Multiple Context Free Grammars

Siddharth Krishna

September 14, 2013

Abstract

Multiple context-free grammars (MCFGs) are a generalization of context-free grammars that deals with tuples of strings. This is a brief survey of MCFGs, their properties, and their relations to other formalisms. We outline simple constructions to convert MCFGs to Hyperedge Replacement Grammars (HRGs) and to Deterministic Tree Walking Transducers (DTWTs), and also to simulate DTWTs by MCFGs. We also describe a simple way to recognize bounded split-width MNWs using MCFGs, which gives us another way to prove MSO-decidability over bounded split-width MNWs.

1 Introduction

This is a report on an internship done with Paul Gastin, Aiswarya Cyriac and K Narayan Kumar in the Laboratoire Spécification et Vérification (LSV) at the École normale supérieure de Cachan (ENS Cachan) in the summer of 2013.

We studied the formalism known as Multiple Context Free Grammars (MCFGs), initially to see if there was any connection to the notion of split-width over Multiply Nested Words (MNWs). It turned out that MCFGs are actually a robust formalism, with connections to many other types of grammars and machine models. We studied two equivalent formalisms, namely Hyperedge Replacement Grammars (HRGs) and to Deterministic Tree Walking Transducers (DTWTs). Although in the literature, the equivalence is proved in a rather long-winded way through other formalisms, we found simple constructions to convert MCFGs to HRGs and to DTWTs, and also back from DTWTs to MCFGs.

To answer our initial question, we discovered an easy way to describe bounded split-width MNWs using MCFGs, which gives us another way to prove decidability of MSO. We also tried to study some machine models for MCFGs, hoping that this would allow us to formulate a restriction in a machine theoretic sense that captured bounded split-width, another important open question in that area.

2 Multiple Context Free Grammars

MCFGs were proposed in 1991 by Seki, Matsumura, Fujii and Kasami [10], to describe the syntax of natural languages. The class of languages generated by MCFGs properly includes the class of context-free languages and is properly included in the class of context-sensitive languages.

Unlike CFGs, while defining MCFGs we look at non-terminals as predicates and rewriting rules as inference rules. Thus in a CFG, all non-terminals are predicates of arity 1. MCFGs generalize CFGs in the sense that they allow non-terminals to have arbitrary arity. Thus each non-terminal derives not a string, but a tuple of strings.

For example, consider the MCFG with the following production rules:

$$\begin{aligned} S(xy) &\leftarrow A(x,y) \\ A(axb,cyd) &\leftarrow A(x,y) \\ A(ab,cd) & \end{aligned}$$

This generates the non-context-free language $\{a^n b^n b^n d^n | n \geq 1\}$.

2.1 Motivation

It is widely known that CFGs are an inadequate formalism to describe the structures present in natural language. For example, cross-serial dependencies in natural language, eg Swiss German, cannot be captured by CFGs (for a nice example, see [8]). Aravind K. Joshi proposed a formalism known as Mildly Context Sensitive Grammar formalism, hoping to find a class of grammars that could capture natural language but still have tractable parsing properties. The criteria he proposed are:

1. Properly extends CFGs,
2. Poly-time parsable,
3. Semilinear,
4. Exhibits limited cross serial dependencies.

MCFGs seemed to fit all the above criteria, and also turned out to be equivalent to a large number of other MCSG formalisms, and thus a near-consensus emerged, identifying MCSGs with MCFGs. But recently this consensus is being questioned, especially after a recent result by Sylvain Salvati [7] that the language MIX, that was not supposed to be mildly context-sensitive, could in fact be recognized by a 2-MCFG.

