of variables. The set $\Lambda(\Sigma)$ of (untyped) λ -terms over a higher-order signature $\Sigma = (A, C, \tau)$ is the smallest superset of **X** \cup C satisfying the following conditions:⁹

- 1. If $M, N \in \Lambda(\Sigma)$, then $(MN) \in \Lambda(\Sigma)$;
- 2. If $M \in \Lambda(\Sigma)$ and $x \in \mathbf{X}$, then $(\lambda x.M) \in \Lambda(\Sigma)$.

The set FV(M) of *free variables* of M is understood in the usual way. A λ -term M is *closed* if $FV(M) = \emptyset$; it is *pure* if it contains no constants.

 λ -terms come in different types, and when a λ -term has free variables in it, its type is determined relative to an assignment of types to its free variables. A *type environment* is a finite set Γ of variable declarations of the form $x : \alpha$ (where $x \in \mathbf{X}, \alpha \in \mathscr{T}(A)$) in which no variable is declared more than once. A type environment is usually written as a list $x_1:\alpha_1, \ldots, x_n:\alpha_n$. The following inference system, $\lambda \to \Sigma$, derives *typing judgments* of the form $\Gamma \vdash_{\Sigma} M: \alpha$, where Γ is a type environment, $M \in \Lambda(\Sigma)$, and $\alpha \in \mathscr{T}(A)$:

$$\begin{split} \vdash_{\Sigma} c : \tau(c) & \text{for } c \in C, \qquad x : \alpha \vdash_{\Sigma} x : \alpha \quad \text{for } x \in \mathbf{X} \text{ and } \alpha \in \mathscr{T}(A), \\ \frac{\Gamma \vdash_{\Sigma} M : \beta}{\Gamma - \{x : \alpha\} \vdash_{\Sigma} \lambda x.M : \alpha \to \beta} & \text{if } \Gamma \cup \{x : \alpha\} \text{ is a type environment,} \\ \frac{\Gamma \vdash_{\Sigma} M : \alpha \to \beta \quad \Delta \vdash_{\Sigma} N : \alpha}{\Gamma \cup \Delta \vdash_{\Sigma} MN : \beta} & \text{if } \Gamma \cup \Delta \text{ is a type environment.} \end{split}$$

We write $\Gamma \vdash M : \alpha$ when M is pure, omitting reference to Σ .¹⁰

⁹As usual, we omit the 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 notation MN^k abbreviates $MN \dots N$ with "N" repeated k times.

¹⁰Our use of the symbol \vdash corresponds to \mapsto in Hindley 1997 and \Rightarrow in Mints 2000. This inference system has the property that if $x_1:\alpha_1,\ldots,x_n:\alpha_n\vdash_{\Sigma} M:\alpha$, then $FV(M) = \{x_1,\ldots,x_n\}$. However, 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 $x, y \notin FV(M)$.

ACGs only use *linear* λ -terms. A λ -term M is *linear* if the following conditions both hold:

- 1. for any subterm $\lambda x.N$ of $M, x \in FV(N)$;
- 2. for any subterm NP of M, $FV(N) \cap FV(P) = \emptyset$.

We denote the set of linear λ -terms over Σ by $\Lambda_{\text{lin}}(\Sigma)$. It is known that for every Γ and α , there are only finitely many pure linear λ -terms M in β -normal form such that $\Gamma \vdash M : \alpha$.

See Barendregt 1984, Hindley 1997, Sørensen and Urzyczyn 2006, or Hindley and Seldin 2008 for other standard notions in λ -calculus, such as substitution (of a λ -term for a free variable in a λ -term), α -conversion, β reduction, β -normal form, and η -long form. We write $\twoheadrightarrow_{\beta}$ for β -reduction, and $=_{\beta}$ for β -equality. We denote the β -normal form of M by $|M|_{\beta}$.

