Linear Context-Free Rewriting Systems and Deterministic Tree-Walking Transducers*

David J. Weir

School of Cognitive and Computing Sciences
University of Sussex
Falmer, Brighton BN1 9QH
davidw@cogs.sussex.ac.uk

Abstract

We show that the class of string languages generated by linear context-free rewriting systems is equal to the class of output languages of deterministic tree-walking transducers. From equivalences that have previously been established we know that this class of languages is also equal to the string languages generated by context-free hypergraph grammars, multicomponent tree-adjoining grammars, and multiple context-free grammars and to the class of yields of images of the regular tree languages under finite-copying top-down tree transducers.

Introduction

In [9] a comparison was made of the generative capacity of a number of grammar formalisms. Several were found to share a number of characteristics (described below) and the class of such formalisms was called linear context-free rewriting systems. This paper shows how the class of string languages generated by linear context-free rewriting systems relates to a number of other systems that have been studied by formal language theorists. In particular, we show that the class of string languages generated by linear context-free rewriting systems is equal to the class of output languages of deterministic tree-walking transducers [1].

A number of other equivalences have already been established. In [10] it was shown that linear context-free rewriting systems and multicomponent tree adjoining grammars [6] generate the same string languages. The multiple context-free grammars of [7] are equivalent to linear context-free systems. This follows from the fact that multiple context-free grammars are exactly that subclass of the linear context-free rewriting systems in which the objects generated by the grammar are tuples of strings. The class of output languages of deterministic tree-walking transducers is known to be equal to the class of yields of images of the regular tree languages under finite-copying top-down tree transducers [4] and in [3] it was shown that it also equal to the string languages generated by context-free hypergraph grammars [2, 5].

We therefore have a number of characterizations of the same class of languages and results that have been established for the class of languages associated with one system carry over to the others. This is particularly fruitful in this case since the output languages of deterministic tree-walking transducers have been well studied (see [4]).

In the remainder of the paper we describe linear context-free rewriting systems and deterministic tree-walking transducers and outline the equivalence proof. We then describe context-free hypergraph

*I would like to thank Joost Engelfriet for drawing my attention to context-free hypergraph grammars and their relationship to deterministic tree-walking automata.

grammars and observe that they are a context-free rewriting system.

Linear Context-Free Rewriting Systems

Linear context-free rewriting systems arose from the observation that a number of grammatical formalisms share two properties.

1. Their derivation tree sets can be generated by a context-free grammar.
2. Their composition operations are size-preserving, i.e., when two or more substructures are combined only a bounded amount of structure is added or deleted.

Examples of formalisms that satisfy these conditions are head grammars [8], tree adjoining grammars [6], multicomponent tree adjoining grammars [6] and context-free hypergraph grammars. It was shown [9] that a system satisfying the above conditions generates languages that are semilinear and can be recognized in polynomial time. The definition of linear context-free rewriting systems is deliberately not specific about the kinds of structures being manipulated. In the case of head grammars these are pairs of strings whereas tree adjoining grammars manipulate trees and context-free hypergraph grammars manipulate graphs.

In [9] size-preserving operations are defined for arbitrary structures in terms of properties of the corresponding functions over the *terminal yield* of the structures involved. The yield is taken to be a tuple of terminal strings. We call the function associated with a composition operation the **yield function** of that operation. The yield function of $\cap f$ of a composition operation f gives the yield of the structure $f(c1, ldots, c_n)$ based on the yield of the structures $c_1, \ldots, c_n$.

Let Σ be an alphabet of terminal symbols. f is an n-ary **linear regular operation** over tuples of strings in Σ if it can be defined with an equation of the form

$$f(\langle x_{1,1}, \ldots, x_{1,k_1}\rangle, \ldots, \langle x_{n,1}, \ldots, x_{n,k_n}\rangle) = \langle t_1, \ldots, t_k\rangle$$

where each $k_i > 0$, $n \geq 0$ and each t_i is a string of variables (x's) and symbols in Σ and where the equation is regular (all the variables appearing on one side appear on the other) and linear (the variables appear only once on the left and right).