2.2 Definition

A m-MCFG(r) is a 4-tuple $G = (N, T, P, S)$ such that:

- N is a finite ranked alphabet of non-terminals of maximum rank m. I.e., each non-terminal is a k-ary predicate with $k \leq m$.
- T is a finite alphabet of terminals.
- P is a set of rules of the form:

  $$A_0(s_1, \ldots, s_{k_0}) \leftarrow A_1(x_1^1, \ldots, x_{k_1}^1) \cdots A_n(x_1^n, \ldots, x_{k_n}^n)$$

  where

  – A_i is a non-terminal of arity k_i for every i, $n \leq r$,
  – the variables x_j^i are pairwise distinct,
  – the strings s_i are in $(T \cup X)^*$ with $X = \cup_{i,j} x_j^i$,

– each variable x^i_j has at most one appearance in $s_1 \cdots s_k$.

- S is a non-terminal of rank 1.

Given a rule such as the one above, we define the *dimension* to be the maximum arity of the A_is, the *rank* to be n, and the *degree* to be the sum of the arities of A_is. The dimension, rank and degree of the MCFG G is the maximum dimension, rank, degree (respectively) of all its rules.

Derivations are defined in the natural way: for any rule without a RHS, the LHS is said to be derivable; and for any rule, if all the predicates on the RHS are derivable for some tuples of strings, then the LHS (with appropriate substitutions) is said to be derivable. The language derived by a MCFG G is simply: $L(G) = \{w | S(w) \text{ is derivable}\}$. A language that can be recognized by a MCFG is called a Multiple Context Free Language (MCFL). A language recognized by a MCFG of dimension q and rank r is a q-MCFL(r).

2.3 Types

Definition 2.1 *An MCFG is called* non-deleting *if in every rule, every variable x^i_j appears* exactly once *in the LHS $s_1 \cdots s_k$.*

For example, the rule

$$A(xy) \leftarrow B(x)C(y)$$

is non-deleting, whereas

$$A(ay) \leftarrow B(x)C(y)$$

is not.

Definition 2.2 *An MCFG is called* non-permuting *if in every rule, there does not exist i, j, l such that $i < j$ but x^l_j appears before x^l_i in the string $s_1 \cdots s_k$.*

For example, the rule

$$A(zx, y) \leftarrow B(x, y)C(z)$$

is non-permuting, whereas

$$A(zy, x) \leftarrow B(x, y)C(z)$$

is not.

Theorem 2.3 *(Kracht 2003) For every MCFG G there exists a non-permuting and non-deleting MCFG G' such that $L(G) = L(G')$.*

Definition 2.4 *An MCFG is called* well-nested *if it is non-deleting and non-permuting and satisfies the following condition: for every rule, there does not exist $l_1, l_2, i_1, i_2, j_1, j_2$ such that the variables $x^{l_1}_{i_1}, x^{l_2}_{j_1}, x^{l_1}_{i_2}, x^{l_2}_{i_2}$ appear in that order in the string $s_1 \cdots s_k$.*

For example, the rule

$$A(x_1, x_2y_1y_2) \leftarrow B(x_1, x_2)C(y_1, y_2)$$

is well nested, whereas

$$A(x_1y_1, x_2y_2) \leftarrow B(x_1, x_2)C(y_1, y_2)$$

is not.

Unlike with the previous two restrictions, well-nested MCFGs are strictly weaker than general MCFGs. There are examples of languages in the literature that are shown to be MCFLs, but not well-nested MCFLs (Engelfriet and Skyum 1976, Staudacher 1993, Michaelis 2005).

2.4 Properties

- CFL $\subsetneq$ MCFL $\subsetneq$ CSL (Context Sensitive Languages). Note that all are strict inclusions.
- MCFLs are semilinear.
- MCFLs are a full AFL: they are a class of languages closed under homomorphisms, inverse homomorphisms, intersection with regular sets, union, and Kleene closure.
- MCFLs are not closed under intersection.
- Decidability:
  - Emptiness ($L(G) = \emptyset$?) : $O(|G|)$-time decidable
  - Recognition ($w \in L(G)$?) : poly-time decidable
  - Inclusion ($L(G_1) \subseteq L(G_2)$?) : undecidable
- Pumping lemma for MCFLs:

  Lemma 2.5

  $$\begin{aligned}&\forall L \in q\text{-}MCFL\ \ \exists n \geq 1\ \exists z \in L, |z| \geq n\\ &\exists\ a\ partition\ z = u_1v_1w_1s_1 \cdots u_qv_qw_qs_qu_{q+1}, \sum v_js_j \geq 1\\ &\forall i \geq 0, z_i = u_1v_1^iw_1s_1^i \cdots u_qv_q^iw_qs_q^iu_{q+1} \in L.\end{aligned}$$

  Note that this is a weaker version than the normal pumping lemma we have for CFLs for example, because of the $\exists$ symbol highlighted in red. There is a stronger version of the pumping lemma with this particular $\exists$ replaced by a $\forall$, but that holds only for well nested MCFLs.
- There is an infinite (strict) hierarchy on the dimension of MCFLs. That is, 1-MCFL (= CFL) $\subsetneq$ 2-MCFL $\subsetneq \cdots q$-MCFL $\subsetneq (q+1)$-MCFL $\cdots$.
- There is also a strict infinite hierarchy on ranks for a fixed dimension $q \geq 2$ (1-MCFL is basically equal to CFLs, where we have a normal form that collapses the rank down to 2). [6]
- If we do not fix the dimension, then the hierarchy on ranks collapses to the second level. This is because of a trade-off theorem that says that q-MCFL$(r) \subseteq (k+1)q$-MCFL$(r-k)$. [6]

2.5 Connections To Other Formalisms

I talked to Sylvain Salvati at Laboratoire Bordelais de Recherche en Informatique (LaBRI), Université Bordeaux I, about MCFGs. He gave me many valuable references and tips. According to him, MCFLs are equivalent to a variety of class of languages, including:

1. String language of Hyperedge Replacement Grammars (HRGs)
2. Output of Deterministic Tree Walking Transducers (DTWTs)
3. Linear Context Free Rewriting Systems
4. Second order string Abstract Categorial Grammars

Of these equivalences, the relation to HRGs is particularly useful because graph languages generated by HRGs have *bounded tree width* and so with a little work on that construction, we see that even graph structures generated by MCFGs should have bounded tree width. This in fact, gave us another way to prove decidable MSO for MNWs with bounded split-width, as detailed below.

2.6 Expressing Bounded Split-Width Using MCFGs

We also found that with a suitable encoding, we could construct an MCFG that generated only Multiply-Nested Words (MNWs) that had a split-width bounded by some k (for definitions see [2]).

Let s be the total number of stacks under consideration. The encoding of MNWs as strings is as follows: we denote every internal action by the terminal symbol a, and every push onto a stack i is denoted by b_i and every corresponding pop by d_i. Note that since within every stack the MNW is well nested, it is easy to recover the information of which push corresponds to which pop. To encode a m-split MNW (m-SMNW), we use an m-tuple of strings.

Now the non-terminals in our new MCFG consist of symbols $S_{m,f}$ for every $1 \leq m \leq k$ and where f is a collection of functions $\{f_i : [m] \times [m] \to \{0,1\} | 1 \leq i \leq s\}$. $S_{m,f}$ has arity m and contains all tuples that are m-SMNWs and have split-width bounded by k, and who's nesting edges satisfy f. This means that in some valid $S_{m,f}(x_1, \cdots, x_m)$, $f_i(p,q) = 1$ if and only if there is a nesting edge between x_p and x_q. The production rules mirror the algebraic definition of how k bounded split-width MNWs are built up in (as in [2]), and are as follows:

- $S_{1,f}(a)$ for the identically zero f.
- $S_{2,f}(b_i, d_i)$ for the f in which all f_js are zero, except $f_i(1,2) = 1$.
- $S_{m+n,f}(s_1, \cdots, s_{m+n}) \leftarrow S_{m,g}(x_1, \cdots, x_m) S_{n,h}(y_1, \cdots, y_n)$ where $s_1, \cdots, s_{m+n}$ is a permutation of $x_1, \cdots, x_m, y_1, \cdots, y_n$, but we only allow permutations that preserve well-nestednes, by the following constraint: there does not exist i, j, p_1, p_2, q_1, q_2 such that $g_i(p_1, q_1) = 1 = h_j(p_2, q_2)$ and $x_{p_1}, y_{p_2}, x_{q_1}, y_{q_2}$ appear in that order in the string $s_1 \cdots s_{m+n}$. Also, we update f to have the new nested edge information as follows: $f_i(p,q) = 1$ if and only if $s_p = x_{p'}$, $s_q = x_{q'}$ and $g_i(p', q') = 1$ (and symmetrically for y and h). This emulates the shuffle operation.
- $S_{n,f}(s_1, \cdots, s_n) \leftarrow S_{m,g}(x_1, \cdots, x_m)$ such that $n < m$ and $s_1 \cdots s_n = x_1 \cdots x_m$ as strings. Also, f is updated as follows: $f_i(p,q) = 1$ if and only if s_p contains $x_{p'}$, s_q contains $x_{q'}$ and $g_i(p', q') = 1$. This emulates the merge operation.

Although not a rigorous proof, since we followed the definition of split-width almost exactly, it is easy to see that the tuples of strings generated using these non-terminals are exactly the SMNWs of split-width bounded by k. Using the relation to HRGs explained below, we get that hence MNWs with bounded split-width have bounded tree width, and hence decidable MSO.

3 Hyperedge Replacement Grammars

HRGs are a generalization of graph grammars. They are also called Context-Free Hypergraph Grammars. For a comprehensive survey see [3], or for a paper relating it to DTWTs see [4].

Definition 3.1 *Let C be a finite alphabet such that each $A \in C$ has a type$(A) \in \mathbb{N}$. A* hypergraph *over C is a tuple $H = (V, E, \texttt{lab}, \texttt{att}, \texttt{ext})$ such that:*

- *V and E are finite sets of nodes ad hyperedges respectively,*
- *$\texttt{lab}$ is a labelling function from $E \to C$,*
- *$\texttt{att}$ is a function from $E \to V^*$ that denotes the nodes attached to each hyperedge, complying with $|\texttt{att}(e)| = \mathit{type}(\texttt{lab}(e))$ for every $e \in E$,*
- *$\texttt{ext} \in V^*$ specifies which nodes are external nodes.*

When there are multiple hypergraphs in a certain context, we use a subscript to denote which function (for example $\texttt{lab}_H, \texttt{att}_H, \texttt{ext}_H$) we are referring to. We define the type of the hypergraph to be $\mathit{type}(H) = |\texttt{ext}_H|$.

To define graph grammars, we first need to define the notion of *hyperedge replacement.* Given a hyperedge e in a hypergraph H and another hypergraph H' such that $\mathit{type}_H(e) = \mathit{type}(H')$, the replacement of e in H by H' yields the hypergaph $H[e/H']$ which is constructed as follows:

- Build $H - e$ by removing e from H.
- Take the disjoint union of $H - e$ and H', renaming edges and nodes if necessary, and let the external nodes be that of H.
- For all $i \in [\mathit{type}_H(e)]$, identify the i-th external node of H' with the i-th attached node of e.

Definition 3.2 *A* hyperedge replacement grammar *(HRG) is a tuple $G = (N, \Sigma, R, S)$, where*

- *N and $\Sigma \subseteq C$ are finite sets of non-terminal and terminal symbols respectively,*
- *R is a set of rules of the form $A \to H$ with $A \in N$ and H a hypergraph such that $\mathit{type}(A) = \mathit{type}(H)$,*
- *$S \in N$ is the start non-terminal.*

Given a label of a hyperedge $A \in C$, we have the canonical hypergraph called the *handle* denoted $A^\bullet$, which is a hypergraph H consisting of just a single hyperedge e labelled A, all nodes are external nodes, and $\mathtt{att}_H(e) = \mathtt{ext}_H$.

The derivation in a HRG starts with $S^\bullet$. If H is a hypergraph with $e \in E_H$ and $\mathtt{lab}_H(e) \to H'$ is a rule, then there is a derivation step $H \to H[e/H']$. The generated language is $L(G) = \{H | S^\bullet \to^* H\}$.