In a second-order ACG (over a higher-order signature $\Sigma = (A, C, \tau)$), each nonterminal *B* is associated with its *type* $\sigma(B) \in \mathscr{T}(A)$. A rule of a second-order ACG over a higher-order signature Σ is of the form

$$B(M) := B_1(X_1), \ldots, B_n(X_n).$$

where M is a linear λ -term such that

$$X_1: \sigma(B_1), \ldots, X_n: \sigma(B_n) \vdash_{\Sigma} M: \sigma(B).$$

Formally, a second-order ACG is a quintuple $G = (\mathcal{N}, \Sigma, \sigma, \mathcal{P}, S)$, where Σ is a higher-order signature (A, C, τ) and σ is a function from \mathcal{N} to $\mathscr{T}(A)$. The language of G is defined to be $L(G) = \{ |M|_{\beta} \mid \vdash_G S(M) \}$. We say that G has complexity n if the order of $\sigma(B)$ is bounded by n.¹¹ We denote the class of second-order ACGs of complexity n by ACG_(2,n).

This presentation of second-order ACGs is different from the official definition of second-order ACGs and does not generalize to ACGs in general, but makes it easier to compare second-order ACGs with other context-free formalisms. If π is a rule

$$B(M) := B_1(X_1), \ldots, B_n(X_n),$$

¹¹The order of a type α is defined recursively as follows:

$$\operatorname{ord}(p) = 1 \quad \text{if } p \in A,$$

 $\operatorname{ord}(\alpha \to \beta) = \max(\operatorname{ord}(\alpha) + 1, \operatorname{ord}(\beta)).$

This convention is standard in the literature on higher-order unification and matching, but there are a significant number of people who start counting at 0 rather than 1. Secondorder ACGs would then be called first-order ACGs.

then in official presentation there will be an abstract constant corresponding to π whose type is $B_1 \to \cdots \to B_n \to B$ and whose object realization is $\lambda X_1 \dots X_n M$.

3 Encoding Context-Free Formalisms with Second-Order ACGs

3.1 Encoding Tree Grammars

A ranked alphabet Δ can be represented by a second-order signature $\Sigma_{\Delta}^{\text{tree}} = (\{o\}, \Delta, \tau_{\Delta})$, where for each $f \in \Delta^{(n)}$, $\tau_{\Delta}(f) = o^n \to o$. We call $\Sigma_{\Delta}^{\text{tree}}$ a *tree signature*. We identify a tree in \mathbb{T}_{Δ} with a closed β -normal λ -term in $\Lambda_{\text{lin}}(\Sigma_{\Delta}^{\text{tree}})$ of type o in the obvious way. It is then clear that regular tree grammars and linear context-free tree grammars are special cases of second-order ACGs.

A second-order ACG $G = (\mathcal{N}, \Sigma, \sigma, \mathcal{P}, S)$ is called tree-generating if Σ is a tree signature and $\sigma(S) = o$.

If \mathcal{G} is a class of ACGs, we let $TR(\mathcal{G})$ denote the class of tree languages generated by tree-generating ACGs in \mathcal{G} . We have

$$TR(ACG_{(2,1)}) = RTL$$

 $TR(ACG_{(2,2)}) = LCFTL$

In both cases, one direction is obvious given the representation of trees by linear λ -terms. The other direction is also not difficult.

3.2 Encoding Strings

One way to encode strings with linear λ -terms is to view them as unary tree contexts over a monadic ranked alphabet, where each symbol has rank 1. Thus, a string $a_1 \dots a_n$ is represented by

$$/a_1 \dots a_n / = \lambda z . a_1 (\dots (a_n z) \dots).$$

Strings over an alphabet Υ are represented by closed linear λ -terms of type $o \to o$ over the signature $\Sigma_{\Upsilon}^{\text{string}} = (\{o\}, \Upsilon, \tau)$, where $\tau(a) = o \to o$ for all $a \in \Upsilon$. We call a higher-order signature of the form $\Sigma_{\Upsilon}^{\text{string}}$ a string signature. A second-order ACG $G = (\mathcal{N}, \Sigma, \sigma, \mathcal{P}, S)$ is string-generating if Σ is a string signature and $\sigma(S) = o \to o$.