For example, the operations of head grammars can be define with the equations[1]:

$$wrap(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) = \langle x_1 y_1, y_2 x_2\rangle$$

$$conc1(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) = \langle x_1, x_2 y_1 y_2\rangle$$

$$conc2(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) = \langle x_1 x_2 y_1, y_2\rangle$$

Thus, we have

$$wrap(\langle ab, ca\rangle, \langle ac, bc\rangle) = \langle abac, bcca\rangle$$

$$conc1(\langle ab, ca\rangle, \langle ac, bc\rangle) = \langle ab, caacbc\rangle$$

$$conc2(\langle ab, ca\rangle, \langle ac, bc\rangle) = \langle abcaac, bc\rangle$$

A **generalized context-free grammar** (gcfg) [8] is denoted $G = (V_N, S, F, P)$ where
V_N is a finite set of nonterminal symbols,
S is a distinguished member of V_N,
F is a finite set of function symbols and
P is a finite set of productions of the form

$$A \rightarrow f(A_1, \ldots, A_n)$$

where $n \geq 0$, $f \in F$, and $A, A_1, \ldots, A_n \in V_N$.

With a grammatical formalism we associate an **interpretation function** m that maps symbols in F onto the formalism's composition operations. For example, in a typical head grammar the set F might include $\{W, C1, C2\}$ where $m(W) = wrap$, $m(C1) = conc1$ and $m(C2) = conc2$.

A formalism is a **linear context-free rewriting system** (lcfrs) if every grammar can be expressed as a gcfg and its interpretation function m maps symbols onto operations whose yield functions are linear regular operations.

[1] These operations differ from (but are equivalent to) those used in [8]

In order to simplify the remaining discussion we assume that m maps directly onto the yield functions themselves.

The language $L(G)$ generated by a gcfg $G = (V_N, S, F, P)$ with associated interpretation function m is defined as

$$L(G) = \left\{ w_1 \dots w_k \mid S \underset{G}{\Longrightarrow} \langle w_1, \dots, w_k \rangle \right\}$$

where

- $A \underset{G}{\Longrightarrow} m(f)$
  if $A \rightarrow f() \in P$
- $A \underset{G}{\Longrightarrow} m(f)(t_1, \dots, t_n)$
  if $A \rightarrow f(A_1, \dots, A_n) \in P$ and
  $A_i \underset{G}{\Longrightarrow} t_i$ $(1 \leq i \leq n)$.

We denote the class of all languages generated by lcfrs as LCFRL.

Deterministic Tree-Walking Transducers

A deterministic tree-walking transducer is an automaton whose inputs are derivation trees of some context-free grammar. The automaton moves around the tree starting at the root. At each point in the computation, depending on the label of the current node and the state of the finite state control, the automaton moves up, down or stays at the current node and outputs a string. The computation ends when the machine tries to move to the parent of the root node.

We denote a **deterministic tree-walking transducer** (dtwt) by $M = (Q, G, \Delta, \delta, q_0, F)$ where
Q is a finite set of states,
$G = (V_N, V_T, S, P)$ is a context-free grammar without ϵ-rules,
Δ is a finite set of output symbols,
$\delta : Q \times (V_N \cup V_T) \rightarrow Q \times D \times \Delta^*$ is the transition function where
$D = \{ stay, \mathrm{up} \} \cup \{ \mathrm{d}(k) \mid k \geq 1 \}$,
$q_0 \in Q$ is the initial state and
$F \subseteq Q$ is the set of final states.

A configuration of M is a 4-tuple (q, γ, η, w) where $q \in Q$ is the current state, γ is the derivation tree of G under consideration, η is a node in γ or $\uparrow$ (where $\uparrow$ can be thought of as the parent of the root of γ), and $w \in \Delta^*$ is the output string produced up to that point in the computation. We have

$$(q, \gamma, \eta, w) \vdash_M (q', \gamma, \eta', ww')$$

if the label of η is X, $\delta(q, X) = (q', d, w')$ such that when $d = stay$ then $\eta' = \eta$, when $d = \mathrm{d}(i)$ then η' is the ith child of η (if it exists), and when $d = \mathrm{up}$ then η' is the parent of η ($\uparrow$ if η is the root of γ).

The output language $\mathrm{OUT}(M)$ of M is the set of strings:

$$\{ w \in \Delta^* \mid \begin{array}{l} (q_0, \gamma, \eta_r, \epsilon) \vdash_M^* (q_f, \gamma, \uparrow, w), \\ q_f \in F \text{ and} \\ \gamma \text{ is a derivation tree of } G \text{ with root } \eta_r \end{array} \}$$

where $\vdash_M^*$ is the reflexive transitive closure of $\vdash_M$.

