International Journal of Foundations of Computer Science
© World Scientific Publishing Company

DISTRIBUTED PROCESSING IN AUTOMATA[*]

KAMALA KRITHIVASAN

M. SAKTHI BALAN

and

PRAHLADH HARSHA

Department of Computer Science and Engineering, Indian Institute of Technology, Madras, Chennai - 600036, India, E-mail : kamala@iitm.ernet.in

ABSTRACT

With distributed computing beginning to play a major role in modern Computer Science, the theory of grammar systems and distributed automata has been developed in order to model distributed computing. In this paper, we introduce the notion of distributed automata in the sequential sense. Distributed Automata are a group of automata working in unison to accept one language. We build the theory of distributed automata for FSA and PDA in different modes of acceptance like the t-mode, *-mode, $= k$-mode, $\leq k$-mode and $\geq k$-mode. We then analyze the acceptance power of each automata in all the above modes. We present proofs that distributed FSA do not have any additional power over "centralized" FSAs in any of the modes, while distributed PDA with only two components is as powerful as Turing Machines in all of the modes. We give proofs for the equivalence of all modes in the case of PDAs. We also study a restricted version of distributed PDA called k-turn distributed PDA.

Keywords: Grammar systems, distributed computing, distributed automata, modes of acceptance.

1. Introduction

In classic Formal Language and Automata theory, grammars and automata model classic computing devices. These devices are "centralized" - the computation is accomplished by one "central" agent. Hence in classic Formal Language Theory, a language is generated by one grammar or accepted by one automaton. In modern Computer Science, distributed computation plays a major role. Analyzing such computations in computer networks, distributed databases, etc., leads to notions such as distribution parallelism, concurrency and communication. The theory of grammar systems is a grammatical model for distributed computation where these notions could be defined and analyzed.

[*]This research work was partially supported by the project from Department of Science and Technology

A grammar system is a set of grammars working in unison, according to a specified protocol, to generate one language. There are many reasons to consider such a generative mechanism: to model distribution, to increase the generative power, to decrease the complexity, etc. These distributed grammar systems can be either sequential or parallel in nature.

A comprehensive treatment of grammar systems and a survey of the recent developments in this area can be found in [1]. The study of sequential grammar systems(also called *Cooperating Distributed(CD)Grammar Systems*) was initiated by *Csuhaj-Varju* and *Dassow* [2]. The explicit notion of CD grammar systems was introduced by *Meersman et.al.* [3]. *Gh.Paun et.al.* [4] introduced a problem solving architecture parallel in nature similar to parallel operating grammar systems(also called *Parallel Communicating(PC) Grammar Systems*). They also discuss many relationships of CD and PC Grammar Systems to issues related to distribution, co-operation, parallelism in artificial intelligence, cognitive psychology, robotics, complex systems study etc.

In this paper, the notion of sequential grammar systems is extended to automata and the increase in acceptance power of this system is analyzed. Distributed Automata is a set of automata working together, according to specific rules, to accept one language. Distributed FSAs and Distributed PDAs in t-mode, *-mode, $= k$-mode, $\leq k$-mode, and $\geq k$-mode are discussed. We show that under this notion of distributed automata, distributed FSAs in all the above modes are no powerful than "centralized" FSAs whereas in the case of PDAs in all modes the distributed counterparts are as powerful as Turing Machines. We also consider a restricted version of distributed PDA and show that it also has the same power as that of a Turing Machine.

In the next section we give some definitions and notations needed. In section 3 we consider distributed FSA and in section 4 we consider distributed PDA. Section 5 deals with the restricted version of a PDA and the paper concludes with section 6.

2. Formal Language and Automata Theory Prerequisites

For an alphabet V, we denote by V^* the free monoid generated by V under the operation of concatenation; the empty string is denoted by λ. Moreover $V^+ = V^* - \{\lambda\}$ and $|x|$ is the length of $x \in V^*$. If $U \subseteq V$, then the morphism $pr_U : V^* \to U^*$ defined by $pr_U(a) = a$ if $a \in U$ and $pr_U(a) = \lambda$ if $a \in V - U$, is called a *projection*.

FIN, REG, LIN, CF and RE denote the families of finite, regular, linear, context-free and recursively enumerable languages respectively. For further information in formal language and automata theory the reader is directed to some excellent material in this field.[6, 7, 8]

A grammar (N, T, P, S) is called *non-terminal bounded* if there exists an integer k such that for every word $w \in N^*$ if $S \Rightarrow^* w$, then w has atmost k occurrences of non-terminals. A language is said to be non-terminal bounded if there exists a non-terminal bounded grammar that generates it.

FSA, PDA and TM denote finite state automaton, push-down automaton and

Turing-machine respectively. A PDA is said to perform a turn if the stack goes up and down respectively in two consecutive descriptions of the automata. If the number of turns the PDA can perform to accept a string is bounded by k, then this restricted version of the PDA is called a *k-turn PDA*. One can easily observe that the family of languages accepted by 1-turn PDAs is LIN. It is known that the family of languages that is accepted by k-turn PDAs, $k \geq 1$ is the family of non-terminal bounded languages.