Under the representation of strings by linear λ -terms, concatenation is just the **B** combinator:

$$\mathbf{B} = \lambda x y z . x (y z).$$

It is easy to see that for all strings u, v,

$$\mathbf{B}/u/v/ \twoheadrightarrow_{\beta}/uv/.$$

The yield function is represented by the homomorphism (or *lexicon*):

$$f \mapsto \begin{cases} \lambda y_1 \dots y_n z. y_1 (\dots (y_n z) \dots) & \text{if } \tau(f) = o^n \to o \text{ for some } n \ge 1, \\ \lambda z. f z & \text{if } \tau(f) = o, \end{cases}$$
$$o \mapsto o \to o.$$

If G is a tree-generating second-order ACG, applying the yield homomorphism to G gives a string-generating second-order ACG G' generating the yield image of the tree language of G. If G is of complexity n, the complexity of G' will be n + 1.

If \mathcal{G} is a class of ACGs, we let $STR(\mathcal{G})$ denote the class of string languages generated by string-generating ACGs in \mathcal{G} . We have

$$STR(ACG_{2,2}) = CFL,$$

 $STR(ACG_{2,3}) = yCFTL,$

where yCFTL is the class of yield images of CFTLs (i.e., non-duplicating macro languages).¹² The inclusion from right to left follows from the corresponding results about tree-generating ACGs (at complexity level decremented by 1). The other direction also does not require too much work.

3.3 Encoding Tuples as Typed Linear λ -Terms

There is a standard way to represent tuples by λ -terms. An *n*-tuple (M_1, \ldots, M_n) of λ -terms of type $\alpha_1, \ldots, \alpha_n$ can be represented by a λ -term

$$\langle M_1,\ldots,M_n\rangle = \lambda w.wM_1\ldots M_n,$$

which has type $(\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow \beta) \rightarrow \beta$ for any β . Note that if M_1, \ldots, M_n are linear λ -terms, then $\langle M_1, \ldots, M_n \rangle$ is also linear. The projection functions

$$p_i \colon \langle M_1, \ldots, M_n \rangle \mapsto M_i$$

 $^{^{12}\}text{According to the definition of string generating second-order ACGs, we have <math display="inline">STR(\text{ACG}_{2,1}) = \varnothing.$

are usually defined by

$$p_i = \lambda y. y(\lambda x_1 \dots x_n. x_i),$$

but these are not linear λ -terms. In multiple context-free grammars, as well as in multiple linear context-free tree grammars, each component of a tuple is used exactly once in some context $C[\Box, \ldots, \Box]$, and we have

$$C[M_1,\ldots,M_n] =_\beta \langle M_1,\ldots,M_n \rangle (\lambda x_1 \ldots x_n \cdot C[x_1,\ldots,x_n]),$$

so the effect of projections and plugging into a context $C[\Box, ..., \Box]$ is achieved by feeding the linear λ -term $\lambda x_1 \dots x_n \cdot C[x_1, \dots, x_n]$ to the tuple.

There is the problem of typing, however. In order to type the linear λ -term $\langle M_1, \ldots, M_n \rangle (\lambda x_1 \ldots x_n . C[x_1, \ldots, x_n])$, we have to assign to $\langle M_1, \ldots, M_n \rangle$ the type $(\alpha_1 \to \cdots \to \alpha_n \to \beta) \to \beta$, where β is the type of $C[x_1, \ldots, x_n]$. If one wants to use components of tuples in contexts $C[\Box, \ldots, \Box]$ of varying types, no single typing of $\langle M_1, \ldots, M_n \rangle$ makes it possible for $\langle M_1, \ldots, M_n \rangle$ to apply to $\lambda x_1 \ldots x_n . C[x_1, \ldots, x_n]$. However, we have to represent the tuple of arguments of a nonterminal by a λ -term of some fixed type.