We denote the class of all languages $\mathrm{OUT}(M)$ where M is a dtwt as OUT(DTWT).

Consider the dtwt

$$M = (\{ q_0, q_1, q_2, q_3 \}, G, \{ a, b, c, d \}, \delta, q_0, \{ q_3 \})$$

where $G = (\{ S \}, \{ e \}, S, \{ S \rightarrow A, A \rightarrow A, A \rightarrow e \})$ and the relevant component of δ is defined as follows.

$$\begin{array}{ll} \delta(q_0, S) = (q_0, \mathrm{d}(1), \epsilon) & \delta(q_0, e) = (q_1, \mathrm{up}, \epsilon) \\ \delta(q_0, A) = (q_0, \mathrm{d}(1), a) & \delta(q_1, A) = (q_1, \mathrm{up}, b) \\ \delta(q_1, S) = (q_2, \mathrm{d}(1), \epsilon) & \delta(q_2, e) = (q_3, \mathrm{up}, \epsilon) \\ \delta(q_2, A) = (q_2, \mathrm{d}(1), c) & \delta(q_3, A) = (q_3, \mathrm{up}, d) \\ \delta(q_3, S) = (q_3, \mathrm{up}, \epsilon) & \end{array}$$

It can be seen that $\mathrm{OUT}(M) = \{ a^n b^n c^n d^n \mid n \geq 1 \}$.

Equivalence

In this section we outline a two part proof that OUT(DTWT) = LCFRL.

OUT(DTWT) $\subseteq$ LCFRL

Consider a dtwt $M = (Q, \Sigma, G, \Delta, \delta, q_0, F)$ where $G = (V_N, V_T, S, P)$. For convenience we assume that M is a dtwt without *stay* moves (see Lemma 5.1 in [3] for proof that this can be done).

Given a derivation tree of G, and a node η in this tree, we record the strings contributed to the output between the first and last visit to nodes in the subtree rooted at η. These contributed terminal strings can be viewed as a k tuple where k is the number of times that the transducer enters and then leaves the subtree.

For each production $X \rightarrow X_1 \ldots X_n$ in P and each $p \in Q$ we call

$$\mathcal{C}(\underline{(X, p, \cdot)} \rightarrow (X_1, \epsilon, 0) \ldots (X_n, \epsilon, 0))$$

$\mathcal{C}(\underline{(A, p, \cdot)} \rightarrow (X_1, \epsilon, 0) \ldots (X_n, \epsilon, 0))$ simulates all subcomputations of M that start in state p at a node labelled X that has been expanded using the production $X \rightarrow X_1 \ldots X_n$. The node labelled A may be visited several times, but each time the machine must be in a different state (otherwise, being deterministic, it would loop indefinitely). The sequence of visits is recorded as a string of states. The component of the rule that is underlined indicates which of the children or parent is currently being visited. The call $\mathcal{C}(\underline{(X, \alpha, \phi)} \rightarrow (X_1, \alpha_1, i_1) \ldots (X_n, \alpha_n, i_n))$ is made when a computation is being simulated in which the node labelled A has been visited $|\alpha|$ times ($|\alpha|$ denotes the length of α) such that on the ith visit the machine was in the state indicated by the ith symbol in α. $\alpha_1, \ldots, \alpha_n$ are used in a similar way to encode the state of the machine during visits to each child node. ϕ is a string of terms that is used to encode the output produced between the first and last visit to the subtree rooted at the node labelled A. Ultimately, it has the form $\cdot t_1 \cdot \ldots \cdot t_k \cdot$ where each t_i encodes the composition of the ith component of the tuple. The notation used for each t_i is identical to that used in the equations used to define lcfrs composition operations given earlier, i.e., each t_i is a string of output symbols and x's. $i_1, \ldots, i_n$ are used to encode the number of times that a given child has been visited *from above*. This gives the number of times the subtree rooted at that node has been visited and, hence, encodes which component of the tuple was completed most recently. Thus, for each j, $1 \leq j \leq n$, the simulation has moved from the parent to the jth child i_j times. This number is used to determine which component of the tuple derived from the jth node should contribute to the parent's current component. When a move is made from the parent node to the jth child we add the variable x_{j,i_j+1} to the term currently being constructed for the parent node. In other words, the next component of the parent output is the $i_j + 1$th component of its jth child.

