System of Automata Recognizing Languages

Kelvin Cui Buño
Department of Computer Science
(Algorithms and Complexity)
Velasquez Ave., UP Diliman
Quezon City 1101
kcbuno@up.edu.ph

ABSTRACT

This paper looks into on the results of Čermák on the Multiaccepting Automata Systems[1, 2]. This automata system was inspired and is an automata counterpart of the Multigenerative Grammar Systems introduced by Meduna and Lukas [6]. The results show that the Multiaccepting Automata System and Multigenerative Grammar System is equivalent. In this paper, we show an alternate proof for showing equivalence between the two formal systems from the one given in [1].

1. INTRODUCTION

Grammar systems are a collection of formal grammars that generate a language in a distributed manner, and may even work in parallel. The individual grammar components are often context-free, but by adding some form of communicating mechanism, the grammar system is able to generate non-context-free languages. These grammar components can work on a shared string, in which case the string is rewritten by one grammar component and then passed on to another grammar component described by some protocol, as in the case of Cooperating Distributed Grammar Systems (CDGS) [5]. Grammar components can also work individually on their own strings, but these individual strings can also be passed on to other components for further processing, as in the case of Parallel Communicating Grammar Systems (PCGS) [9].

The concept of Multigenerative Grammar Systems (MGS) was introduced by Alexander Meduna and Roman Lukas [6]. For a given integer n, an n-generative grammar uses n context-free grammars of leftmost derivation. These n leftmost derivation are controlled by n-tuples of non-terminal symbols or rules. These generates n strings on which set or string operations can be performed. The *first* operation considers only the generated string of the grammar component indicated as the first component, that is, all other components are just accessories that aid in the restriction of the derivations. The *concatenation* operation concatenates all the output string into a single string, the order is determined by the index of the grammar components. And finally, the *union* operation which forms a set that includes the generated strings of all grammar components.

An automata system, similar to grammar systems, is a collection of automata components that accepts languages. Automata systems borrow computation and communication strategies from grammar systems to be able to accept languages that are difficult to be recognized by the individual automata components. Some studies on automata system include Multilanguage and Multiaccepting Automata System and its equivalence to Multigenerative Grammar System [1], Parallel Finite Automata System Communicating by Transitions [8], and Parallel Communicating Pushdown Automata Systems [4].

In this paper, we look into the hierarchy of n-languages shown in [2]. Specifically we look into the equivalence of the n-accepting move-restricted pushdown automata system and the n-generating rule-synchronized context-free grammar system. We present a construction of an equivalent pushdown automata system given the context-free grammar system.

The rest of the paper is organized as follows: Section 2 discusses the basic notations and defintions used in the paper, Section 3 discusses the Multigenerative Grammar System and the Multiaccepting Automata System, Section 4 gives a proof on the equivalence of the two formal systems, and finally Section 5 gives some final remarks about research on formal systems.

2. PRELIMINARIES

The reader is assumed to have background knowledge on formal languages and automata[11][10].

Let Σ be a finite set of symbols called an alphabet. We denote the Kleene closure of Σ, Σ^*, as the set of all finite words over the alphabet Σ including the empty string, ε. $|\Sigma|$ denotes the cardinality of the set Σ. Let w be a finite word. Then $|w|$ is the length of w. The empty string, ε, has length 0. If Σ is an alphabet and $a \in \Sigma$, and $w \in \Sigma^*$, then we denote the number of occurrences of a in w as $|w|_a$. If x, y are words and if $w = xy$, then w is the concatenation of x and y.

A context-free grammar, CFG G, is a four-tuple (N, T, P, S). N is the set of non-terminal symbols. T is the set of terminal symbols and is disjoint with N. S is the start non-terminal symbol and is an element of N. P is a finite set of production rules of the form $A \rightarrow x$, where $A \in N$, and $x \in (N \cup T)^*$. To label a production rule with a label r, we write a production rule as $r : A \to x$. Let $u, v \in (N \cup T)^*$. For every rule $A \to x \in P$, we write $uAv \Rightarrow uxv$. We denote the transitive-reflexive closure of $\Rightarrow$ as $\Rightarrow^*$. The language of G, L(G), is defined as $\{w \in T^* : S \Rightarrow^* w \in L(G)\}$.

A pushdown automaton, PDA M, is a seven-tuple (Q, Σ, Δ, δ, q_0, Z_0, F) where Q is a finite set of states, Σ is the input alphabet, Δ is the stack alphabet, δ is the transition function that maps $Q \times (\Sigma \cup \{\varepsilon\}) \times \Delta$ into finite subsets of $Q \times \Delta^*$, q_0 is the initial state, Z_0 is the initial stack symbol, and $F \subseteq Q$ is the set of final states. The transition function δ, given the current state, reads the input string and the symbol at the top of the stack. It then moves to the next state and writes the symbols into the top of the stack. We usually use the '\$' symbol as the initial stack symbol. Since there are only a finite number of input to the transition function, we use a label r for each input to output mapping of the transition function δ. We can write it as $r : (q_1, a, b) \to (q_2, c)$, where $q_1, q_2 \in Q, a \in \Sigma, b, c \in \Delta$. The PDA would accept the input string only if it reaches the final state with no symbol in the stack other than '\$'. The language recognized by a PDA M is denoted by $L(M)$.

The procedure for constructing a pushdown automaton from a given context-free grammar, or constructing a grammar from a given pushdown automaton can be found at [11], [10] and [**?**]. We give the following theorems to show the equivalence of PDA and CFG.

THEOREM 1. *[10][11] A language L is in the set of Context-Free Languages (CFL) if and only if there exists a CFG G such that L(G) = L.*

THEOREM 2. *[10][11] A language L is in CFL if and only if there exists a PDA M such that L(M) = L.*

We say that a recognizer (i.e automata) and a generator (i.e. grammars) are equivalent if and only if they accept the same language.

3. GRAMMAR AND AUTOMATA SYSTEMS

3.1 Multigenerative Grammar System

Multigenerative grammars are classified into two but equivalent grammar systems. They differ upon the production rule control tuple. The two controls are either non-terminal, which synchronizes all derivations of the grammars through their leftmost non-terminal, or by rule, which synchronizes all derivations of the grammars through the labeled production rules.

DEFINITION 1. *[6] An n-generative non-terminal-synchronized grammar system ($n - MGN$) is an $n + 1$ tuple,*

$$\Gamma = (G_1, ..., G_n, Q),$$

where $G_i = (N_i, T_i, P_i, S_i)$ is a context-free grammar for each $i = 1, ..., n$ and Q is a finite set of n-tuples of the form $(A_1, ..., A_n), A_i \in N_i, i = 1, ..., n$.