Fortunately, since string- and tree-generating ACGs use a higher-order signature with just one atomic type o, it suffices to use $(\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow o) \rightarrow o$ as the type of $\langle M_1, \ldots, M_n \rangle$, where $\alpha_1, \ldots, \alpha_n$ are the types of M_1, \ldots, M_n . Since the type of any context ends in o, we have

$$C[M_1,\ldots,M_n] =_{\beta\eta} \lambda y_1 \ldots y_k \langle M_1,\ldots,M_n \rangle (\lambda x_1 \ldots x_n \cdot C[x_1,\ldots,x_n]y_1 \ldots y_k)$$

and this is typable by assigning to y_1, \ldots, y_k types β_1, \ldots, β_k , respectively, if $C[x_1, \ldots, x_n]$ has type $\beta_1 \to \ldots \to \beta_k \to o$.

Using this representation of tuples, our example MRTG can be encoded by the following second-order ACG:

$$S(X(\lambda x_1 x_2.f x_1 x_2)) := P(X).$$

$$P(\lambda w.wee).$$

$$P(\lambda w.X(\lambda x_1 x_2.w(gax_1b)(gcx_2d))) := P(X).$$

This ACG has $\sigma(S) = o$ and $\sigma(P) = (o^2 \rightarrow o) \rightarrow o$, corresponding to $\alpha(S) = 1$ and $\alpha(P) = 2$ in the MRTG. In general, MRTGs can be encoded by an ACG in AC_(2,3).

$$TR(ACG_{(2,3)}) \supseteq MRTL.$$

Applying the yield function to both sides, we can conclude

$$STR(ACG_{(2,4)}) \supseteq MCFL$$

Using the same technique, we can represent our example MLCFTG by the following second-order ACG:

$$S(X(\lambda y_1 y_2.f(y_1 e e)(y_2 e e))) := P(X).$$

$$P(\lambda w.w(\lambda x_1 x_2.f x_1 x_2)(\lambda x_1 x_2.f x_1 x_2)).$$

$$P(\lambda w.X(\lambda y_1 y_2.w(\lambda x_1 x_2.a(y_1(b x_1)(b x_2)))(\lambda x_1 x_2.a(y_2(b x_1)(b x_2))))) := P(X)$$

In this second-order ACG, $\sigma(S) = o$ and $\sigma(P) = ((o^2 \rightarrow o) \rightarrow (o^2 \rightarrow o) \rightarrow o) \rightarrow o$, corresponding to $\sigma(S) = (0)$ and $\sigma(P) = (2, 2)$ in the MLCFTG. In general, a second-order ACG encoding an MLCFTG has complexity 4.

$$TR(ACG_{(2,4)}) \supseteq MLCFTL$$

Since strings are unary tree contexts, we can also encode MCFGs with ACGs in $ACG_{(2,4)}$ as a special case. This gives an alternative proof of

$$STR(ACG_{(2,4)}) \supseteq MCFL.$$

The inclusion of CFL, yCFTL, MCFL in $ACG_{(2,2)}$, $ACG_{(2,3)}$, $ACG_{(2,4)}$, respectively, was first proved by de Groote and Pogodalla (2004).

4 Characterization of String and Tree Generating Power

A typed linear λ -term over a string or tree signature in general stands for a functional on strings and trees of higher type. A second-order term denotes a function, a third-order term denotes a function from functions to strings/trees, etc. As one moves up the hierarchy $ACG_{(2,n)}$ of second-order ACGs, more and more complex objects become available to grammars. Does this increase in the complexity of objects handled by grammars lead to an ever-growing hierarchy of string and tree language classes?

In 2006, Sylvain Salvati surprised the ACG community by showing that the string language hierarchy collapses at complexity level 4 (Salvati 2007). He was able to prove

$$STR(ACG_{(2,n)}) \subseteq MCFL$$

for all n. Since this fact implies

$$STR(ACG_{(2,n)}) = MCFL$$

for all $n \ge 4$, second-order ACGs joined the long list of grammar formalisms characterizing the class of multiple context-free languages, widely equated with the class of *mildly context-sensitive languages*. Since then, two alternative proofs of Salvati's theorem have been found, and a similar result has been established for the tree language hierarchy (Kanazawa 200x, Kanazawa and Salvati 2007):

$$TR(ACG_{(2,n)}) = MLCFTL$$
 for all $n \ge 4$.