The call

$$\mathcal{C}((X, \alpha, \phi) \rightarrow (X_1, \alpha_1, i_1) \ldots \underline{(X_j, \alpha_j, i_j)} \ldots (X_n, \alpha_n, i_n))$$

sumulates the machine visiting the jth child of a node expanded using the rule $X \rightarrow X_1 \ldots X_n$.

From M the gcfg G' is constructed such that $G' = (V'_N, S', F, P')$ where

$$V'_N = \{ S' \} \cup \{(X, \alpha) \mid X \in V_N \cup V_T \text{ and non-repeating } \alpha \in Q^*\}$$

and the procedure $\mathcal{C}$ determines P' and F where for each production $A \rightarrow X_1 \ldots X_n$ in P and each $p \in Q$ we call

$$\mathcal{C}(\underline{(A, p, \cdot)} \rightarrow (X_1, \epsilon, 0) \ldots (X_n, \epsilon, 0))$$

In addition, for each $a \in V_T$ and each $p \in Q$ we call

$$\mathcal{C}((a, p, \cdot)) \rightarrow \epsilon$$

$\mathcal{C}$ is defined as follows.

Case 1.

$$\mathcal{C}(\underline{(X, \alpha p, \phi)} \rightarrow (X_1, \alpha_1, i_1) \ldots (X_n, \alpha_n, i_n))$$

Note that if $n = 0$ then $X \in V_T$, otherwise, $X \in V_N$.

If $\delta(p, X) = (q, \text{up}, w)$ then

$$(X, \alpha p) \rightarrow f((X_1, \alpha_1), \ldots, (X_n, \alpha_n)) \in P'$$

for a new function $f \in F$ where $m(f)$ is defined by

$$f((x_1, \ldots, x_{i_1}), \ldots, (x_1, \ldots, x_{i_n})) = (t_1, \ldots, t_k)$$

where $\phi w \cdot = \cdot t_1 \cdot \ldots \cdot t_k \cdot$. (note that when $i_j = 0$ for some j then $(x_1, \ldots, x_{i_j})$ will appear as ϵ), in addition, for each p' in Q that does not appear in αp call

$$\mathcal{C}(\underline{(X, \alpha p p', \phi w \cdot)} \rightarrow (X_1, \alpha_1, i_1) \ldots (X_n, \alpha_n, i_n))$$

Note that $\cdot$ has been placed after ϕw. This indicates that we have finished with the current component of the tuple.

Otherwise, if $\delta(p, X) = (q, \mathrm{d}(j), w)$ and $1 \leq j \leq n$ then call

$$\mathcal{C}((X, \alpha p, \phi w x_{j,i_j+1}) \rightarrow \\ (X_1, \alpha_1, i_1) \ldots \underline{(X_j, \alpha_j q, i_j + 1)} \ldots (X_n, \alpha_n, i_n))$$

Note that if $X_j \in V_T$ then it is not possible for the machine to move down the tree any further.

Case 2.

$$\mathcal{C}((X, \alpha, \phi) \rightarrow \\ (X_1, \alpha_1, i_1) \ldots \underline{(X_j, \alpha_j p, i_j)} \ldots (X_n, \alpha_n, i_n))$$

If $\delta(p, X_j) = (q, \mathrm{up}, w)$ then call

$$\mathcal{C}(\underline{(X, \alpha q, \phi)} \rightarrow \\ \underline{(X_1, \alpha_1, i_1)} \ldots (X_j, \alpha_j p, i_j) \ldots (X_n, \alpha_n, i_n))$$

Note that ϕ will end with x_{j,i_j} and the i_jth compoent of the yield at X_j will end in w.

Otherwise, if $\delta(p, X_j) = (q, \mathrm{d}(k), w)$ then if $X_j \in V_N$ for each p' in Q and not in $\alpha_j p$ call

$$\mathcal{C}((X, \alpha, \phi) \rightarrow \\ (X_1, \alpha_1, i_1) \ldots \underline{(X_j, \alpha_j p p', i_j)} \ldots (X_n, \alpha_n, i_n))$$

This simulates the next visit to this node (which must be from below) in the (guessed) state p'.