3. Distributed Finite State Automata

In this section we define distributed FSA and consider the language accepted in different modes. We find that the power of distributed FSA is not different from the power of a single "centralized" FSA. Distributed FSA in different modes of acceptance accept only regular sets.

Definition 1 *An n-FSA is a 5-tuple* $\Gamma = (Q, V, \Delta, q_0, F)$ *where,*

1. *Q is a n-tuple $(Q_1, Q_2, \cdots, Q_n)$ where each Q_i is a set of states of the ith component*
2. *V is the finite set of alphabet*
3. *Δ is an n-tuple $(\delta_1, \delta_2, \cdots, \delta_n)$ of state transition functions where each $\delta_i : Q_i \times (V \cup \{\lambda\}) \longrightarrow 2^{\cup_i Q_i}$, $1 \leq i \leq n$*
4. *$q_0 \in \cup_i Q_i$ is the initial state*
5. *$F \subseteq \cup_i Q_i$ is the set of final accepting states.*

Each of the component FSAs of the n-FSA is of the form
$M_i = (Q_i, V, \delta_i)$, $1 \leq i \leq n$. Note that here $Q_i's$ need not be disjoint. In this system, we can consider many modes of acceptance depending upon the number of steps the system has to go through in each of the n components. The different modes of acceptance are t-mode, *-mode, $\leq k$-mode, $\geq k$-mode, and $= k$-mode. Description of each of the above modes of acceptance is as follows:
***t*-mode acceptance:** Initially, the automaton which has the initial state begins the processing of input string. Suppose that the system starts from the component i. In component i the system follows its transition function as any "stand alone" FSA. Suppose in the component i the system arrives at a state q where $q \notin Q_i$ i.e. outside the domain of the transition function of the component i. Then the automaton goes to the jth component $(1 \leq j \leq n)$ provided $q \in Q_j$. If q belongs to more than one Q_j any one of them can be chosen nondeterministically. The jth component acts until it arrives at a state outside its domain for its transition function and the above procedure is repeated. The automaton accepts the string if it reaches any one of the final states. It does not matter which component the system is in.
If for some $i (1 \leq i \leq n)$ $Q_i = Q$ then by no way the system can go out of the *ith* component. In that case *ith* component acts like a "sink".

Definition 2 *The instantaneous description of the n-FSA (ID) is given by a 3-tuple (q, w, i) where $q \in Q,\ w \in V^*,\ 1 \leq i \leq n$.*

In this ID of the n-FSA, q denotes the current state of the whole system, w the portion of the input string yet to be read and i the index of the component in which the system is currently in.

The transition between the ID's is defined as follows:

1. $(q, aw, i) \vdash (q', w, i)$ iff $q' \in \delta_i(q, a)$ where $q, q' \in Q_i, a \in V \cup \{\lambda\}, w \in V^*, 1 \leq i \leq n$
2. $(q, w, i) \vdash (q, w, j)$ iff $q \in Q_j - Q_i$

Let $\vdash^*$ be the reflexive and transitive closure of $\vdash$.

Definition 3 *The language accepted by the n-FSA $\Gamma = (Q, V, \Delta, q_0, F)$ is defined as follows,*

$$L(\Gamma) = \{w \in V^* \mid (q_0, w, i) \vdash (q_f, \lambda, j) \text{ for some } q_f \in F\ 1 \leq j, i \leq n \text{ and } q_0 \in Q_i\}$$

***-mode acceptance:** Initially, the automaton which has the initial state begins the processing of the input string. Suppose the system starts the processing from the component i. Unlike the termination mode, here there is no restriction. The automaton can transfer the control to any of the component at any time if possible. i.e. if there is some j such that $q \in Q_j$ then the system can transfer the control to the component j. The selection is done nondeterministically if there is more than one j.

The instantaneous description and the language accepted by the system in *-mode can be defined analogously.

Theorem 1 *For any n-FSA Γ in t-mode, we have $L(\Gamma) \in REG$.*

Proof. Let $\Gamma = (Q, V, \Delta, q_0, F)$ be a n-FSA in t-mode where $\Delta = (\delta_1, \delta_2, \cdots, \delta_n)$ and the components have states $Q_1, Q_2, \cdots, Q_n$.
Consider the FSA $M = (Q', V, \delta, q'_0, F')$ where,

$$Q' = \{[q, i] \mid q \in \cup_i Q_i,\ 1 \leq i \leq n\} \cup \{q'_0\}$$
$$F' = \{[q_f, i] \mid q_f \in F,\ 1 \leq i \leq n\}$$

δ contains the following transitions:
for each $q_k \in \delta_i(q_j, a),\ q_j \in Q_i,\ a \in V \cup \{\lambda\},\ 1 \leq i \leq n$,

1. $[q_0, i'] \in \delta(q'_0, \lambda)$ such that $q_0 \in Q_{i'}$
2. if $q_k \in Q_i$ then $[q_k, i] \in \delta([q_j, i], a)$
3. if $q_k \in Q_j - Q_i,\ 1 \leq i \leq n$ then $[q_k, j] \in \delta([q_j, i], \lambda),\ 1 \leq j \leq n$