A string can be represented as a sequence of labelled edges in a directed path graph. Thus it is easy to see that every context free string grammar can be viewed as a graph grammar. But more importantly, graph grammars can be used as a type of string grammar by restricting attention to those graph grammars that generate strings only. This is a-priori more powerful, since sentential forms of such a grammar may be arbitrary graphs. It is also known that it is decidable whether an arbitrary HRG generates a string language [1].

An important property of HRGs is that the graph languages generated by them have *bounded tree width* [5].

3.1 Simulating MCFGs

It is quite easy to see that every MCFL can be recognized by a HRG. Given an MCFG $G = (N, T, P, S)$, we construct a HRG $G' = (N, \Sigma = T, R, S)$ with the same terminal and non-terminal symbols (and the same start symbol). Every non-terminal in the HRG is simply a non-terminal edge of type twice the arity of the corresponding non-terminal in the MCFG. For a non-terminal of arity k, we denote the attachments of the hyperedge as $1, 1', 2, 2', \ldots, k, k'$. The idea is that if $A(w_1, \ldots, w_k)$ is derivable in the MCFG, then starting from $A^\bullet$, one can derive the string graphs for $w_1, \ldots, w_k$ such that w_1 is contained between the nodes to which 1 and $1'$ were attached to. Every rule in the MCFG that looks like:

$$A_0(s_1, \ldots, s_{k_0}) \leftarrow A_1(x^1_1, \ldots, x^1_{k_1}) \cdots A_n(x^n_1, \ldots, x^n_{k_n})$$

is replaced by a rule $A^\bullet \to H$ in the HRG, where H is a hypergraph defined as follows:

For each s_i, there are $|s_i| + 1$ nodes in a "cluster", connected by $|S_i|$ edges. If the jth character in s_i is a terminal $a \in T$, then the jth edge is a simple edge of type 2 labelled a. If it is x^l_m then the two nodes are attached to the hyperedge labelled A_l, between the attachments m and m'. The external nodes of H are the two nodes at the edge of each "cluster".

As an example, consider the following MCFG that generates the language $a^n b^m a^n b^m$:

$$\begin{aligned}
S(x_1 y_1 x_2 y_2) &\leftarrow A(x_1, x_2) B(y_1, y_2)\\
A(a x_1, a x_2) &\leftarrow A(x_1, x_2)\\
B(a y_1, a y_2) &\leftarrow B(y_1, y_2)\\
A(a, a)&\\
B(b, b)&
\end{aligned}$$

Using the construction above, we get the following HRG (shown in Figure 1), where the letters Y, y are used as a short way of expressing two similar rules for A, a and B, b respectively. Each of the three rules shown below are derived from the corresponding rules in the MCFG above, using the strategy outlined above.