Let $\Gamma = (G_1, ..., G_n, Q)$ be an n-MGN. Then, a sentential n-form of n-MGN is an n-tuple of the form $\chi = (x_1, ..., x_n)$ where $x_i \in (N \cup T)^*$ for all $i = 1, ..., n$. Let $\chi = (\ u_1A_1v_1,$ ..., $u_nA_nv_n$) and $\bar{\chi} = (u_1x_1v_1, ..., u_nx_nv_n)$ be two sentential n-form, where $A_i \in N_i, u_i \in T_i^*$, and $v_i, x_i \in (N \cup T)^*$ for all $i = 1, ..., n$. Let $A_i \to x_i \in P_i$ for all $i = 1, ..., n$ and $(A_1, ..., A_n) \in Q$. Then $\chi \Rightarrow \bar{\chi}$. $\Rightarrow^*$ represents the reflexive-transitive closure of $\Rightarrow$.

Let $w = (w_1, ..., w_n)$ be an n-tuple, and each w_i, $i = 1, ..., n$ is a terminal string. We say that Γ generates w if and only if w is reachable from the initial sentential n-form of Γ through $\Rightarrow^*$.

The n-Language of Γ, n-L(Γ), is defined as

n-L(Γ) $= \{(w_1, ..., w_n) : (S_1, ..., S_n) \Rightarrow^* (w_1, ..., w_n), w_i \in T_i^*, i = 1, ..., n\}$

The language generated by Γ in the union mode, $L_{union}(\Gamma)$, is defined as

$$L_{union}(\Gamma) = \{w : (w_1, ..., w_n) \in \text{n-L}(\Gamma), \\ w \in \{w_i : i = 1, ..., n\}\}$$

The language generated by Γ in the concatenation mode, $L_{conc}(\Gamma)$, is defined as

$$L_{conc}(\Gamma) = \{w_1w_2...w_n : (w_1, ..., w_n) \in \text{n-L}(\Gamma)\}$$

The language generated by Γ in the first mode, $L_{first}(\Gamma)$, is defined as

$$L_{first}(\Gamma) = \{w_1 : (w_1, ..., w_n) \in \text{n-L}(\Gamma)\}$$

DEFINITION 2. *[6] An n-generative rule-synchronized grammar system ($n - MGR$) is an $n + 1$ tuple,*

$$\Gamma = (G_1, ..., G_n, Q),$$

where $G_i = (N_i, T_i, P_i, S_i)$ is a context-free grammar for each $i = 1, ..., n$ and Q is a finite set of n-tuples of the form $(p_1, ..., p_n), p_i \in P_i, \forall i = 1, ..., n$.

A sentential n-form for n-MGR is defined similarly as the sentential n-form in n-MGN. Let $\Gamma = (G_1, ..., G_n, Q)$ be a n-MGR. Let $\chi = (u_1A_1v_1, ..., u_nA_nv_n), \bar{\chi} = (u_1x_1v_1, ..., u_nx_nv_n)$ be two sentential n-forms, where $A_i \in N_i, u_i \in T_i^*, v_i, x_i \in (N \cup T)^*, \forall i = 1, ..., n$. Let $p_i : A_i \to x_i \in P_i$ and $(p_1, ..., p_n) \in Q$. Then $\chi \Rightarrow \bar{\chi}$ denotes the transition from configuration χ to $\bar{\chi}$. $\Rightarrow^*$ represents the reflexive-transitive closure of $\Rightarrow$.

Let $w = (w_1, ..., w_n)$ be an n-tuple, and each w_i, $i = 1, ..., n$ is a terminal string. We say that Γ generates w if and only if w is reachable from the initial sentential n-form of Γ through $\Rightarrow^*$.

The n-Language of Γ, n-L(Γ), is defined as

n-L(Γ) $= \{(w_1, ..., w_n) : (S_1, ..., S_n) \Rightarrow^* (w_1, ..., w_n), w_i \in T_i^*, i = 1, ..., n\}$

The language generated by Γ in the union mode, $L_{union}(\Gamma)$, is defined as

$$L_{union}(\Gamma) = \{w : (w_1, ..., w_n) \in \text{n-L}(\Gamma), \\ w \in \{w_i : i = 1, ..., n\}\}$$

The language generated by Γ in the concatenation mode, $L_{conc}(\Gamma)$, is defined as

$$L_{conc}(\Gamma) = \{w_1 w_2 ... w_n : (w_1, ..., w_n) \in \text{n-}L(\Gamma)\}$$

The language generated by Γ in the first mode, $L_{first}(\Gamma)$, is defined as

$$L_{first}(\Gamma) = \{w_1 : (w_1, ..., w_n) \in \text{n-}L(\Gamma)\}$$

The following is an example of a 2 component Multigenerative Grammar System that uses rule synchronization, a 2-MGR.

EXAMPLE 1. *We construct a 2-MGR,* $\Gamma = (G_1, G_2, Q)$.

G_1 = ({S_1}, {a, b}, {$r_{11} : S_1 \Rightarrow aS_1b$, $r_{12} : S_1 \Rightarrow \varepsilon$}, S_1)

G_2 = ({S_2}, {c, d}, {$r_{21} : S_2 \Rightarrow cS_2d$, $r_{22} : S_2 \Rightarrow \varepsilon$}, S_2)

Q = {(r_{11}, r_{21}), (r_{12}, r_{22})}

The initial sentential n-form for Γ is (S_1, S_2) as G_1 starts its computation with S_1 and G_2 starts its computation with S_2.

The rule-synchronization control tuple Q has two elements in it: (r_{11}, r_{21}) and (r_{12}, r_{22}). This means that when Γ performs a derivation, then G_1 and G_2 has to perform the production rules r_{11} and r_{21}, respectively, at the same time or r_{12} and r_{22} at the same time. It cannot be so that G_1 uses r_{11} but G_2 uses r_{22} since it was not defined in Q. For instance, Γ uses (r_{11}, r_{21}) twice, then the resulting sentential n-form is $(S_1, S_2) \Rightarrow (aS_1b, cS_2d) \Rightarrow (a^2S_1b^2, c^2S_2d^2)$.

To say the Γ generates an n-tuple, it is required that the each element of the n-tuple is only composed of terminal strings and is reachable from the initial sentential n-form. For instance, Γ finally applies the control (r_{12}, r_{22}) to have (S_1, S_2) $\Rightarrow^*$ (a^2b^2, c^2d^2).

2-L(Γ) = {(a^nb^n, c^nd^n): n $\geq$ 0},

$L_{first}(\Gamma)$ = {a^nb^n: n $\geq$ 0},

$L_{union}(\Gamma)$ = {a^nb^n: n $\geq$ 0} $\cup$ {c^nd^n: n $\geq$ 0},

$L_{conc}(\Gamma)$ = {$a^nb^nc^nd^n$: n $\geq$ 0}.

3.2 Multiaccepting Automata System

We know from classic literature that a context-free grammar, G, has an equivalent pushdown automaton, M. We say that they are equivalent because they have the same expressive power and both can recognize the set of Context-free languages.