This construction of FSA clearly shows that,

$$L(M) = L(\Gamma)$$
$$\text{So, } L(\Gamma) \in REG$$

□

Theorem 2 *For any n-FSA Γ in *-mode, we have $L(\Gamma) \in REG$.*

Proof. Let $\Gamma = (Q, V, \Delta, q_0, F)$ be a n-FSA in *-mode where $\Delta = (\delta_1, \delta_2, \cdots, \delta_n)$ and the components have states $Q_1, Q_2, \cdots, Q_n$.
Consider the FSA $M = (Q', V, \delta, q_0', F')$ where,

$$Q' = \{[q, i] \mid q \in \cup_i Q_i,\ 1 \leq i \leq n\} \cup \{q_0'\}$$

$$F' = \{[q_f, i] \mid q_f \in F,\ 1 \leq i \leq n\}$$

δ contains the following transitions:

1. $[q_0, i'] \in \delta(q_0', \lambda)$ such that $q_0 \in Q_{i'}$
2. for each $q_y \in \delta_i(q_s, a),\ q_s \in Q_i,\ a \in V \cup \{\lambda\},\ 1 \leq i \leq n$,
   $\{[q_y, i], [q_y, j]\} \subseteq \delta([q_s, i], a),\ 1 \leq j \leq n$ and $q_y \in Q_{j'}$.

This construction of FSA clearly shows that,

$$L(M) = L(\Gamma)$$

$$\text{So, } L(\Gamma) \in REG$$

□

<u>**$=k$-mode acceptance:**</u> Initially, the component which has the initial state begins the processing of the input string. Suppose the system starts the processing from the component i. The system transfers the control to the other component j only after the completion of exactly k number of steps in the component i i.e. if there is a state $q \in Q_j$ then the transition from component i to the component j takes place only if the system has already completed k steps in component i. If there is more than one choice for j the selection is done nondeterministically.
<u>**$\leq k$-mode acceptance:**</u> Initially, the automaton which has the initial state begins the processing of the input string. Suppose the system starts the processing from the component i. The system transfers the control to another component j only after the completion of some $k'(k' \leq k)$ steps in component i i.e. if there is a state $q \in Q_j$ then the transition from component i to component j takes place only if the system has already completed some $k'(1 \leq k' \leq k)$ steps in component i. If there is more than one choice for j the selection is done nondeterministically.
<u>**$\geq k$-mode acceptance:**</u> Initially, the automaton which has the initial state begins the processing of strings. Suppose the system starts the processing from the component i. The system transfers the control to another component j only after the completion of some $k'(k' \geq k)$ steps in the component i i.e. if there is a state $q \in Q_j$ then the transition from component i to component j takes place only if the system has already completed some $k'(k' \geq k)$ steps in the component i. If there is more than one choice for j the selection is done nondeterministically.

The instantaneous description of n-PDA in the above three modes of derivations and the language generated by the them are defined as follows,

Definition 4 *The instantaneous description of the n-FSA (ID) is given by a 4-tuple (q, w, i, j) where $q \in Q,\ w \in V^*\ , 1 \leq i \leq n$, j is a non negative integer.*

In this ID of the n-FSA, q denotes the current state of the whole system, w the portion of the input string yet to be read and i the index of the component in which the system is currently in, and j denotes the number of steps for which the system has been in, in the *ith* component.

Theorem 3 *For any n-FSA Γ in $= k$-mode, we have $L(\Gamma) \in REG$.*

Proof. Let $\Gamma = (Q, V, \Delta, q_0, F)$ be a n-FSA in $= k$-mode where $\Delta = (\delta_1, \delta_2, \cdots, \delta_n)$ and the components have states $Q_1, Q_2, \cdots, Q_n$.
Consider the FSA $M = (Q', V, \delta, q_0', F')$ where,

$$Q' = \{[q, i, j] \mid q \in \cup_i Q_i,\ 1 \leq i \leq n,\ 0 \leq j \leq k\}$$

$$F' = \{[q_f, i, k] \mid q_f \in F,\ 1 \leq i \leq n\}$$

δ contains the following transitions:
for each $q_y \in \delta_i(q_s, a),\ q_s \in Q_i,\ a \in V \cup \{\lambda\},\ 1 \leq i \leq n,\ 0 \leq j \leq k$

1. $[q_0, i', 0] \in \delta(q_0', \lambda)$ such that $q_0 \in Q_i'$
2. if $j < k$ then $[q_y, i, j] \in \delta([q_s, i, j-1], a)$
3. if $j = k$ then $[q_s, j', 0] \in \delta([q_s, i, k], \lambda), 1 \leq j' \leq n$ and $q_s \in Q_j'$.

This construction of FSA clearly shows that,

$$L(M) = L(\Gamma)$$

$$\text{So, } L(\Gamma) \in REG$$

□

In a similar manner we can prove,

Theorem 4 *For any n-FSA Γ in $\leq k$-mode, we have $L(\Gamma) \in REG$.*

Theorem 5 *For any n-FSA Γ in $\geq k$-mode, we have $L(\Gamma) \in REG$.*

Thus we find for n-FSA the different modes of acceptance are equivalent and n-FSA accept only regular set. Basically, the model we have defined is nondeterministic in nature. Restricting the definition to deterministic n-FSA will not decrease the power as any regular set can be accepted by a 1-DFA.