The following method for the string case is from Kanazawa and Salvati 2007.

Let $M \in \Lambda_{\text{lin}}(\Sigma_{\Upsilon}^{\text{string}})$ be a linear λ -term in η -long β -normal form such that $\Gamma \vdash_{\Sigma_{\Upsilon}^{\text{string}}} M : \alpha$. We extract from M a tuple (w_1, \ldots, w_m) of strings in Υ^+ and a pure λ -term P such that

$$\Gamma, z_1 : o \to o, \dots, z_m : o \to o \vdash P : \alpha,$$
$$P[z_i := /w_i/]_{i \in [1,m]} \twoheadrightarrow_{\beta} M.$$

We ensure m to be the minimal number for which there exist (w_1, \ldots, w_n) and P with these properties. Roughly, (w_1, \ldots, w_m) consists of the maximal consecutive strings of symbols from Υ that occur in M. In the following definition, $\ln(\vec{w})$ denotes the length (i.e., number of components) of a sequence \vec{w} , and $\widehat{}$ denotes concatenation of sequences. The letters a and y range over symbols in Υ and variables, respectively.

$$\begin{split} \operatorname{tuple}(aM) &= \begin{cases} (aw_1, w_2, \dots, w_m) & \text{if } M \text{ starts with a constant and} \\ & \operatorname{tuple}(M) = (w_1, \dots, w_m), \\ (a)^{\text{tuple}}(M) & \text{otherwise,} \end{cases} \\ pure(aM) &= \begin{cases} pure(M) & \text{if } M \text{ starts with a constant,} \\ z_1(\operatorname{pure}(M)[z_i := z_{i+1}]_{1 \leq i \leq m}) & \text{otherwise, where } m = \operatorname{lh}(\operatorname{tuple}(M)), \\ \operatorname{tuple}(yM_1 \dots M_n) &= \operatorname{tuple}(M_1)^{\text{charged}} \dots^{\text{charged}} \operatorname{tuple}(M_n), \\ pure(yM_1 \dots M_n) &= yQ_1 \dots Q_n, & \text{where } Q_j = \operatorname{pure}(M_j)[z_i := z_{i+\sum_{k=1}^{j-1} m_k}]_{1 \leq i \leq m_j}, \\ & m_k = \operatorname{lh}(\operatorname{tuple}(M_k)), \\ \\ \operatorname{tuple}(\lambda y.M) &= \operatorname{tuple}(M), \\ & \operatorname{pure}(\lambda y.M) = \operatorname{tuple}(M), \\ & \operatorname{pure}(\lambda y.M) = \lambda y.\operatorname{pure}(M) \end{split}$$

For example, consider

$$M = \lambda xyz.a(a(b(y(\lambda z.b(c(x(cz))))(d(dz))))),$$

where $a, b, c, d \in \Upsilon$. Note

$$\vdash_{\Sigma^{\mathrm{string}}_{\Upsilon}} M: (o \to o) \to ((o \to o) \to o \to o) \to o \to o.$$

We have

$$tuple(M) = (aab, bc, c, dd),$$

$$pure(M) = \lambda xyz.z_1(y(\lambda z.z_2(x(z_3z)))(z_4z)).$$

Lemma 1. Let M be an η -long β -normal λ -term in $\Lambda_{\text{lin}}(\Sigma_{\Upsilon}^{\text{string}})$ such that $y_1: \alpha_1, \ldots, y_k: \alpha_k \vdash_{\Sigma_{\Upsilon}^{\text{string}}} M: \beta$, and let $\text{tuple}(M) = (w_1, \ldots, w_m)$. Then the following hold:

(i) $y_1: \alpha_1, \ldots, y_k: \alpha_k, z_1: o \to o, \ldots, z_m: o \to o \vdash \text{pure}(M): \beta$.