Multiaccepting Automata System using PDA

DEFINITION 3. *[1, 2] A n-accepting move-restricted automata system (n-MAS) is an* $n + 1$ *tuple:*

$$\Lambda = (M_1, ..., M_n, P)$$

where each $M_i = (Q_i, \Sigma_i, \Delta_i, \delta_i, q_{0_i}, Z_{0_i}, F_i)$ *is a pushdown automaton and* P *is a finite set of n-tuples of the form* $(r_1, ..., r_n)$*, each* r_i *defined by* δ_i, $\forall i = 1, ..., n$.

The input to Λ *is a n-tuple of the form* $(x_1, ..., x_n), x_i \in \Sigma_i^*$. *The input to PDA* M_i *in* Λ *is* x_i.

Each PDA M_i can be defined by its current state, q_i, the remaining unread string, $a_i \in \Sigma_i^*$, and the stack content, $b_i \in \Delta_i^*$, where the first symbol of b_i is the top of the stack. We define the n-configuration of n-MAS as a n-tuple of the form $\chi = ((q_1, a_1, b_1), ..., (q_n, a_n, b_n))$, where, for $i = 1, ..., n$, $q_i \in Q_i$, $a_i \in \Sigma_i^*$, and $b_i \in \Delta_i^*$. We refer to each (q_i, a_i, b_i) as a configuration of M_i. If M_i has no more symbol to read, then $a_i = \varepsilon$.

Given an input tuple $w = (w_1, ..., w_n)$, for $i = 1, ..., n$, the initial configuration of M_i would be $(q_{0_i}, w_i, \varepsilon)$, where q_{0_i} is the start state, and $w_i \in \Sigma_i^*$. Then the initial n-configuration of Λ is $((q_{0_1}, w_1, \varepsilon), ..., (q_{0_n}, w_n, \varepsilon))$.

The accepting configuration of M_i would be $(q_{f_i}, \varepsilon, \varepsilon)$, where $q_{f_i} \in F_i$. An accepting n-configuration of Λ is $((q_{f_1}, \varepsilon, \varepsilon)$, ..., $(q_{f_n}, \varepsilon, \varepsilon))$.

Let $\chi = ((x_1, a_1, b_1)$, ..., $(x_n, a_n, b_n))$ and $\bar{\chi} = ((y_1, c_1, d_1)$, ..., $(y_n, c_n, d_n))$ be two n-configuration of Λ, where $x_i, y_i \in Q_i$, $a_i, c_i \in \Sigma_i^*$, and $b_i, d_i \in \Delta_i^*$. $a_i = \alpha c_i$, where $\alpha \in (\Sigma_i \cup \{\varepsilon\})$. $b_i = e\beta$ and $d_i = f\beta$, where $e, f \in (\Delta_i \cup \{\varepsilon\})$, $\beta \in \Delta_i^*$. Let $s_i : (x_i, \alpha, e) \vdash (y_i, f)$ be a transition in δ_i and $(s_1, ..., s_n) \in P$. Then χ directly transits to $\bar{\chi}$, denoted as $\chi \vdash \bar{\chi}$. We denote the reflexive-transitive closure of $\vdash$ as $\vdash^*$. We call the sequence of n-configuration of Λ obtained through the sequence of transitions as the computation of Λ.

Given an n-MAS Λ, and input n-tuple $w = (w_1, ..., w_n)$, we say that w is accepted by Λ if from the initial n-configuration of Λ, Λ reaches an accepting n-configuration of Λ. This would imply that each PDA component M_i of Λ reaches an accepting state and each PDA component has read all symbols of their respective input string from the input n-tuple.

There are two conditions for Λ to reject w. The first is, if after a halting computation of Λ, Λ does not reach an accepting n-configuration, then Λ rejects w. The second condition is, if at some point during the computation, no transition defined by the transition control P can be applied to obtain the next n-configuration of Λ and if the current n-configuration is not an accepting n-configuration of Λ, then Λ rejects w.

The n-language of Λ, n-L(Λ), is defined as:

n-L(Λ) = $\{(w_1, ..., w_n) : ((q_{0_1}, w_1, \varepsilon), ..., (q_{0_n}, w_n, \varepsilon)) \vdash^* ((q_{f_1}, \varepsilon, \varepsilon), ..., (q_{f_n}, \varepsilon, \varepsilon))\}$

The language recognized by Λ in the union mode, $L_{union}(\Lambda)$ is defined as

$$L_{union}(\Lambda) = \{w : (w_1, ..., w_n) \in \text{n-}L(\Lambda), \\ w \in \{w_i : i = 1, ..., n\}\}$$

The language recognized by Λ in the concatenation mode, $L_{conc}(\Lambda)$ is defined as

$$L_{conc}(\Lambda) = \{w_1 w_2 ... w_n : (w_1, ..., w_n) \in \text{n-}L(\Lambda)\}$$

The language recognized by Λ in the first mode, $L_{first}(\Lambda)$ is defined as

$$L_{first}(\Lambda) = \{w_1 : (w_1, ..., w_n) \in \text{n-}L(\Lambda)\}$$

The following is an example of MAS.

EXAMPLE 2. *We construct a 2-MAS, Λ = (M_1, M_2, P) as follows:*

$M_1 = (\{x_0, x_1, x_2\}, \{a, b\}, \{S_1, a, b, \$\}, \delta_1, x_0, \$, \{x_2\})$, δ_1 *is defined as follows:*

$\delta_1 : 10 : (x_0, \varepsilon, \varepsilon) \vdash (x_1, S_1\$), 11 : (x_1, \varepsilon, S_1) \vdash (x_1, aS_1b), 12 : (x_1, \varepsilon, S_1) \vdash (x_1, \varepsilon), 13 : (x_1, a, a) \vdash (x_1, \varepsilon), 14 : (x_1, b, b) \vdash (x_1, \varepsilon), 15 : (x_1, \varepsilon, \$) \vdash (x_2, \varepsilon)$

$M_2 = (\{y_0, y_1, y_2\}, \{c, d, \}, \{S_2, c, d, \$\}, \delta_2, y_0, \$, \{y_2\})$, δ_2 *is defined as follows:*

$\delta_2 : 20 : (y_0, \varepsilon, \varepsilon) \vdash (y_1, S_2\$), 21 : (y_1, \varepsilon, S_2) \vdash (y_1, cS_2d), 22 : (y_1, \varepsilon, S_2) \vdash (y_1, \varepsilon), 23 : (y_1, c, c) \vdash (y_1, \varepsilon), 24 : (y_1, d, d) \vdash (y_1, \varepsilon), 25 : (y_1, \varepsilon, \$) \vdash (y_2, \varepsilon)$

$P = \{(10, 20), (11, 21), (12, 22), (13, 23), (14, 24), (15, 25)\}$

Let $w = (a^k b^k, c^k d^k)$ be an input 2-tuple for Λ, for k a non-negative integer. M_1 would have input string $a^k b^k$ and M_2 would have $c^k d^k$

The initial 2-configuration of Λ is $((x_0, a^k b^k, \varepsilon), (y_0, c^k d^k, \varepsilon))$. From this configuration, M_1 and M_2 can only use transitions 10 and 20. The next 2-configuration of Λ is $((x_1, a^k b^k, S_1\$), (y_1, c^k d^k, S_2\$))$.