4. Distributed Push Down Automata

In this section we define distributed PDA and consider the language accepted in different modes. We find that the power of distributed PDA is more than the power of a single "centralized" PDA. Distributed PDA with different modes of acceptance have equal power. In the case of PDA usually two types of acceptance viz. acceptance by empty store and acceptance by finite state are considered. Initially we consider only acceptance by final state. Towards the end of the section we show the equivalence to acceptance by empty store.

Definition 5 *An n-PDA is a 7-tuple $M = (Q, V, \Gamma, \Delta, q_0, Z, F)$ where,*

1. *Q is an n-tuple $(Q_1, Q_2, \cdots, Q_n)$ where each Q_i are the set of states for component i*
2. *V is the finite set of input alphabet.*
3. *Γ is an n-tuple $(\Gamma_1, \Gamma_2, \cdots, \Gamma_n)$.*
4. *Δ is an n-tuple $(\delta_1, \delta_2, \cdots, \delta_n)$ of state transition functions where each δ_i : $Q_i \times (V \cup \{\lambda\}) \times \Gamma_i \longrightarrow 2^{\cup_i Q_i \times \Gamma_i^*}$, $1 \leq i \leq n$*
5. *$q_0 \in \cup_i Q_i$ is the initial state.*
6. *Z is an n-tuple $(Z_1, Z_2, \cdots, Z_n)$ where each $Z_i \in \Gamma_i$ $(1 \leq i \leq n)$ is the start symbol of stack for the ith component.*
7. *$F \subseteq \cup_i Q_i$ is the set of final accepting states.*

Each of the component PDAs of the n-PDA is of the form $M_i = (Q_i, V, \Gamma_i, \delta_i, Z_i)$, $1 \leq i \leq n$. Here $Q_i's$ need not be disjoint. As in the case of distributed Finite State Automata we can have several modes of acceptance.

t-mode acceptance: Initially, the component which has the initial state begins the processing of the input string. Suppose the component i has the start state. The ith component starts the processing with the stack having the start symbol Z_i. The processing proceeds in the component i as in a stand alone PDA. Suppose in the component i the system arrives at a state q where $q \notin Q_i$ the system goes to the jth component $(1 \leq j \leq n)$ provided $q \in Q_j$. If there is more than one choice for j we choose any one of them nondeterministically. After choosing a particular jth component the automaton remains in that component until it reaches a state outside the domain of its transition function and the above procedure is repeated. The string is accepted if the automaton reaches any one of the final states. It does not matter which component the system is in or the stacks of the components are empty or not. The presence of multi-stacks increases the generative capacity of the whole system.

Note: As t-mode derivations, we can have other kind of derivations such as *-mode, $= k$-mode, $\leq k$-mode, and $\geq k$-mode. Definitions for them is analogous to the definitions of n-FSA in the respective modes.

Definition 6 *The instantaneous description (ID) of the n-PDA is given by a n+3-tuple $(q, w, \alpha_1, \alpha_2, \cdots, \alpha_n, i)$ where $q \in Q$, $w \in V^*, \alpha_k \in \Gamma_k^*$, $1 \leq i, k \leq n$.*

In this ID of the n-PDA, q denotes the current state of the whole system, w the portion of the input string yet to be read and i the index of the component in which the system is currently in and $\alpha_1, \alpha_2, \cdots, \alpha_n$ the contents of the stacks of the components respectively.
The transition between the ID's is defined as follows:

$$(q, aw, \alpha_1, \alpha_2, \cdots, X\alpha_i, \cdots, \alpha_n, i) \vdash (q', w, \alpha_1, \alpha_2, \cdots, \beta\alpha_i, \cdots, \alpha_n, i)$$

$$\text{iff}$$

$$(q', \beta) \in \delta_i(q, a, X)$$

$$\text{where}$$

$$q \in Q_i, q' \in Q, a \in V \cup \{\lambda\}, w \in V^*, 1 \leq i \leq n,\ \alpha_1, \alpha_2, \cdots, \alpha_n, \beta \in \Gamma^*, X \in \Gamma_i$$

$$(q, w, \alpha_1, \alpha_2, \cdots, \alpha_n, i) \vdash (q, w, \alpha_1, \alpha_2, \cdots, \alpha_n, j)$$

$$\text{iff}$$

$$q \in Q_j - Q_i,$$

$$w \in V^*,\ 1 \leq i, j \leq n,\ \alpha_i \in \Gamma_i^*$$

Let $\vdash^*$ be the reflexive and transitive closure of $\vdash$.

Definition 7 *The language accepted by the n-PDA $M = (Q, V, \Gamma, \Delta, q_0, Z, F)$ is defined as follows,*

$$L(M) = \{w \in V^* \mid (q_o, w, Z_1, Z_2, \cdots, Z_n, i') \vdash^* (q_f, \lambda, \alpha_1, \alpha_2, \cdots, \alpha_n, i)$$

$$\textit{for some } q_f \in F,\ 1 \leq i, i' \leq n,\ \alpha_i \in \Gamma_i^* \textit{ and } q_0 \in Q_{i'}\}$$

Note: Similar to n-FSA we can have other modes of acceptance such as as *-mode, $= k$-mode, $\leq k$-mode, and $\geq k$-mode. These definitions are analogous to the definitions of n-FSA in the respective modes.