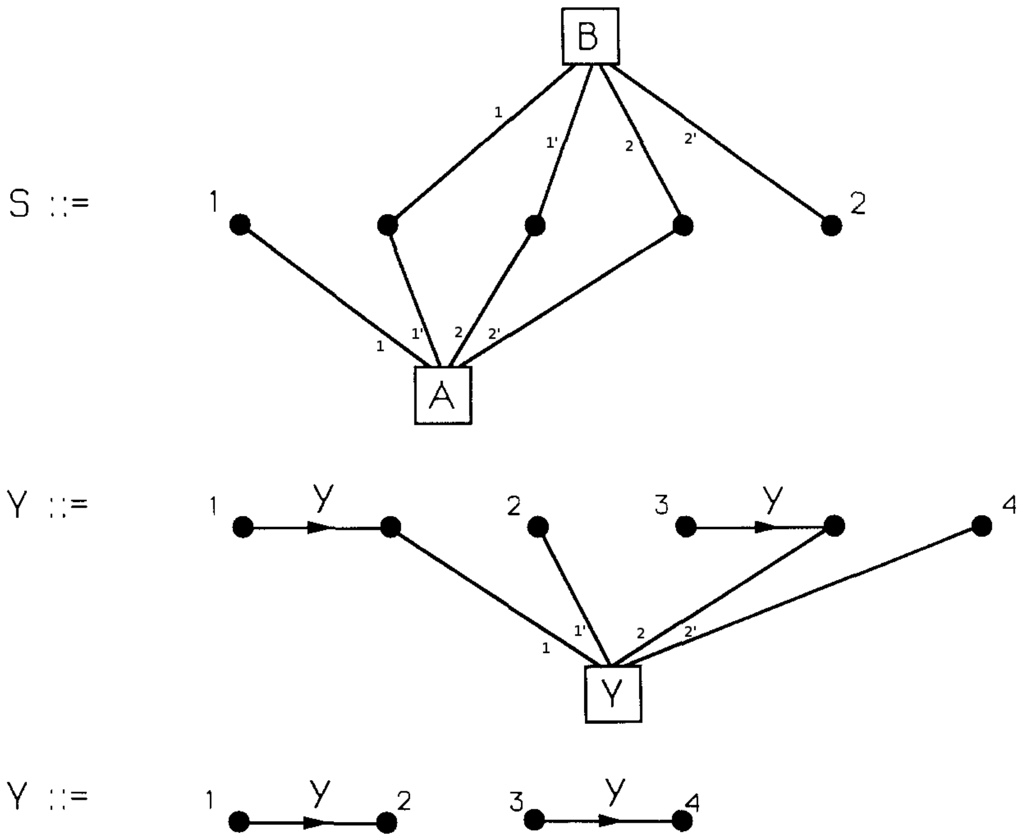

Figure 1: A HRG for the MCFG generating $a^n b^m a^n b^m$

4 Deterministic Tree Walking Transducers

A deterministic tree-walking transducer is an automaton with a finite control, an input tree, and an output tape. See [4] for a definition, and for the equivalence to HRGs.

Definition 4.1 *A* deterministic tree-walking transducer *(DTWT) is a tuple* $M = (Q, G, \Delta, \delta, q_0, F)$*, where*

- *Q is a finite set of states, with $q_0 \in Q$ the initial state and $F \subseteq Q$ the set of final states,*
- *$G = (V_N, V_T, P, S_G)$ a context-free grammar without ε-productions,*
- *Δ is the output alphabet,*
- *$\delta : Q \times (V_N \cup V_T) \to Q \times D \times \Delta^*$ is the transition function with $D = \{stay,\ up\} \cup \{down(k) | k \in \mathbb{N}, k \geq 1\}$.*

A configuration of M is a tuple (q, t, n, w), where $q \in Q$, t is a complete derivation tree of G, n is either a node of t or $n =$"father of the root", and $w \in D^*$. Here, "father of the root" just denotes a special object that is not a node of t. If n is a node of t, X is the label of n, and $\delta(q, X) = (q', d, y)$, then M moves in one step from configuration (q, t, n, w) to configuration (q', t, n', wy), where $n' = n$ in case $d =$ stay, n' is the father of n in case $d =$up (in particular, if n is the root of t, then $n' =$ "father of the root"), and n' is the kth son of n (if it exists) in case $d =$ down(k). A computation of M consists of a sequence of such moves, starting in an initial configuration; it is successful if it ends in a final configuration. For a given input tree t, the initial configuration is (q_o, t, r, ε) where r is the root of t. A final configuration is any configuration $(q, t,$"father of the root"$, w)$ with $q \in F$; it is said to have output w. For a complete derivation tree t we define $M(t)$ to be the output of the final configuration of M, which it reaches in the computation that starts in the initial configuration for t. Since the computation of M may not end in a final configuration, $M(t)$ may be undefined. The output language of M (or the language generated by M), denoted OUT(M), is OUT(M) $= \{w \in A^* | w = M(t)$ for some complete derivation tree t of $G\}$.

By OUT(DTWT) we denote the class of all output languages of DTWTs. A result of Engelfriet and Heyker (1991) shows that this class of languages is exactly the class of string languages generated by HRGs [4].

We found two simple constructions, described below, that show the equivalence between MCFGs and DTWTs.