For the next $2k$ computation steps, Λ will alternate between using $(11, 21)$ and $(13, 23)$. Every time M_1 uses transition 11, the symbols aS_1b appears at the top of the stack, with a being the topmost symbol. There would always be only one instance of S_1 within the stack while b accumulates below S_1. The same goes for M_2.

After $2k$ steps, Λ can use $(12, 22)$ to remove S_1 and S_2 from the top of the stack of M_1 and M_2 respectively. The 2-configuration now is $((x_1, b^k, b^k\$), (y_1, d^k, d^k\$)$. Λ can then use (14, 24) to remove all terminal symbols from the stack and read all symbols of the input string.

Finally, Λ can use $(15, 25)$ to make M_1 and M_2 empty their stacks and go to their accepting states.

n-L(Λ) $= \{(a^n b^n, c^n d^n) : n \geq 0\}$

$L_{union}(\Lambda) = \{a^n b^n : n \geq 0\} \cup \{c^n d^n : n \geq 0\}$,

$L_{conc}(\Lambda) = \{a^n b^n c^n d^n : n \geq 0\}$, and

$L_{first}(\Lambda) = \{a^n b^n : n \geq 0\}$.

Observe that Λ has the is able to accept non-context free languages.

4. EQUIVALENCE OF MGS AND MAS

We argue that since a Multigenerative Grammar System (MGS) is composed of context-free grammar components. Then we could construct a Multiaccepting Automata System (MAS) composed of pushdown automata components. Since MGS has a language generated depending on mode $X \in$ {union, conc, first}, we would also find an equivalent language for MAS given the mode X.

An n-generative grammar is divided into two types, non-terminal synchronized and rule synchronized. But since a pushdown automata does not have any differentiation of a non-terminal or terminal symbol, the control would synchronize the transition function of each PDA in the n-accepting automata system.

THEOREM 3. *Let $L \subseteq \Sigma^*$. Then for every Γ, an n-MGR, there exists a Λ, an n-MAS, such that $L(\Gamma) = L(\Lambda)$.*

Proof:
We construct an n-MAS, Λ, from a given n-MGR, Γ as follows:

1.) For each grammar component $G_i = (N_i, T_i, S_i, P_i)$ of Γ, for $i = 1, ..., n$, construct its equivalent PDA and label it $M_i = (Q_i, \Sigma_i, \Delta_i, \delta_i, q_{i0}, Z_i, F_i)$. The set of states of M_i, Q_i, would consist of the initial state q_{i0}, the loop state $q_{i_{loop}}$, the accept states $F_i = \{q_{if}\}$, and additional states to simulate each production rule in G_i. The input alphabet $\Sigma_i = T_i$, and the stack alphabet, $\Delta_i = (N_i \cup T_i \cup \{\$\})$.

Since Γ uses leftmost derivation context-free grammar components, for a production rule $r : A \to x$, $x = x_1 x_2 ... x_k$, for a positive integer k, the first symbol to be placed in the stack is x_k. Then followed by x_{k-1} and so on, so that x_1 is now at the top of the stack.

For each rule $r : A \to x \in P_i$, for $x = x_1 x_2 ... x_k$, there would be k transitions associated with r. We label them as follows: $s_1 : (q_{i_{loop}}, \varepsilon, A) \vdash (q_{s_1}, x_k)$, $s_2 : (q_{s_1}, \varepsilon, \varepsilon) \vdash (q_{s_2}, x_{k-1})$, ..., $s_{k-1} : (q_{s_{k-2}}, \varepsilon, \varepsilon) \vdash (q_{s_{k-1}}, x_2)$, $s_k : (q_{s_{k-1}}, \varepsilon, \varepsilon) \vdash (q_{loop}, x_1)$. These k transitions are added to δ_i. We add the following states to Q_i: $\{q_{s_1}, ..., q_{s_{k-1}}\}$.

The k transitions are denoted by $s : (q_{i_{loop}}, \varepsilon, A) \vdash (q_{i_{loop}}, x)$ for brevity and s is referred to as a transition sequence. We say that rule r is equivalent to transistion sequence s. $s \in \delta_i$ if $s_1, ..., s_k \in \delta_i$.

The transition $t : (q_{i_{loop}}, a, a) \vdash (q_{i_{loop}}, \varepsilon)$ is also added to δ_i for all $a \in T_i$. In addition, a special rule is added, an ε-

transtion, $\varepsilon_i : (q_{i_{loop}}, \varepsilon, \varepsilon) \vdash (q_{i_{loop}}, \varepsilon)$. Finally, let the initial stack Z_i be empty.

2.) For each tuple in the production rule control, Q of Γ, of the form $(r_1, ..., r_n)$, each $r_i \in P_i$, add $(s_1, ..., s_n)$ to the transition control of Λ, P, for each $s_i \in \delta_i$, and s_i is the equivalent transition sequence of the production rule r_i.

For a given transition control tuple $(s_1, ..., s_n)$, notice that the number of transitions in each sequence, s_i, $i = 1, ..., n$ may not be equal. A transition sequence s_i can be represented as $s_{i1}, s_{i2}, ..., s_{ik_i}$ in order, where $k_i \geq 1$ is the number of transitions in s_i. Let max be the most number of transitions in any transition sequence in the elements of $(s_1, s_2, ..., s_n)$. Then $(s_1, s_2, ..., s_n)$ represents the following set of transition control tuples: $\{(s_{11}, s_{21}, ..., s_{n1})$, $(s_{12}, s_{22}, ..., s_{n2})$, ..., $(s_{1max}, s_{2max}, ..., s_{nmax})\}$.

For all $i = 1, ..., n$, $j = 1...max$, if $1 \leq j \leq k_i$, then s_{ij} is the jth transition of sequence s_i. Else, $k_i < j \leq max$, s_{ij} is an ε-transition, ε_i.

Note that $(s_{1y}, s_{2y}, ..., s_{ny})$ cannot be used by Λ before $(s_{1x}, s_{2x}, ..., s_{nx})$, for $1 \leq x < y \leq max$. As stated in step (1), each transition sequence, s, is composed of at least one transition in δ_i. The transition sequence s can only be completed if the set of transitions are applied in proper order. Therefore as a consequence, the transition control tuples can only be applied in proper order.

The notation, $(s_1, s_2, ..., s_n)$, can then be used to refer to these set of transition control tuples in P without ambiguity.