The definitions for language acceptance in other modes can be defined in a similar manner. There will be one more component denoting the number of steps in the instantaneous description in the case of $= k$-mode, $\leq k$-mode and $\geq k$-mode.

Example 1 : Consider the =2-mode 2-PDA

$$M = ((Q_1, Q_2), V, (\Gamma_1, \Gamma_2), (\delta_1, \delta_2), \{q_0\}, (Z_1, Z_2), \{q_f\})$$

where

$$\begin{aligned}
Q_1 &= \{q_0, q_1, q_p, q_{p'}, q_s, q_z, q_c, q_{c'}\} \\
Q_2 &= \{q_1, q_2, q_c, q_b, q_f\} \\
V &= \{a, b, c\} \\
Z_1 &= Z \\
Z_2 &= Z \\
F &= \{q_f\} \\
\Gamma_1 &= \{Z, a\} \\
\Gamma_2 &= \{Z, b\}
\end{aligned}$$

δ_1 and δ_2 are defined as follows, with the assumption that $X \in \{Z, a\}$ and $Y \in \{Z, b\}$.

$$
\begin{aligned}
\delta_1(q_0, \lambda, Z) &= \{(q_1, Z)\} && (1)\\
\delta_1(q_1, a, X) &= \{(q_1, aX)\} && (2)\\
\delta_2(q_1, \lambda, X) &= \{(q_2, X)\} && (3)\\
\delta_2(q_2, a, Z) &= \{(q_p, Z)\} && (4)\\
\delta_1(q_p, \lambda, X) &= \{(q_{p'}, aX)\} && (5)\\
\delta_1(q_{p'}, \lambda, X) &= \{(q_1, X)\} && (6)\\
\delta_2(q_2, b, Y) &= \{(q_s, bY)\} && (7)\\
\delta_1(q_s, \lambda, a) &= \{(q_z, \lambda)\} && (8)\\
\delta_1(q_z, \lambda, Z) &= \{(q_c, \lambda)\} && (9)\\
\delta_1(q_z, \lambda, a) &= \{(q_1, a)\} && (10)\\
\delta_2(q_c, c, b) &= \{(q_b, \lambda)\} && (11)\\
\delta_2(q_b, \lambda, Z) &= \{(q_f, \lambda)\} && (12)\\
\delta_2(q_b, \lambda, b) &= \{(q_c, b)\} && (13)\\
\delta_1(q_c, \lambda, Z) &= \{(q_{c'}, Z)\} && (14)\\
\delta_1(q_{c'}, \lambda, Z) &= \{(q_c, Z)\} && (15)
\end{aligned}
$$

The above 2-PDA in =2-mode accepts the following language L:

$$L = \{a^n b^n c^n \mid n \geq 1\}.$$

Explanation: The first component starts the processing. When it uses the first two transitions it should have read an a. Then it switches the control to the second component where it is in the state q_1. After using the λ transition to go to the state q_2, it can either read a a or b . Suppose it reads an a then the system will be in the state q_p. The state q_p is used here to put the already read a on to the first component stack. This task is carried out by fifth and sixth rules. Suppose when in the second component it reads a b then the system will be in the state q_s, which is used to see whether there is one a for each b. This task is carried out by eighth, ninth and tenth rules. Immediately after seeing there is no more a's in the first component's stack then it realizes that number of b's is equal to number of b's and it goes to the state q_c, which is used to read c's. This task is carried out by ninth rule. After reading each and every c through rule eleventh it will erase a b. When there is no b left in the stack the system will be in the final state q_f. If there are more b's left in the second component stack then the system will be in the state q_c. Then it uses the last two λ rules in the first component and repeats the above procedure until it arrives at the state q_f.
Example 2 : Consider the t-mode 2-PDA

$$M = ((Q_1, Q_2), V, (\Gamma_1, \Gamma_2), (\delta_1, \delta_2), \{q_0\}, (Z_1, Z_2), \{q_f\})$$

where

$$\begin{array}{rcl} Q_1 & = & \{q_0, q_a, q_b, q_T\} \\ Q_2 & = & \{q_{a'}, q_{b'}, q_s, q_f\} \\ V & = & \{a, b\} \\ F & = & \{q_f\} \\ \Gamma_1 & = & \{Z, a, b\} \\ \Gamma_2 & = & \{Z, a, b\} \end{array}$$

where δ_1 and δ_2 are defined as follows, with the assumption that $X \in \{Z_1, a, b\}$ and $Y \in \{Z_2, a, b\}$.