(ii) pure
$$(M)[z_i := /w_i/]_{1 \le i \le m} \twoheadrightarrow_{\beta} M.$$

(iii)
$$m \le \frac{1}{2} \left(|\beta| + \sum_{i=1}^{k} |\alpha_i| \right).$$

Proof. Easy induction on M. For (iii), show that $2m \leq |\beta| + \sum_{i=1}^{k} |\alpha_i| - 2$ if the head of M is a variable.¹³

Let $G = (\mathcal{N}, \Sigma_{\Upsilon}^{\text{string}}, \sigma, \mathcal{P}, S)$ be a second-order string-generating ACG. We assume that all λ -terms that appear on the left-hand side of rules in \mathcal{P} are in η -long β -normal form. For $\alpha \in \mathscr{T}(\{o\})$, let \mathbf{P}_{α} be the set of pure linear λ -terms M in η -long β -normal form such that $z_1: o \to o, \ldots, z_r: o \to o \vdash M: \alpha$ for some $r \leq \frac{1}{2} |\alpha|$. Note that \mathbf{P}_{α} is finite for all α . Define an MCFG $G' = (\mathcal{N}', \Upsilon, \alpha, \mathcal{P}', S')$ as follows:

¹³Readers of Kanazawa 2006 may realize that this rather technical lemma is in fact an application of interpolation. Let $\widehat{M}[x_1, \ldots, x_n]$ be a pure linear λ -term such that $\widehat{M}[a_1, \ldots, a_n] = M$, where $a_1, \ldots, a_n \in \Upsilon$. Then tuple(M) and pure(M) may be computed simply by applying the method of computing interpolants in Kanazawa 2006 to $y_1: \alpha_1, \ldots, y_k: \alpha_k, x_1: o \to o, \ldots, x_n: o \to o \vdash \widehat{M}[x_1, \ldots, x_n]: \alpha$ with respect to the partition $(x_1: o \to o, \ldots, x_n: o \to o; y_1: \alpha_1, \ldots, y_k: \alpha_k)$ and replacing the variables x_i by a_i in the interpolant thus found.

- $\mathcal{N}' = \{S'\} \cup \{(B, P) \mid B \in \mathcal{N}, P \in \mathbf{P}_{\sigma(B)}\}, \text{ where } \alpha((B, P)) = |\mathrm{FV}(P)|.$
- For each rule $B(M) := B_1(X_1), \ldots, B(X_n)$ and $P_i \in \mathbf{P}_{\sigma(B_i)}$ with $|FV(P_i)| = r_i \ (1 \le i \le n), \mathcal{P}'$ contains the rule

$$(B, \operatorname{pure}(Q))(\operatorname{tuple}(Q)) := (B_1, P_1)(x_{1,1}, \dots, x_{1,r_1}), \dots, (B_n, P_n)(x_{n,1}, \dots, x_{n,r_n}),$$

where

$$Q = |M[X_1 := P_1[z_j := /x_{1,j}/]_{1 \le j \le r_1}, \dots, X_n := P_n[z_j := /x_{n,j}/]_{1 \le j \le r_n}]|_{\beta}$$

and tuple(Q), pure(Q) are defined with respect to the alphabet $\Upsilon \cup \{x_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq r_i\}.$

• In addition, \mathcal{P} contains the following rules:¹⁴

$$S'(\epsilon) := (S, \lambda z. z). \qquad S'(x) := (S, \lambda z. z_1 z)(x).$$

Lemma 2. Let Υ and Ψ be alphabets, and $h: \Upsilon \to \Psi^+$ be a non-erasing homomorphism. For every η -long β -normal λ -term M in $\Lambda_{\text{lin}}(\Sigma_{\Upsilon}^{\text{string}})$, we have

$$pure(|M[a:=/h(a)/]_{a\in\Upsilon}|_{\beta}) = pure(M),$$

$$tuple(|M[a:=/h(a)/]_{a\in\Upsilon}|_{\beta}) = tuple(M)[a:=h(a)]_{a\in\Upsilon}.$$

Proof. Easy induction on M.

Lemma 3. Let $G = (\mathcal{N}, \Sigma_{\Upsilon}^{\text{string}}, \sigma, \mathcal{P}, S)$ be a string-generating second-order ACG generating $L \subseteq \Upsilon^*$ and $G' = (\mathcal{N}', \Upsilon, \alpha, \mathcal{P}', S')$ be the MCFG obtained by the above construction. Then L(G') = L.