3.) Add the two n-tuples $(q_{1_{ol}}, ..., q_{n_{ol}})$ and $(q_{1_{lf}}, ..., q_{n_{lf}})$, where for all $i = 1, ..., n$ $q_{i_{0l}} : (q_{i_0}, \varepsilon, \varepsilon) \vdash (q_{i_{loop}}, S_i\$)$ and $q_{i_{lf}} : (q_{i_{loop}}, \varepsilon, \$) \vdash (q_{i_f}, \varepsilon)$

4a.) Add an n-tuple $(t_1, \varepsilon_2, ..., \varepsilon_n)$ to P for all $t_1 \in \delta_1$. Add $(\varepsilon_1, t_2, ..., \varepsilon_n)$ to P for all $t_2 \in \delta_2$. Add an n-tuple $(\varepsilon_1, ..., t_i, ..., \varepsilon_n)$ for all $t_i \in \delta_i$, for all $i = 1, ..., n$, for $t_i : (q_{i_{loop}}, a, a) \vdash (q_{i_{loop}}, \varepsilon)$, $a \in T_i$.

Another way is to have all possible permutations of which components have terminal symbols on the top of the stack at the same time. This is so that Λ can perform in parallel the popping procedure for terminal symbols. (4a) is not used at the same time with (4b) when constructing Λ.

4b.) Add all possible n-tuple $(\bar{t_1}, \bar{t_2}, ..., \bar{t_n})$ to P such that for all $i = 1, ..., n$, a transition $\bar{t_i}$ is either ε_i or $t_i : (q_{i_{loop}}, a, a) \vdash (q_{i_{loop}}, \varepsilon)$, $a \in T_i$, $\varepsilon_i, t_i \in \delta_i$.

The purpose of steps (4a) and (4b) is to make Λ prioritize the removal of terminal symbols from the top of the stack. This is so that Λ faithfully follows the derivation steps of Γ.

Given an input tuple $w = (w_1, ..., w_n)$, the initial n-configuration of Λ is of the form $((q_{0_1}, w_1, \varepsilon), ... ,(q_{0_n}, w_n, \varepsilon))$. The accepting n-configuration is $((q_{f_1}, \varepsilon, \varepsilon), ... ,(q_{f_n}, \varepsilon, \varepsilon))$.

We have successfully constructed a Λ given a Γ. We now prove that Λ correctly simulates the derivation of Γ.

We prove the claim:

CLAIM 1. *$(S_1, ..., S_n) \Rightarrow^* (\alpha_{11}\alpha_{12}, ..., \alpha_{n1}\alpha_{n2})$, where for $i = 1, ..., n$, $\alpha_{i1} \in T_i^*$ and $\alpha_{i2} \in N_i(N_i \cup T_i)^* \cup \{\varepsilon\}$, if and only if $((q_{1_{loop}}, \alpha_{11}, S_1\$), ..., (q_{n_{loop}}, \alpha_{n1}, S_n\$)) \vdash^* ((q_{1_{loop}}, \varepsilon, \alpha_{12}\$), ..., (q_{n_{loop}}, \varepsilon, \alpha_{n2}\$))$.*

Once the claim is proven, it follows that $(S_1, S_2, ..., S_n) \Rightarrow^* (\alpha_1, ..., \alpha_n)$, where for $i = 1, ..., n$, $\alpha_i \in T_i^* = \Sigma_i^*$, if and only if $((q_{1_{loop}}, \alpha_1, S_1\$), ..., (q_{n_{loop}}, \alpha_n, S_n\$)) \vdash^* ((q_{1_{loop}}, \varepsilon, \$), ..., (q_{n_{loop}}, \varepsilon, \$))$. When Λ reaches the n-configuration $((q_{1_{loop}}, \varepsilon, \$), ..., (q_{n_{loop}}, \varepsilon, \$))$, it will only take one more direct transition to reach its accepting n-configuration $((q_{f_1}, \varepsilon, \varepsilon), ..., (q_{f_n}, \varepsilon, \varepsilon))$ since no other transitions can be made at this point of the computation. This would mean that for $\alpha = (\alpha_1, ..., \alpha_n)$, $\alpha \in$ n-L(Γ) if and only if $\alpha \in$ n-L(Λ).

Proof of Claim 1:

The claim is an if and only if statement and therefore the proof will have two parts.

Suppose that $(S_1, S_2, ..., S_n) \Rightarrow^* (\alpha_1, ..., \alpha_n)$, where for $i = 1, ..., n$, $\alpha_i = \alpha_{i1}\alpha_{i2}$, $\alpha_{i1} \in T_i^*$ and $\alpha_{i2} \in N_i(N_i \cup T_i)^* \cup \{\varepsilon\}$. The proof is by induction on the length of the computation of $(\alpha_1, ..., \alpha_n)$ from $(S_1, S_2, ..., S_n)$ by Γ.

Basis Step. If the derivation length is 0, then $(S_1, S_2, ..., S_n) = (\alpha_1, ..., \alpha_n) = (\alpha_{12}, ..., \alpha_{n2})$, since for $i = 1, ..., n$, $\alpha_{i1} = \varepsilon$; then, $((q_{1_{loop}}, \alpha_{11}, S_1\$), ..., (q_{n_{loop}}, \alpha_{n1}, S_n\$)) \vdash^* ((q_{1_{loop}}, \varepsilon, \alpha_{12}\$), ..., (q_{n_{loop}}, \varepsilon, \alpha_{n2}\$))$

Induction Hypothesis. Assume that if $(S_1, S_2, ..., S_n) \Rightarrow^* (\alpha_{11}\alpha_{12}, ..., \alpha_{n1}\alpha_{n2})$ by a derivation of length k or less, $k \geq 0$, then $((q_{1_{loop}}, \alpha_{11}, S\$), ..., (q_{n_{loop}}, \alpha_{n1}, S\$)) \vdash^* ((q_{1_{loop}}, \varepsilon, \alpha_{12}\$), ..., (q_{n_{loop}}, \varepsilon, \alpha_{n2}\$))$.