In addition to the productions added by $\mathcal{C}$, include in P' the production $S' \rightarrow (S, q_0 \alpha q_f)$ for each $q_f \in F$ and $\alpha \in Q^*$ such that $a_0 \alpha q_f$ is non-repeating and $\delta(q, S) = (q_f, \mathrm{up}, w)$ for some w where q is the last symbol in $q_0 \alpha$.

A complete proof would establish that the following equivalence holds.

$$(A\alpha) \underset{G'}{\Longrightarrow} \langle w_1, \ldots, w_n \rangle$$

if and only if there is a derivation tree γ of G with root η_r labelled A such that $\alpha = \alpha_1 \ldots \alpha_n$ for some $\alpha_1, \ldots, \alpha_n \in Q^+$ and for each i $(1 \leq i \leq n)$

$$(p_i, \gamma, \eta_r, \epsilon) \vdash_M^* (q_i, \gamma, \uparrow, w_i)$$

where $\alpha_i = p_i \alpha_i' = \alpha_i'' q_i$ for some $\alpha_i', \alpha_i'' \in Q^*$.

Consider the application of this construction to example the dtwt given earlier. The grammar contains the following productions (where productions containing useless nonterminals have been omitted).

$$(S, q_0 q_1 q_3) \rightarrow f_1((A, q_0 q_1 q_2 q_3))$$

where $f_1((x_{1,1}, x_{1,2})) = x_{1,1} x_{1,2}$

$$(A, q_0 q_1 q_2 q_3) \rightarrow f_2((A, q_0 q_1 q_2 q_3))$$

where $f_2((x_{1,1}, x_{1,2})) = (a x_{1,1} b, c x_{1,2} d)$

$$(A, q_0 q_1 q_2 q_3) \rightarrow f_3((e, q_0 q_2))$$

where $f_3((x_{1,1}, x_{1,2})) = (a x_{1,1} b, c x_{1,2} d)$

$$(e, q_0 q_2) \rightarrow f_4()$$

where $f_4() = (\epsilon, \epsilon)$.

By renaming nonterminal we get the four productions

$$S \rightarrow f_1(A) \qquad A \rightarrow f_2(A)$$

$$A \rightarrow f_3(e) \qquad e \rightarrow f_4()$$

LCFRL $\subseteq$ OUT(DTWT)

Consider the gcfg $G = (V_N, S, F, P)$ and mapping m that interprets the symbols in F. Without loss of generality we assume that no nonterminal appears more than once on the right of a production and that for each $A \in V_N$ there is some $rank(A) = k$ such that only k-tuples are derived from A.

We define a dtwt $M = (Q, \Sigma, G', V_T, \delta, q_0, F)$ where G' is a context-free grammar that generates derivation trees of G in the following way. A derivation involving the use of a production π will be represented by a tree whose root is labelled by $\pi = A \rightarrow f(A_1, \ldots, A_n)$ with n subtrees encoding the derivations from $A_1, \ldots, A_n$. The roots of these subtrees will be labelled by the n productions used to rewrite the $A_1, \ldots, A_n$. Let $lhs(\pi) = A$ and $rhs(\pi) = \{ A_1, \ldots, A_n \}$.

The dtwt M walks around a derivation tree γ of G' in such a way that it outputs the yield of γ. Each subtree of γ rooted at a node η labelled by the production π will be visited on $k = rank(lhs(\pi))$ occasions by M. During the ith visit to the subtree M will output the ith component of the tuple. We therefore include in Q k states $\{ 1, \ldots, k \}$ that are used to keep track of which tuple is being considered. This will generally involve visiting children of η as determined by the equation used to define function used in π. Additional states in Q are used to keep track of these visits as follows. When the lth child of η has finished its mth component, M will move back up to η in state (A_l, m). Since no nonterminal appears twice on the right of a production it is possible for M to determine the value of l from A_l while at η.

For each production $\pi = A \rightarrow f(A_1, \ldots, A_n) \in P$ where f is interpreted as the function defined by the equation

$$f(\langle x_{1,1}, \ldots, x_{1,k_1} \rangle, \ldots, \langle x_{n,1}, \ldots, x_{n,k_n} \rangle) = \langle t_1, \ldots, t_k \rangle$$

we include the following components in the definition of δ.