4.1 Simulating MCFGs

We assume first that the given MCFG is in a normal form such that there is only one appearance of each non-terminal in the RHS of every rule. This can be easily done by having multiple copies of non-terminals that appear more than once in the RHS of a rule.

The idea is that the DTWT we construct walks on the derivation tree of some derivation from the MCFG, and tries to output the word generated in a linear order. The complication arises because in an MCFG, the word is generated in not necessarily a linear order, because of the power of deriving tuples. But since we have the normal form above, it is enough for our machine to only remember which argument of which nonterminal it is currently looking for, and what position of that argument it is in.

Given an MCFG $G = (N, \Sigma, P, S)$, we construct a DTWT on the CFG $G' = (V_N, V_T, P', S_{in})$ where

$$\begin{aligned}
V_T &= \{\pi \in P | \mathrm{RHS}(\pi) = \emptyset\} \\
V_N &= P \setminus V_T \\
P &= \{\pi \to \pi_1 \cdots \pi_k | \pi : A \leftarrow A_1 \cdots A_k, \text{ and for every } 1 \leq i \leq k, \mathrm{LHS}(\pi_i) = A_i\} \\
S_{in} &= \{\pi \in P | \mathrm{LHS}(\pi) = S\}
\end{aligned}$$

The DTWT we define is $M = (Q, G', \Sigma, \delta, q_0, F)$, where

$$\begin{aligned}
Q =&\{(arg, pos) | 1 \leq arg \leq \text{dimension of } G, 0 \leq pos \leq \text{max length of rule in } G \text{ as string}\} \\
&\cup \{find(A, arg) | A \in N, 1 \leq arg \leq \text{dimension of } G\}
\end{aligned}$$

and the transition function δ is defined as follows

- $((arg, pos), \pi) \mapsto ((arg, pos{+}1), stay, a)$ if $\pi : A_0(s_1, \ldots, s_{k_0}) \leftarrow A_1(x_1^1, \ldots, x_{k_1}^1) \cdots A_n(x_1^n, \ldots, x_{k_n}^n)$ and the pos-th character of s_{arg} is $a \in \Sigma$.
- $((arg, pos), \pi) \mapsto ((arg', 0), down(k), \varepsilon)$ if π is as above and the pos-th character of s_{arg} is $x^k_{arg'}$.
- $((arg, pos), \pi) \mapsto (find(A, arg), up, \varepsilon)$ if π is as above, and pos is equal to the length of the string $s_1 \cdots s_k$.
- $(find(A, arg), \varepsilon) \mapsto ((arg', pos), stay, \varepsilon)$ where if π is as above, and A is the k-th non-terminal in the RHS, then the variable x^k_{arg} is the $(pos - 1)$-th character in $s_{arg'}$.

The DTWT starts at the root of the derivation tree at the state $(1, 0)$, and finishes at the father of the root, with state $find(S, arity(S))$.

4.2 Simulating DTWTs

We are given a DTWT: $M = (Q, G, \Delta, \delta, q_0, F)$. We take every derivation tree of G on which M has a valid run and basically add the run information to the labels of the tree. We do this as follows: for every node labelled L, we add on a sequence of tuples from $(Q \times D)$ where $D = \{\text{stay, up}\} \cup \{\text{down}(k) | k \in \mathbb{N}, k \geq 1\}$. There is one tuple for each time a node is visited, and it stores the state when the run entered the node, and the direction M moved in. Note that since M is deterministic, the size of this sequence is bounded by the number of states, $|Q|$, for if the machine ever enters the same node of the tree at the same state more than once, there will be an infinite loop and it cannot possibly have a valid run on this tree. Also the set D is finite since the k in down(k) is bounded by size of rules in G. So the total number of new labels is finite.