Proof. To show that $L \subseteq L(G')$, we prove by induction on derivations that if $\vdash_G B(M)$, then $\vdash_{G'} (B, \text{pure}(|M|_{\beta}))(\text{tuple}(|M|_{\beta}))$. Suppose that the last step of the derivation of $\vdash_G B(M)$ is by the rule

$$B(M'):=B_1(X_1),\ldots,B_n(X_n).$$

¹⁴This construction produces an MCFG where some nonterminals have arity 0. Such nonterminals can be removed from the grammar by a standard technique to obtain an MCFG that is in accordance with our definition.

Then there must be $M_1, ..., M_n$ such that $M = M'[X_1 := M_1, ..., X_n := M_n]$ and $\vdash_G B_i(M_i)$ (i = 1, ..., n). Let $P_i = \text{pure}(|M_i|_{\beta}), (w_{i,1}, ..., w_{i,r_i}) =$ tuple $(|M_i|_{\beta})$, and $Q = |M'[X_1 := P_1[z_j := /x_{1,j}/]_{j \in [1,r_1]}, ..., X_n := P_n[z_j := /x_{n,j}/]_{j \in [1,r_n]}]|_{\beta}$. Then G' contains the rule

$$(B, \operatorname{pure}(Q))(\operatorname{tuple}(Q)) :- (B_1, P_1)(x_{1,1}, \dots, x_{1,r_1}), \dots, (B_n, P_n)(x_{n,1}, \dots, x_{n,r_n}).$$

By induction hypothesis, $\vdash_{G'} (B_i, P_i)(w_{i,1}, \ldots, w_{i,r_i})$ for $i = 1, \ldots, n$, and this implies

$$\vdash_{G'} (B, \operatorname{pure}(Q))(\operatorname{tuple}(Q)[x_{i,j} := w_{i,j}]_{i \in [1,n], j \in [1,r_i]}).$$

Now

$$Q[x_{i,j} := /w_{i,j}/]_{i \in [1,n], j \in [1,r_i]}$$

= $_{\beta} M'[X_1 := P_1[z_j := /w_{1,j}/]_{j \in [1,r_1]}, \dots, X_n := P_n[z_j := /w_{n,j}/]_{j \in [1,r_n]}]$
 $\Rightarrow_{\beta} M'[X_1 := |M_1|_{\beta}, \dots, X_n := |M_n|_{\beta}]$ by Lemma 1,
= $_{\beta} M$.

By Lemma 2, then, $\operatorname{pure}(|M|_{\beta}) = \operatorname{pure}(Q)$ and $\operatorname{tuple}(|M|_{\beta}) = \operatorname{tuple}(Q)[x_{i,j} := w_{i,j}]_{i \in [1,n], j \in [1,r_i]}$. Hence we have $\vdash_{G'}(B, \operatorname{pure}(|M|_{\beta}))(\operatorname{tuple}(|M|_{\beta}))$, as desired.

To show that $L(G') \subseteq L$, we prove by induction on derivations that if $\vdash_{G'} (B, P)(w_1, \ldots, w_m)$, then there is an $M \in \Lambda_{\text{lin}}(\Sigma_{\Upsilon}^{\text{string}})$ such that $\vdash_G B(M)$ and $M =_{\beta} P[z_i := /w_i/]_{i \in [1,m]}$. Suppose that the last step of the derivation of $\vdash_{G'} (B, P)(w_1, \ldots, w_m)$ is by the rule

$$(B, P)(v_1, \dots, v_m) := (B_1, P_1)(x_{1,1}, \dots, x_{1,r_1}), \dots, (B_n, P_n)(x_{n,1}, \dots, x_{n,r_n})$$