For each i $(1 \leq i \leq k)$

- if $t_i = w x_{l,m} \phi$, where w is a possibly empty terminal string then let
$$\delta(i, \pi) = (m, down(l), w)$$
- if $t_i = w$ (in which case it is time to move up the tree) let
$$\delta(i, \pi) = ((lhs(\pi), i), up, w)$$

For each $B \in rhs(\pi)$ and each m, $1 \leq m \leq rank(B)$, let

$$\delta((B, m), \pi) = (q, \text{move}, w)$$

where (q, move, w) is determined as follows. For some unique l we know that B is the lth nonterminal on the right-hand side of π. There is a unique t_i such that $t_i = \phi_1 x_{l,m} w \phi_2$ where w is a possibly empty string of terminals.

Case 1: ϕ_2 is empty
In this case the ith component of the current node is complete. Thus, $q = (lhs(\pi), i)$ and move = up.
Case 2: ϕ_2 begins with the variable $x_{l',m'}$
In this case the machine M must find the m'th component of the l'th child. Thus, $q = m'$ and move = d(l').

It should be clear that the start state q_0 should be 1 and the set of final states $F = \{ (S, rank(S)) \}$.

A complete proof would involve verifying that the following equivalence holds.

$$(A\alpha) \underset{G}{\Longrightarrow} \langle w_1, \ldots, w_n \rangle$$

if and only if there is a derivation tree γ of G' with root η_r labelled π such that $lhs(\pi) = A$ and for each i $(1 \leq i \leq n)$

$$(i, \gamma, \eta_r, \epsilon) \vdash_M^* ((A, i), \gamma, \uparrow, w_i)$$

We apply the construction to the grammar produced in the illustration of the first construction. First, we name the productions of the grammar

$$\pi_1 = S \rightarrow f_1(A) \qquad \pi_2 = A \rightarrow f_2(A)$$

$$\pi_3 = A \rightarrow f_3(e) \qquad \pi_4 = e \rightarrow f_4()$$

The construction gives a machine in which the function δ is defined as follows.

$$\begin{array}{ll}
\delta(1,\pi_1) = (1, \mathrm{d}(1), \epsilon) & \delta((A,1),\pi_1) = (2, \mathrm{d}(1), \epsilon) \\
\delta(1,\pi_2) = (1, \mathrm{d}(1), a) & \delta((A,2),\pi_1) = ((S,1), \mathrm{up}, \epsilon) \\
\delta(2,\pi_2) = (2, \mathrm{d}(1), c) & \delta((A,1),\pi_2) = ((A,1), \mathrm{up}, b) \\
\delta(1,\pi_3) = (1, \mathrm{d}(1), a) & \delta((A,2),\pi_2) = ((A,2), \mathrm{up}, d) \\
\delta(2,\pi_3) = (2, \mathrm{d}(1), c) & \delta((e,1),\pi_3) = ((A,1), \mathrm{up}, b) \\
\delta(1,\pi_4) = ((e,1), \mathrm{up}, \epsilon) & \delta((e,2),\pi_3) = ((A,2), \mathrm{up}, d) \\
\delta(2,\pi_4) = ((e,2), \mathrm{up}, \epsilon) &
\end{array}$$

The context-free grammar whose derivation trees are to be transduced has the following productions.

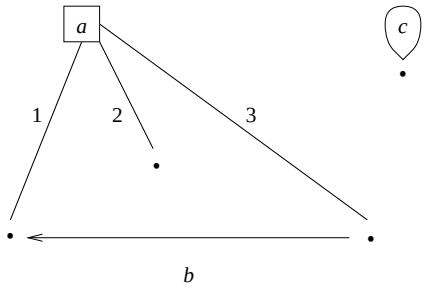
$$\pi_1 \rightarrow \pi_2 \qquad \pi_1 \rightarrow \pi_3$$

$$\pi_2 \rightarrow \pi_2 \qquad \pi_2 \rightarrow \pi_3 \qquad \pi_3 \rightarrow \pi_4$$

Context-Free Hypergraph Grammars

In this section we describe context-free hypergraph gramars since they are an example of a lcfrs involving the manipulation of graphs. zThe class of string languages generated by context-free hypergraph grammars is equal to OUT(DTWT) [3] and the above result shows that they are also equal to LCFRS.