$$\begin{array}{rcll} \delta_1(q_0, a, Z_1) & = & \{(q_a, aZ_1)\} & (1) \\ \delta_1(q_0, b, Z_1) & = & \{(q_b, bZ_1)\} & (2) \\ \delta_1(q_a, b, X) & = & \{(q_a, bX)\} & (3) \\ \delta_1(q_b, a, X) & = & \{(q_b, aX)\} & (4) \\ \delta_1(q_a, a, X) & = & \{(q_T, X), (q_a, aX)\} & (5) \\ \delta_1(q_b, b, X) & = & \{(q_T, X), (q_b, bX)\} & (6) \\ \delta_1(q_T, \lambda, a) & = & \{(q_{a'}, \lambda)\} & (7) \\ \delta_1(q_T, \lambda, b) & = & \{(q_{b'}, \lambda)\} & (8) \\ \delta_2(q_{a'}, \lambda, Y) & = & \{(q_T, aY)\} & (9) \\ \delta_2(q_{b'}, \lambda, Y) & = & \{(q_T, bY)\} & (10) \\ \delta_1(q_T, \lambda, Z_1) & = & \{(q_e, Z_1)\} & (11) \\ \delta_2(q_e, \lambda, \{a, b\}) & = & \{(q_s, \lambda)\} & (12) \\ \delta_2(q_s, a, a) & = & \{(q_s, \lambda)\} & (13) \\ \delta_2(q_s, b, b) & = & \{(q_s, \lambda)\} & (14) \\ \delta_2(q_s, \lambda, Z_2) & = & \{(q_f, Z_2)\} & (15) \end{array}$$

The above 2-PDA in t-mode accepts the following language L:

$$L = \{ww \mid w \in \{a, b\}^*\}.$$

Explanation: From q_0 the first component either reads a or b and stores the information that the first alphabet is a or b by entering the state q_a or q_b respectively. It also stacks the first alphabet already read. This is done by the first two rules. From q_a or q_b it reads a or b and stacks the read alphabet in the first component stack. This is done by rules 3,4,5 and 6. In $q_x (x \in \{a, b\})$ if the first component reads x then it could be the start of the string identical to the string read. So, in order to check that, the stacked up alphabets in the first component are transfered to the second component stack. This is done by the rules 7,8,9, and 10. After transferring the stack, the system will be in the state q_e. The second component in state q_e erases the top alphabet, since it has already checked that the first alphabet matches. This is carried out by the rule 12. Rules 13,14,15 checks whether the read alphabet matches with the top stack alphabet.

We know that a two stack machine can simulate a Turing Machine [8]. Hence a 2-PDA is as powerful as a TM. Thus we have,

Theorem 6 *For any $L \in RE$, there is a 2-PDA M such that $L = L(M)$.*

Theorem 7 *For every distributed PDA in *-mode there is an equivalent distributed PDA in t-mode and for every distributed PDA in t-mode there is an equivalent distributed PDA in *-mode.*

Proof. First we prove the existence of distributed PDA in t-mode for every distributed PDA in *-mode. Let $M = (Q, V, \Gamma, \Delta, q_0, Z, F)$ be a n-PDA in *-mode. Let

$$\begin{aligned} Q &= (Q_1, Q_2, \cdots, Q_n) \\ \Gamma &= (\Gamma_1, \Gamma_2, \cdots, \Gamma_n) \\ \Delta &= (\delta_1, \delta_2, \cdots, \delta_n) \\ Z &= (Z_1, Z_2, \cdots, Z_n) \end{aligned}$$

Let the language generated be $L(M)$. The required n-PDA M' in t-mode is constructed as follows: $M' = (Q', V, \Gamma, \Delta', q_0, Z, F)$ where $Q' = (Q'_1, Q'_2, \cdots, Q'_n)$ is defined as follows,

1. $Q'_i = Q_i \cup \{q_{ji} \mid \forall q \in Q_i \cap Q_j,\ 1 \le j \le n,\ j \ne i\},\ 1 \le i \le n$
2. δ'_i includes all elements of δ_i.
3. If $q \in Q_i \cap Q_j$ $1 \le i, j \le n,\ i \ne j$ then
   (a) $(q_{ij}, Y) \in \delta'_i(p, a, X)$ if $(q, Y) \in \delta_i(p, a, X)$, $a \in V$ and $X \in \Gamma_i^*$
   (b) $(q, X) \in \delta'_j(q_{ij}, \lambda, X)$ $X \in \Gamma_j^*$

The difference between these *-mode and t-mode acceptance arises when there is a state in more than one component. Suppose there is a state $q \in Q_i \cap Q_j$ $(1 \le i, j \le n)$: the *-mode can change its component either from its present ith component to jth component or vice versa, but t-mode system will not be able to change its present component. So, in order to build a t-mode n-PDA accepting the same language, we add two new states q_{ij} and q_{ji} for each state such as q of which q_{ji} is added to the component Q_i and q_{ij} to the component Q_j. Suppose the system is in the component i. Whenever the system arrives at the state s, where s is a state in the component i from where $q \in Q_i \cap Q_j$ can be arrived at we include the state q_{ij} in the transition function $\delta'_i(s, a, X)$, thus making it possible for the system to change the component to j with the state q. It is clear from the construction that the language generated by the constructed system is nothing but $L(M)$.

Now we prove the other way. Let $M = (Q, V, \Gamma, \Delta, q_0, Z, F)$ be a n-PDA in t-mode. Let

$$\begin{aligned} Q &= (Q_1, Q_2, \cdots, Q_n) \\ \Gamma &= (\Gamma_1, \Gamma_2, \cdots, \Gamma_n) \end{aligned}$$

$$\begin{aligned}\Delta &= (\delta_1, \delta_2, \cdots, \delta_n)\\ Z &= (Z_1, Z_2, \cdots, Z_n)\end{aligned}$$

Let the language generated be $L(M)$. The required n-PDA M' in *-mode is constructed as follows: M is modified to M' in *-mode by the following procedure:

1. Do for $i = 1$ *to* $n - 1$
2. Do for $j = i + 1$ *to* n
   Begin
3. For every $q \in Q_i \cap Q_j$ do the following:
   (a) Replace q with q_j in Q_j and in all its transitions.
   (b) If there is some Q_k other than Q_i and Q_j and if q acts as a transition state to component j then replace q with q_j in Q_k.