Our MCFG now has non-terminals of the form L_α, where L is the label of the node and α is the sequence containing the run information. The terminals are the output alphabet Δ. The arity of a non-terminal L_α is equal to the number of times the subtree rooted at L was visited, and the intuition is that each argument ultimately holds the word output by M during that visit. The rules of the MCFG are built as follows:

$$L_\alpha(s_1, \ldots, s_{k_0}) \leftarrow L^1_{\alpha_1}(x^1_1, \ldots, x^1_{k_1}) \cdots L^n_{\alpha_n}(x^n_1, \ldots, x^n_{k_n})$$

only if there is some derivation tree of G in which there is a node labelled L with children $L^1, \ldots, L^n$, and the labels $\alpha_1, \ldots, \alpha_k$ are compatible with α in the natural sense that they combine to give a valid run of M. The strings s_i are built also according to the runs: s_i is the word output by M in the i-th visit to this subtree, so if M outputs a terminal a, s_i will contain a, and whenever M visits the j-th child of L, then s_i contains the variable x_l^j (if this is the l-th time L_j is visited) which will eventually contain the word output by M on that visit of L_j.

We then create a distinct start non-terminal S and create rules to all to all labels of roots of derivation trees with valid runs like:

$$S(x_1 \cdots x_k) \leftarrow L_\alpha(x_1, \ldots, x_k)$$

What we have done is basically make the MCFG create derivation trees of G with additional run information to make sure that M has a valid accepting run on it. We use the power of MCFGs of keeping a tuple of strings at each non-terminal to keep track of M's output, for M's run could be more complicated than a simple top-down run.

5 Hunt for Machine Models

We were also interested in looking for machine models for recognizing MCFLs, mainly because if there was a "nice" model, then this may give us a machine characterization for bounded split-width MNWs, which is an important open question. We were told of some possible already existing machine models that can recognize (possibly more than) MCFLs: Thread Automata, Krivine Machines and Collapsible Pushdown Automata. Of these, we only managed to have time to investigate Thread Automata, the other two (see [9]) we did not have time to read about.

5.1 Thread Automata

Thread Automata were introduced by Villemonte de la Clergerie in 2002 [11] to describe a wide range of parsing strategies for many Mildly Context Sensitive Formalisms. However, we soon discovered that Thread Automata were Turing powerful and this put a stop to this line of enquiry.

6 Acknowledgements

I would like to thank Paul Gastin, Aiswarya Cyriac and K Narayan Kumar for giving me an opportunity to work with them on this topic. It was a great experience, and I learned a lot. I would also like to thank Sylvain Salvati for taking the time to explain the vast area of MCFGs and for his very useful advice, hints and references.

References

[1] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. pages 12–75, 1990.

[2] A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In M. Koutny and I. Ulidowski, editors, *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561, Newcastle, UK, Sept. 2012. Springer.

[3] F. Drewes, H.-J. Kreowski, and A. Habel. Handbook of graph grammars and computing by graph transformation. pages 95–162, River Edge, NJ, USA, 1997. World Scientific Publishing Co., Inc.

[4] J. Engelfriet and L. Heyker. The string generating power of context-free hypergraph grammars. pages 328–360, 1991.

[5] A. Habel. Hyperedge replacement: Grammars and languages. 1992.

[6] O. Rambow and G. Satta. Independent parallelism in finite copying parallel rewriting systems. pages 87–120, 1999.

[7] S. Salvati. MIX is a 2-MCFL and the word problem in $\mathbb{Z}^2$ is solved by a third-order collapsible pushdown automaton. Rapport de recherche, Feb. 2011.

[8] S. Salvati. Multiple context-free grammars. course 1: Motivations and formal denition. `http://www.labri.fr/perso/salvati/downloads/cours/esslli/course1.pdf`, 2011. ESSLLI 2011.

[9] S. Salvati and I. Walukiewicz. Recursive schemes, krivine machines, and collapsible pushdown automata. In *RP*, pages 6–20, 2012.

[10] H. Seki, T. Matsumura, M. Fujii, and T. Kasami. On multiple context-free grammars. pages 191–229, 1991.

[11] E. Villemonte de La Clergerie. Parsing mildly context-sensitive languages with thread automata. In *Proc. of COLING'02*, Aug. 2002.