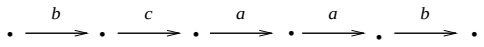
A directed hypergraph is similar to a standard graph except that its (hyper)edges need not simply go from one node to another but may be incident with any number of nodes. If an edge is incident with n nodes then it is a n-edge. The n nodes that are incident to some edge are linearly ordered. For example, in the figure below, dots denote nodes and labelled square boxes are edges. The edge labelled a is a 3-edge, the edge labelled b is a 2-edge and the edge labelled c is a 1-edge. When the number of nodes incident to an edge exceeds 2, numbered tentacles are used to indicate the nodes that are incident to the edge. The numbers associated with the tentacles coming from an edge indicate the linear order of the nodes that are incident to that edge. 2-edges are shown in the standard way and 1-edges can be used as a way of associating labels with nodes as shown.



We denote a **hypergraph** as a five tuple $H = (V, E, \Sigma, \mathit{incident}, \mathit{label})$ where
V is a finite set of nodes,
E is a finite set of edges,
Σ is a finite set of edge labels,
$\mathit{incident} : E \rightarrow V^*$ is the incidence function and
$\mathit{label} : E \rightarrow \Sigma$ is the edge labelling function

For example, in the above graph
$V = \{ v_1, v_2, v_3, v_4 \}$, $E = \{ e_1, e_2, e_3 \}$,
$\Sigma = \{ a, b, c \}$, $\mathit{incident}(e_1) = (v_2, v_1, v_4)$,
$\mathit{incident}(e_2) = (v_4, v_1)$, $\mathit{incident}(e_3) = (v_3)$,
$\mathit{label}(e_1) = a$, $\mathit{label}(e_2) = b$ and $\mathit{label}(e_3) = c$.

A string can be encoded with a **string hypergraph** [5]. The string *bcaab* is encoded with the following graph.



We denote a **context-free hypergraph grammar** (cfhg) as four tuple $G = (V_N, V_T, S, P)$ where
V_N is a finite nonterminal alphabet,
V_T is a finite terminal alphabet,
$S \in V_N$ is the initial nonterminal and
P is a finite set of productions $e \rightarrow H$ where

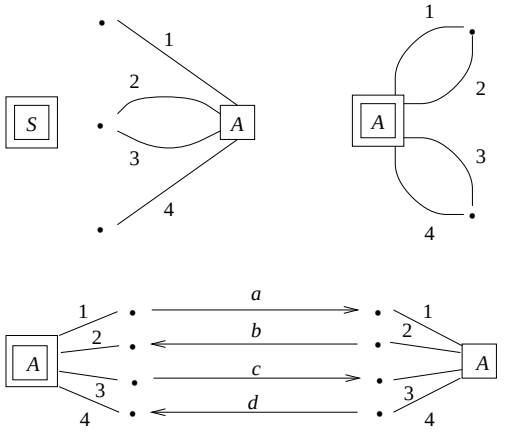
$$H = (V, E, V_N \cup V_T, \mathit{incident}, \mathit{label})$$

is a hypergraph and
$e \in E$ is a nonterminal edge in H, i.e., $\mathit{label}(e) \in V_N$.

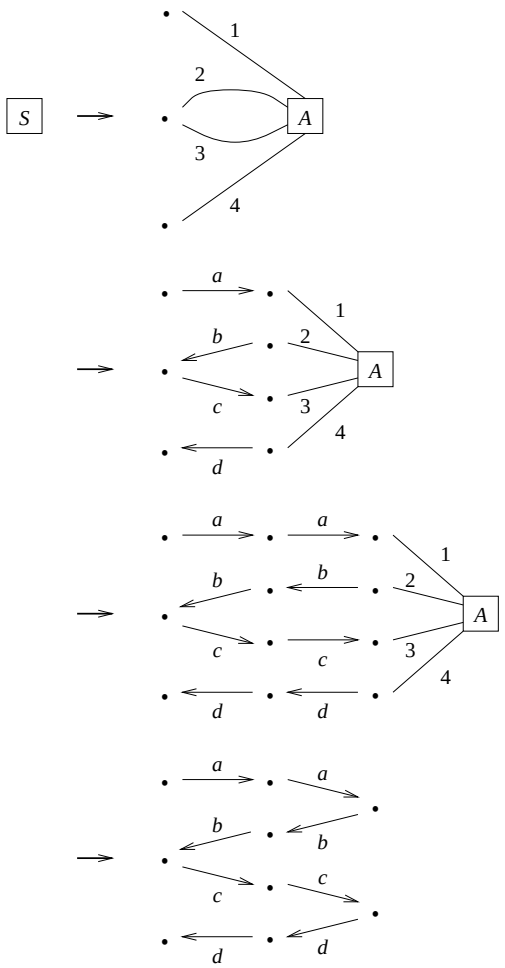
Consider the application of a production $e \rightarrow H$ to a graph H' at a node e' in H' with the same nonterminal label as e. The resulting graph is obtained from H' by replacing e' by the graph H with e removed from it. This involves merging of nodes. In particular, the ith node incident with e is merged with the ith node incident with e'. We require that all edges with the same label have the same number of incident nodes. A derivation begins with a graph containing a single edge labelled S and no edges. A derivation is completed when there are no nonterminal nodes in the graph.