It is straight forward to show that the modified n-PDA accepts the same language in *-mode. □

Theorem 8 *For every distributed PDA in = k-mode there is an equivalent distributed PDA in t-mode.*

Proof. Let $M = (Q, V, \Gamma, \Delta, q_0, Z, F)$ be a n-PDA in $= k$-mode. Let

$$\begin{aligned}Q &= (Q_1, Q_2, \cdots, Q_n)\\ \Gamma &= (\Gamma_1, \Gamma_2, \cdots, \Gamma_n)\\ \Delta &= (\delta_1, \delta_2, \cdots, \delta_n)\\ Z &= (Z_1, Z_2, \cdots, Z_n)\end{aligned}$$

Let the language generated by the system be $L(M)$. The required n-PDA in t-mode $M' = (Q', V, \Gamma, \Delta', [q_0, 1], Z, F')$ where $Q' = (Q'_1, Q'_2, \cdots, Q'_n)$ and $\Delta' = (\delta'_1, \delta'_2, \cdots, \delta'_n)$ is defined as follows:

1. $Q'_i = \{[q, x] \mid q \in Q_i \text{ and } 1 \leq x \leq k\} \cup \{q_{ji} \mid q \in Q_i \cap Q_j,\ 1 \leq j \leq n \text{ and } i \neq j\}$
2. δ'_i is defined as follows:
   (a) $([s, l+1], y) \in \delta'_i([p, l], a, X)$ provided $(s, y) \in \delta'_i(p, a, X)$ and $l < k$
   (b) If $l = k$ then $(s_{ij}, y) \in \delta'_i([q, l], a, X)$ if $s \in Q_i \cap Q_j$ for $X \in \Gamma_i$ and $(s, y) \in \delta_i(q, a, X)$
   (c) $([q, 1], X) \in \delta'_j(q_{ij}, \lambda, X)$ for $X \in \Gamma_j$
3. $F' = \{[q, k] \mid q \in F\}$

The difference between these $= k$-mode and t-mode derivations arises in the number of steps the derivation is carried out in a particular component and when there is a state in more than one-component. Suppose there is a state $q \in Q_i \cap Q_j$ $(1 \leq i, j \leq n)$: the $= k$-mode can change its component either from its present ith component to jth component provided it has completed k steps in the component i or vice versa, but t-mode system will not be able to change its present component.

Suppose the system is in the component i. Whenever the $= k$-mode n-PDA M, starts the processing from the state q the t-mode n-PDA M', starts the processing from the state $[q, 1]$. If after the first derivation the $= k$-mode n-PDA goes to the state s then correspondingly the t-mode n-PDA goes to the state $[s, 2]$. If after the second derivation the $= k$-mode n-PDA goes to the state s' then correspondingly the t-mode n-PDA goes to the state $[s', 3]$. This procedure is extended upto $k - 1$ derivations. So, if after $k - 1$ derivations the $= k$-mode n-PDA is in the state r then the t-mode n-PDA will be in the state $[r, k]$. It means that after one more derivation the system has to change its component. So, if from the state r the $= k$-mode n-PDA goes to the state q where $q \in Q_i \cap Q_j$ for some j then the t-mode n-PDA goes to the state q_{ij}. The state q_{ij} is only in the component j so the system in t-mode has to change its component to j. From the state q_{ij} by λ-transition the t-mode system goes to the state $[q, 1]$ and the processing proceeds in the component j.
From the construction itself it should be clear that the language generated by both the systems are the same. $\Box$
The above construction is explained via an example below.
Example 3 : We take the example 1 and construct the equivalent t-mode n-PDA as follows:

$$\begin{aligned} Q'_1 &= \{[q, x] \mid 1 \leq x \leq k, \ \forall q \in Q_1\} \cup \{q_1^{21}, q_c^{21}\} \\ Q'_2 &= \{[q, x] \mid 1 \leq x \leq k, \ \forall q \in Q_2\} \cup \{q_1^{12}, q_c^{12}, q_f\} \end{aligned}$$

where start state is $[q_0, 1]$ and final state is q_f

The transition functions are defined as follows, on the assumption that $X \in \{Z, a\}$ and $Y \in \{Z, b\}$.