Induction Step. Let

$$(S_1, S_2, ..., S_n) = h_0 \Rightarrow h_1 \Rightarrow ... \Rightarrow h_{k+1} = (\alpha_1, ..., \alpha_n)$$

be a leftmost derivation of $(\alpha_1, ..., \alpha_n)$ from $(S_1, S_2, ..., S_n)$, and for $i = 1, ..., n$, let $\alpha_i = \alpha_{i1}\alpha_{i2}$. h_n has at least one nonterminal, so $h_k = (u_1A_1v_1, ..., u_nA_nv_n)$ and $h_{k+1} = (u_1x_1v_1, ..., u_nx_nv_n)$, where for $i = 1, ..., n$, $u_i \in T_i^*$, $A_i \in N$, $v_i \in (N_i \cup T_i)^*$, $r_i : A_i \to x_i$, and $(r_1, ..., r_n) \in Q$. By the induction hypothesis

$$((q_{1_{loop}}, u_1, S_1\$), ..., (q_{n_{loop}}, u_n, S_n\$)) \vdash^* ((q_{1_{loop}}, \varepsilon, A_1v_1\$), ..., (q_{n_{loop}}, \varepsilon, A_nv_n\$))$$

and since for $i = 1, ..., n$, $r_i : A_i \to x_i$, and $(r_1, ..., r_n) \in Q$, $s_i : (q_{i_{loop}}, \varepsilon, A_i) \vdash (q_{i_{loop}}, x_i)$ is a transition sequence of M_i and $(s_1, ..., s_n) \in P$. And whenever a terminal symbols appears at the top of the stack of at least one component, a transition control tuple from step (4) of the construction is used by Λ to remove the terminal symbol at the top of the stack from all components. So,

$$((q_{1_{loop}}, \varepsilon, A_1 v_1\$), ..., (q_{n_{loop}}, \varepsilon, A_n v_n\$)) \vdash$$
$$((q_{1_{loop}}, \varepsilon, x_1 v_1\$), ..., (q_{n_{loop}}, \varepsilon, x_n v_n\$)).$$

Now $(\alpha_1, ..., \alpha_n) = (u_1 x_1 v_1, ..., u_n x_n v_n) = (\alpha_{11}\alpha_{12}, ..., \alpha_{n1}\alpha_{n2})$. For $i = 1, ..., n$, $\alpha_{i1} \in T_i^*$ and α_{i2} is ε or begins with a non-terminal symbol. Hence $|\alpha_{i1}| \geq |u_i|$ and $|\alpha_{i2}| \leq |x_i v_i|$ and α_{i1} can be rewritten as $u_i u_i'$ for some $u_i' \in T_i^*$ such that $u_i'\alpha_{i2} = x_i v_i$. Therefore

$$((q_{1_{loop}}, u_1', x_1 v_1\$), ..., (q_{n_{loop}}, u_n', x_n v_n\$)) \vdash^*$$
$$((q_{1_{loop}}, \varepsilon, \alpha_{12}\$), ..., (q_{n_{loop}}, \varepsilon, \alpha_{n2}\$))$$

by using transitions defined by the transition control tuples in step (4). Finally, to complete the induction,

$$((q_{1_{loop}}, \alpha_{11}, S_1\$), ..., (q_{n_{loop}}, \alpha_{n1}, S_n\$)) =$$
$$((q_{1_{loop}}, u_1 u_1', S_1\$), ..., (q_{n_{loop}}, u_1 u_1', S_n\$)) \vdash^*$$
$$((q_{1_{loop}}, u_1', A_1 v_1\$), ..., (q_{n_{loop}}, u_1', A_n v_n\$)) \vdash$$
$$((q_{1_{loop}}, u_1', x_1 v_1\$), ..., (q_{n_{loop}}, u_1', x_n v_n\$)) \vdash^*$$
$$((q_{1_{loop}}, \varepsilon, \alpha_{12}\$), ..., (q_{n_{loop}}, \varepsilon, \alpha_{n2}\$))$$

and this completes the first part of the proof for Claim 1. The next induction is for the second part of the proof.

Now suppose that $((q_{1_{loop}}, \alpha_{11}, S_1\$)$, ..., $(q_{n_{loop}}, \alpha_{n1}, S_n\$))$ $\vdash^*$ $((q_{1_{loop}}, \varepsilon, \alpha_{12}\$), ..., (q_{n_{loop}}, \varepsilon, \alpha_{n2}\$))$, where for $i = 1, ..., n$, $\alpha_{i1} \in \Sigma_i^*$ and $\alpha_{i2} \in (N_i \cup T_i)^*$; we show that $(S_1, ..., S_n) \Rightarrow^*$ $(\alpha_{11}\alpha_{12}, ..., \alpha_{n1}\alpha_{n2})$. The proof is by induction on the length of the computation of Λ.

Basis Step. If $((q_{1_{loop}}, \alpha_{11}, S_1\$)$, ..., $(q_{n_{loop}}, \alpha_{n1}, S_n\$))$ $\vdash^*$ $((q_{1_{loop}}, \varepsilon, \alpha_{12}\$)$, ..., $(q_{n_{loop}}, \varepsilon, \alpha_{n2}\$))$ in zero steps, $((q_{1_{loop}}$, α_{11}, $S_1\$)$, ..., $(q_{n_{loop}}$, α_{n1}, $S_n\$)) = ((q_{1_{loop}}$, ε, $\alpha_{12}\$)$, ..., $(q_{n_{loop}}$, ε, $\alpha_{n2}\$))$, then for $i = 1, ..., n$, $\alpha_{i1} = \varepsilon$, $\alpha_{i2} = S_i$, and thus $(S_1, ..., S_n) \Rightarrow^*$ $(\alpha_{11}\alpha_{12}, ..., \alpha_{n1}\alpha_{n2})$.

Induction Hypothesis. If $((q_{1_{loop}}, \alpha_{11}, S_1\$)$, ..., $(q_{n_{loop}}$, α_{n1}, $S_n\$))$ $\vdash^*$ $((q_{1_{loop}}, \varepsilon, \alpha_{12}\$)$, ..., $(q_{n_{loop}}, \varepsilon, \alpha_{n2}\$))$ by a computation of k steps or fewer, $k \geq 0$, then $(S_1, ..., S_n) \Rightarrow^*$ $(\alpha_{11}\alpha_{12}$, ..., $\alpha_{n1}\alpha_{n2})$.

Induction Step. Suppose that if $((q_{1_{loop}}, \alpha_{11}, S_1\$)$, ..., $(q_{n_{loop}}$, α_{n1}, $S_n\$))$ $\vdash^*$ $((q_{1_{loop}}, \varepsilon, \alpha_{12}\$)$, ..., $(q_{n_{loop}}, \varepsilon, \alpha_{n2}\$))$ in $k+1$ steps. Then for $i = 1, ..., n$, there exists $\beta_i \in \Sigma_i^*$, $\gamma_i \in (N_i \cup T_i)^*$, $((q_{1_{loop}}, \alpha_{11}, S_1\$)$, ..., $(q_{n_{loop}}, \alpha_{n1}, S_n\$))$ $\vdash^*$ $((q_{1_{loop}}$, β_1, $\gamma_1\$)$, ..., $(q_{n_{loop}}$, β_n, $\gamma_n\$))$ in k steps, and $((q_{1_{loop}}$, β_1, $\gamma_1\$)$, ..., $(q_{n_{loop}}$, β_n, $\gamma_n\$))$ $\vdash$ $((q_{1_{loop}}$, ε, $\alpha_{12}\$)$, ..., $(q_{n_{loop}}$, ε, $\alpha_{n2}\$))$. This last n-configuration transition is the result of a transition control tuple in step (2) or a set of transition control tuples in step (4).