The string language associated with a cfhg G is denoted STR(G). The class of languages generated by all cfhg is denoted STR(CFHG).

Due to lack of space, rather than a complete formal definition of cfhg derivations, we present an illustrative example. Consider the three productions shown below. Note that the edge on the left-hand-side of the production is indicated with a double box.



Below we show the steps in a derivation of the string *aabbccdd* involving these productions. Note that the set of graphs derived corresponds to the string language $\{ a^n b^n c^n d^n \mid n > 0 \}$.



It is clear from their definition that cfhg satisfy the conditions for being a lcfrs given earlier. As has been observed [3] it is possible to represent the set of derivations of a given cfhg with a set of trees that can be generated by a context-free grammar. The composition operation of cfhg in which a node is replaced by a graph is clearly size-preserving since it does not involve duplication or deletion of an unbounded number of nodes or edges.

Additional Remarks

We end by elaborating on the relationship between lcfrs, dtwt and cfhg in terms of the following complexity measures.

- The maximum of $rank(A)$ nonterminals A of a gcfg. Let LCFRL_k be the class of languages generated by gcfg of some lcfrs whose nonterminals have rank k or less, i.e., derive at most k tuples.
- The **crossing number** of a dtwt M. This is the maximum number of times that it visits any given subtree of an input tree. Let $\mathrm{OUT}(\mathrm{DTWT}_k)$ be the class of languages output by dtwt whose crossing number does not exceed k.
- The maximum number of tentacles of the nonterminals of a cfhg. Let $\mathrm{STR}(\mathrm{CFHG}_k)$ be the class of languages associated with cfhg whose nonterminals have at most k tentacles.

It has been shown (Theorem 6.1 in [3]) that

$$\mathrm{OUT}(\mathrm{DTWT}_k) = \mathrm{STR}(\mathrm{CFHG}_{2k}) = \mathrm{STR}(\mathrm{CFHG}_{2k+1})$$

It can be seen from the above constructions that

$$\begin{aligned} \mathrm{LCFRL}_k &= \mathrm{OUT}(\mathrm{DTWT}_k) \\ &= \mathrm{STR}(\mathrm{CFHG}_{2k}) \\ &= \mathrm{STR}(\mathrm{CFHG}_{2k+1}) \end{aligned}$$

References

[1] Alfred Aho and Jeffrey Ullman. Translations on a context-free grammar. *Information and Control*, 19:439–475, 1971.

[2] M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20:83–127, 1987.

[3] J. Engelfriet and L. Heyker. The string generating power of context-free hypergraph grammars. *Journal of Computer Systems Science*, 43:328–360, 1991.

[4] J. Engelfriet, G. Rozenburg, and G. Slutzki. Tree transducers, l systems, and two-way machines. *Journal of Computer Systems Science*, 20:150–202, 1980.

[5] A. Habel and H. Kreowski. Some structural aspects of hypergraph languages generated by hyperedge replacement. In *STACS*, 1987.

[6] A. K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer Systems Science*, 10(1):136–163, 1975.

[7] T. Kasami, H. Seki, and M. Fujii. Generalized context-free grammars, multiple context-free grammars and head grammars. Technical report, Department of Information and Computer Science, Osaka University, Osaka, Japan, 1988.

[8] Carl Pollard. *Generalized Phrase Structure Grammars, Head Grammars and Natural Language*. PhD thesis, Stanford University, 1984.

[9] K. Vijay-Shanker, David Weir, and Aravind Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics*, pages 104–111, 1987.

[10] David Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1988. Available as Technical Report MS-CIS-88-74.