and this rule came from the rule

$$B(M') := B_1(X_1), \ldots, B_n(X_n)$$

of G. Then we must have $\vdash_{G'} (B_i, P_i)(w_{i,1}, \dots, w_{i,r_i})$ $(i = 1, \dots, n)$ and $(v_1, \dots, v_m)[x_{i,j} := w_{i,j}]_{i \in [1,n], j \in [1,r_i]} = (w_1, \dots, w_m)$ for some $w_{1,1}, \dots, w_{1,r_1}, \dots, w_{n,1}, \dots, w_n n, r_n$. Let

$$Q = |M'[X_1 := P_1[z_j := /x_{1,j}/]_{j \in [1,r_1]}, \dots, X_n := P_n[z_j := /x_{n,j}/]_{j \in [1,r_n]}]|_{\beta}.$$

Then P = pure(Q) and $(v_1, \ldots, v_m) = \text{tuple}(Q)$. By induction hypothesis, there are $M_1, \ldots, M_n \in \Lambda_{\text{lin}}(\Sigma_{\Upsilon}^{\text{string}})$ such that $\vdash_G B_i(M_i)$ and $M_i =_{\beta} P_i[z_j := /w_{i,j}/]_{j \in [1,r_i]}$. Then $\vdash_G B(M'[X_1 := M_1, \ldots, X_n := M_n])$ and

$$M'[X_1 := M_1, \ldots, X_n := M_n]$$

$$\begin{split} &=_{\beta} M'[X_1 := P_1[z_j := /w_{1,j}/]_{j \in [1,r_1]}, \dots, X_n := P_n[z_j := /w_{n,j}/]_{j \in [1,r_n]}] \\ &=_{\beta} Q[x_{i,j} := /w_{i,j}/]_{i \in [1,n], j \in [1,r_i]} \\ &=_{\beta} (P[z_i := /v_i/]_{1 \le i \le m})[x_{i,j} := /w_{i,j}/]_{i \in [1,n], j \in [1,r_i]} \quad \text{by Lemma 1,} \\ &= P[z_i := /w_i/]_{i \in [1,m]}, \end{split}$$

which completes the proof.

References

- Barendregt, Hendrik Pieter. 1984. *The Lambda Calculus*. Amsterdam: North-Holland. Revised Edition.
- 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(4):421–438.
- Engelfriet, Joost. 1997. Context-free graph grammars. In G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 125–213. Berlin: Springer.
- Engelfriet, Joost and Sebastian Maneth. 2000. Tree languages generated by contextfree graph grammars. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, eds., *Theory and Application of Graph Transformations*, vol. 1764 of *Lecture Notes in Computer Science*, pages 15–29. Berlin: Springer.
- Engelfriet, J. and E. M. Schmidt. 1977. IO and OI, part I. Journal of Computer and System Sciences 15:328–353.
- Groenink, Annius V. 1997. Mild context-sensitivity and tuple-based generalizations of context-free grammar. *Linguistics and Philosophy* 20(6):607–636.
- Hindley, J. Roger. 1997. Basic Simple Type Theory. Cambridge: Cambridge University Press.
- Hindley, J. Roger and Jonathan P. Seldin. 2008. Lambda-Calculus and Combinators: An Introduction. Cambridge: Cambridge University Press.
- Kanazawa, Makoto. 2006. Computing interpolants in implicational logics. Annals of Pure and Applied Logic 142(1–3):125–201.
- Kanazawa, Makoto. 200x. Second-order abstract categorial grammars as hyperedge replacement grammars. To appear in *Journal of Logic, Language and Information*.
- Kanazawa, Makoto and Sylvain Salvati. 2007. Generating control languages with abstract categorial grammars. In *Preliminary proceedings of FG 2007: The 12th Conference on Formal Grammar*.

- Kepser, 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.
- 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. 2007. Encoding second order string ACG with deterministic tree walking transducers. In S. Wintner, ed., *Proceedings of FG 2006: The 11th* conference on Formal Grammar, FG Online Proceedings, pages 143–156. CSLI Publications.
- Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science* 88(2):191–229.
- Smullyan, Raymond M. 1961. Theory of Formal Systems. Princeton, N.J.: Princeton University Press.
- 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. thesis, University of Pennsylvania.