If the last transition was a result of using a transition control tuple in step (2), then for $i = 1, ..., n$, $\beta_i = \varepsilon$, $\gamma_i = A_i v_i$, and $\alpha_{i2} = x_i v_i$, for some $A_i \in N_i$, $x_i \in (N_i \cup T_i)^*$, r_i : $A \rightarrow x$, and $(r_1, ..., r_n) \in Q$. Since $(S_1, ..., S_n) \Rightarrow^*$ $(\alpha_{11}A_1 v_1$, ..., $\alpha_{n1}A_n v_n)$ by the induction hypothesis, $(S_1, ..., S_n) \Rightarrow^*$ $(\alpha_{11}x_1 v_1, ..., \alpha_{n1}x_n v_n) = (\alpha_{11}\alpha 12, ..., \alpha_{n1}\alpha n2)$.

If the last transition was a result of using a set of transition control tuple in step (4), then for $i = 1, ..., n$, $\beta_i = a_i$, $a_i \in T_i$ and $\gamma = a_i\alpha_{i2}$. Then $\alpha_{i1} = u_i a_i$, for some $u_i \in T_i^*$, and $((q_{1_{loop}}, u_1, S_1\$)$, ..., $(q_{n_{loop}}, u_n, S_n\$))$ $\vdash^*$ $((q_{1_{loop}}, \varepsilon, a_1\alpha_{12}\$)$, ..., $(q_{n_{loop}}, \varepsilon, a_n\alpha_{n2}\$))$ within k steps, so by the induction hypothesis $(S_1, ..., S_n) \Rightarrow^*$ $(u_1 a_1\alpha_{12}$, ..., $u_n a_n \alpha_{n2}) = (\alpha_{11}\alpha_{12}$, ..., $\alpha_{n1}\alpha_{n2})$.

This completes the proof for Claim 1.

Claim 1 implies that for an n-tuple string w, $w \in n$-L(Γ) if and only if $w \in n$-L(Λ).

Given an input n-tuple $w = (w_1, ..., w_n)$ and $w \in n$-L(Γ), from the initial n-configuration of Λ, $((q_{0_1}$, w_1, $\varepsilon)$, ..., $(q_{0_n}$, w_n, $\varepsilon))$, Λ uses the transition control tuple, $(q_{1_{ol}}, ..., q_{n_{ol}})$, to push the start non-terminals $S_1, S_2, ..., S_n$ and the bottom of the stack marker, \$ to their respective PDA components' stacks. Now the current n-configuration is similar to the basis of Claim 1, $((q_{1_{loop}}$, α_1, $S_1\$)$, ..., $(q_{n_{loop}}$, α_n, $S_n\$))$, where for $i = 1, ..., n$, $\alpha_i = w_i$. By Claim 1, $((q_{1_{loop}}$, α_1, $S_1\$)$, ..., $(q_{n_{loop}}$, α_n, $S_n\$))$ $\vdash^*$ $((q_{1_{loop}}$, ε, $\$)$, ..., $(q_{n_{loop}}$, ε, $\$))$ since $w \in$ n-L(Γ). When Λ can only use the transition control tuple $(q_{1_{lf}}, ..., q_{n_{lf}})$, then $((q_{1_{loop}}$, ε, $\$)$, ..., $(q_{n_{loop}}$, ε, $\$))$ $\vdash$ $((q_{f_1}, \varepsilon, \varepsilon)$, ..., $(q_{f_n}, \varepsilon, \varepsilon))$, which is the accepting n-configuration of Λ and therefore $w \in$ n-L(Λ).

If $w \notin n$-L(Γ), from n-configuration $((q_{1_{loop}}$, α_1, $S_1\$)$, ..., $(q_{n_{loop}}$, α_n, $S_n\$))$ of Λ, by Claim 1, there does not exist a computation such that $((q_{1_{loop}}$, α_1, $S_1\$)$, ..., $(q_{n_{loop}}$, α_n, $S_n\$))$ $\vdash^*$ $((q_{1_{loop}}$, ε, $\$)$, ..., $(q_{n_{loop}}$, ε, $\$))$. Any computation of Λ must pass through the n-configuration $((q_{1_{loop}}$, ε, $\$)$, ..., $(q_{n_{loop}}$, ε, $\$))$ since we require that all symbols of the input strings of the components must be read and that their respective stacks be empty with only the bottom of the stack marker, \$, is left. Λ can only reach its accepting n-configuration from this n-configuration. Therefore, Λ cannot reach the accepting n-configuration and thus $w \notin$ n-L(Λ).

And finally, by Claim 1 and the definition of the languages under an operation of Λ, as a consequence the following is true and is trivial to prove:

- $L_{first}(\Gamma) = L_{first}(\Lambda)$
- $L_{union}(\Gamma) = L_{union}(\Lambda)$
- $L_{conc}(\Gamma) = L_{conc}(\Lambda)$

With these, we can say that $L(\Gamma) = L(\Lambda)$. This ends the proof for Theorem 3.

We illustrate the construction of an n-MAS from an n-MGS through the next example:

EXAMPLE 3. *Suppose we have an n-MGR, $\Gamma = (G_1, G_2, Q)$, with G_1, G_2 and Q defined as follows:*

$G_1 = (N_1 = \{S_1\}, T_1 = \{a, b\}, S_1, P_1 = \{r_{11} : S_1 \Rightarrow aS_1b$, $r_{12} : S_1 \Rightarrow \varepsilon\}$

$G_2 = (N_2 = \{S_2\}, T_2 = \{c, d\}, S_2, P_2 = \{r_{21} : S_2 \Rightarrow cS_2d$, $r_{22} : S_2 \Rightarrow \varepsilon\}$

$Q = \{(r_{11}, r_{21}), (r_{12}, r_{22})\}$

We construct the equivalent n-MAS, Λ, of the n-MGS, Γ using the four steps of the proof of Theorem 3.

Step 1:

The n-MAS $\Lambda = (M_1, M_2, P)$ will be constructed as follows:

$$M_1 = (Q_1, \Sigma_1, \Delta_1, \delta_1, q_{start}, F_1, Z_1)$$

where:

- $Q_1 = \{q_{10}, q_{1loop}, q_{1f}\} \cup \{q_{t11,s1}, q_{t11,s2}\}$
- $\Sigma_1 = \{a, b\}$
- $\Delta_1 = \{S_1, a, b, \$\}$
- $q_{start} = q_{10}$
- $F_1 = \{q_{1f}\}$
- $Z_1 = \emptyset$
- $\delta_1 =$
  - $\{t_{11,s1} : (q_{1loop}, \varepsilon, S_1) \vdash (q_{t11,s1}, b),$
    $t_{11,s2} : (q_{t11,s1}, \varepsilon, \varepsilon) \vdash (q_{t11,s2}, S_1),$
    $t_{11,s3} : (q_{t11,s2}, \varepsilon, \varepsilon) \vdash (q_{1loop}, a)\} \cup$
  - $\{t_{12,s1} : (q_{1loop}, \varepsilon, S_1) \vdash (q_{1loop}, \varepsilon, \varepsilon)\} \cup$
  - $\{t_a : (q_{1loop}, a, a) \vdash (q_{1loop}, \varepsilon),$
    $t_b : (q_{1loop}, b, b) \vdash (q_{1loop}, \varepsilon),$
    $\varepsilon_1 : (q_{1loop}, \varepsilon, \varepsilon) \vdash (q_{1loop}, \varepsilon)\} \cup$
  - $\{q_{1_{0l}} : (q_{10}, \varepsilon, \varepsilon) \vdash (q_{1loop}, \$),$
    $q_{1_{lf}} : (q_{1loop}, \varepsilon, \$) \vdash (q_{1f}, \varepsilon)\}$

$$M_2 = (Q_2, \Sigma_2, \Delta_2, \delta_2, q_{start}, F_2, Z_2)$$

where:

- $Q_2 = \{q_{20}, q_{2loop}, q_{2f}\} \cup \{q_{t21,s1}, q_{t21,s2}\}$
- $\Sigma_2 = \{c, d\}$
- $\Delta_2 = \{S_2, c, d, \$\}$
- $q_{start} = q_{20}$
- $F_2 = \{q_{2f}\}$
- $Z_2 = \emptyset$
- $\delta_2 =$
  - $\{t_{21,s1} : (q_{2loop}, \varepsilon, S_2) \vdash (q_{t21,s2}, d),$
    $t_{21,s2} : (q_{t21,s1}, \varepsilon, \varepsilon) \vdash (q_{t21,s2}, S_2),$
    $t_{21,s3} : (q_{t21,s2}, \varepsilon, \varepsilon) \vdash (q_{2loop}, c)\} \cup$
  - $\{t_{22,s1} : (q_{2loop}, \varepsilon, S_2) \vdash (q_{2loop}, \varepsilon, \varepsilon)\} \cup$
  - $\{t_c : (q_{2loop}, c, c) \vdash (q_{2loop}, \varepsilon),$
    $t_d : (q_{2loop}, d, d) \vdash (q_{2loop}, \varepsilon),$
    $\varepsilon_2 : (q_{2loop}, \varepsilon, \varepsilon) \vdash (q_{2loop}, \varepsilon)\}$
  - $\{q_{2_{0l}} : (q_{20}, \varepsilon, \varepsilon) \vdash (q_{2loop}, \$),$
    $q_{2_{lf}} : (q_{2loop}, \varepsilon, \$) \vdash (q_{2f}, \varepsilon)\}$

Step 2: Define the set of control n-tuple P_{step2}:

$P_{step2} =$

- $\{(t_{11,s1}, t_{21,s1}), (t_{11,s2}, t_{21,s2}), (t_{11,s3}, t_{21,s3})\} \cup$
- $\{(t_{12,s1}, t_{22,s1})\}$

Step 3: Define the set of control n-tuple P_{step3}:

$P_{step3} =$

- $\{(q_{1_{0l}}, q_{2_{0l}}), (q_{1_{lf}}, q_{2_{lf}})\}$

Step 4: Define the set of control n-tuple P_{step4}:

$P_{step4} =$

- $\{(t_a, \varepsilon_2), (t_b, \varepsilon_2)\} \cup$
- $\{(\varepsilon_1, t_c), (\varepsilon_1, t_d)\}$

Combining Step 2 to Step 4, we have

$$P = P_{step2} \cup P_{step3} \cup P_{step4}$$

5. FINAL REMARKS

We have presented a construction method for building an MAS, given an MGS, showing that the MGS can be simulated by an MAS. This construction method is intuitive, as it is based on the classic automata theory literature of constructing an equivalent PDA from a given CFG. The downside of the presented construction method is that the MAS may have components that become idle, indicated when they use the ε transitions. The ε transitions are more prominent when the pushdown automata components produce non-terminal symbols. This is certain to result in increased time complexity of the automata system for accepting a given language.

One of the future works that can be looked at is the construction of a MGS given the MAS. Designing the construction method is difficult, as the process is more complex, if the construction is based on literature of constructing a CFG from a PDA.

Results of [6] and [7] shows that the computing power of MGS is equivalent to that of a Matrix Grammar, that is the set of Context-Sensitive Languages. This shows that MAS can also accept context-sensitive languages.

As the both MAS and MGS work as parallel computing devices, for future works, we can look into how to use MAS and MGS for solving computationally hard problems. Similar to the works done in [3], we can add some features to the existing formal systems to be able to solve hard problems in lower time complexity, with the trade-off of higher space complexity.

6. REFERENCES

[1] M. Čermák.; Multilanguages and Multiaccepting Automata System.; In: Proceedings of the 16th Conference and Competition STUDENT EEICT 2010 Volume 5, Brno, CZ, FIT VUT, 2010, p. 146-150.

[2] M. Čermák.; Formal Systems Based Upon Automata and Grammars.; Information Sciences and Technologies Bulletin of the ACM Slovakia, Volume 4, Number 4 (2012) p. 7-14.

[3] Csuhaj-Varjú, Erzsébet. Vaszil, György. Păun, Gheorghe.; Grammar System versus Membrane Computing: The Case of CD Grammar Systems. Fundamenta Informaticae 76(3) (2007), 271-292.

[4] E. Csuhaj-Varjú, C. Martin-Vide, V. Mitrana, G. Vaszil; Parallel Communicating Pushdown Automata System. Int. J. Found. Comput. Sci. 11(4): 633-650 (2000)

[5] E. Csuhaj-Varjú. T. Masopust. G. Vaszil.; Cooperating Distributed Grammar System with Permitting Grammars as Components.; Romanian Journal of Information Science and Technology, Volume 12, Number 2, 2009, 175-189.

[6] Meduna, Alexander. Lukas, Roman.; Multigenerative Grammar Systems.; Contents & Abstracts, Schedae Informaticae, Issue 15 (2006) Multigenerative Grammar Systems, pp. 175-187.

[7] Meduna, Alexander. Lukas, Roman.; Multigenerative Grammar Systems and Matrix Grammars.; Kybernetika - Vol 46 (2010), Number 1, Pages 68-82.

[8] Petrik, Patřík.; Parallel Finite Automata Systems Communicating by Transistions.; In: Proceedings of the 16th Conference and Competition STUDENT EEICT 2010 Volume 5, Brno, CZ, FIT VUT, 2010.

[9] G. Păun, L. Sântean; Parallel communicating grammar systems: The regular case. Annals of University of Bucharest, Ser. Matematica-Informatica (1989).

[10] Hopcroft, John E.. Ullman, Jeffrey D.. Introduction to automata theory, languages, and computation. Addison-Wesley Publishing Company, 1979.

[11] Sipser, Michael. 1997. Introduction to the Theory of Computation.