$$\delta'_1([q_0, 1], \lambda, Z) = \{([q_1, 2], Z)\} \quad (1)$$
$$\delta'_1([q_1, 2], a, X) = \{(q_1^{12}, aX)\} \quad (2)$$
$$\delta'_2([q_1, 1], \lambda, Y) = \{([q_2, 2], Y)\} \quad (3)$$
$$\delta'_2([q_2, 2], a, Z) = \{([q_p, 1], Z)\} \quad (4)$$
$$\delta'_1([q_p, 1], \lambda, X) = \{([q_{p'}, 2], aX)\} \quad (5)$$
$$\delta'_1([q_{p'}, 2], \lambda, X) = \{(q_1^{12}, X)\} \quad (6)$$
$$\delta'_2([q_2, 2], b, Y) = \{([q_s, 1], bY)\} \quad (7)$$
$$\delta'_1([q_s, 1], \lambda, a) = \{([q_z, 2], \lambda)\} \quad (8)$$
$$\delta'_1([q_z, 2], \lambda, Z) = \{(q_c^{12}, \lambda)\} \quad (9)$$

$$
\begin{aligned}
\delta_1'([q_z, 2], \lambda, a) &= \{(q_1^{12}, a)\} && (10)\\
\delta_2'([q_c, 1], c, b) &= \{([q_b, 2], \lambda)\} && (11)\\
\delta_2'([q_b, 2], \lambda, Z) &= \{(q_f, \lambda)\} && (12)\\
\delta_2'([q_b, 2], \lambda, b) &= \{(q_c^{21}, b)\} && (13)\\
\delta_1'([q_c, 1], \lambda, Z) &= \{([q_{c'}, 2], Z)\} && (14)\\
\delta_1'([q_{c'}, 2], \lambda, Z) &= \{(q_c^{12}, Z)\} && (15)\\
\delta_2'(q_1^{12}, \lambda, Y) &= \{([q_1, 1], Y)\} && (16)\\
\delta_2'(q_c^{12}, \lambda, Y) &= \{([q_c, 1], Y)\} && (17)\\
\delta_1'(q_1^{21}, \lambda, X) &= \{([q_1, 1], X)\} && (18)\\
\delta_1'(q_c^{21}, \lambda, X) &= \{([q_c, 1], X)\} && (19)
\end{aligned}
$$

Theorem 9 *For every distributed PDA in t-mode there is an equivalent distributed PDA in $= k$-mode.*

Proof. Let $M = (Q, V, \Gamma, \Delta, q_0, Z, F)$ be a n-PDA in t-mode. Let

$$
\begin{aligned}
Q &= (Q_1, Q_2, \cdots, Q_n)\\
\Gamma &= (\Gamma_1, \Gamma_2, \cdots, \Gamma_n)\\
\Delta &= (\delta_1, \delta_2, \cdots, \delta_n)\\
Z &= (Z_1, Z_2, \cdots, Z_n)
\end{aligned}
$$

Let the language generated by the system be $L(M)$. Let $Q = \cup_i Q_i$. The n+2-PDA $M' = (Q', V, \Gamma', \Delta', q_0, Z', F')$ in $= k$-mode is constructed as follows:

$$
\begin{aligned}
Q' &= (Q_1', Q_2', \cdots, Q_n', Q_{n+1}', Q_{n+2}')\\
\Gamma' &= (\Gamma_1', \Gamma_2', \cdots, \Gamma_n', \Gamma_{n+1}', \Gamma_{n+2}')\\
\Delta' &= (\delta_1', \delta_2', \cdots, \delta_n', \delta_{n+1}', \delta_{n+2}')\\
Z' &= (Z_1, Z_2, \cdots, Z_n, Z_{n+1}, Z_{n+2})
\end{aligned}
$$

1. $Q_i' = Q_i \cup \{[q, s] \mid 1 \leq s \leq k,\ q \in Q_i\} \cup \{r_{qi}' \mid q \in Q_i\} \cup \{r_{q_{x,s}} \mid q \in Q_{T_i},\ 1 \leq i \leq n,\ q_x$ is the state outside the domain of thecomponent i which can be arrived from the state $q, 1 \leq s \leq k,\ 1 \leq x \leq z$, where z is a positive integer$\} \cup \{p_{i_1}, p_{i_2}, \cdots, p_{i_k}\}$ for $1 \leq i \leq n$.

2. $Q_{n+1}' = \{r_{qi} \mid q \in Q_i, 1 \leq i \leq n\} \cup \{d_2, d_3, \cdots, d_k\}$.

3. $Q_{n+2}' = \{f_1, f_2, \cdots, f_k\}$.

4. $\Gamma_i' = \Gamma_i$ for $1 \leq i \leq n$.

5. $\Gamma_{n+1}' = \{Z_{qi} \mid q \in Q_i, 1 \leq i \leq n\} \cup \{Z_{n+1}\}$.

6. $\Gamma_{n+2}' = \{Z_{n+2}\}$.

Let Q_{T_i} be the set of all states q in $Q - Q_i$ such that q is a transition state for the component i and let $N(Q_{T_i})$ be the set of all states q in Q_i such that from q the system can arrive at a state $p \in Q_{T_i}$ by a single transition. We define the transistion function $\delta'_i(1 \leq i \leq n)$ as follows,

1. $([q,1], YX) \in \delta'_i(q_0, a, X)$ if $(q, YX) \in \delta_i(q_0, a, X)$ $\forall i$ s.t $q_0 \in Q_i,\ 1 \leq i \leq n$.
2. $([q,1], X) \in \delta'_i(q, \lambda, X)$ $\forall i,\ 1 \leq i \leq n$.
3. $([q,1], X) \in \delta'_i(r'_{qi}, \lambda, Y)$.
4. $(p_{i_{k-s+2}}, Z_{qi}X) \in \delta'_i(r_{q_{x,s}}, a, X)$.
5. $(p_{i_{j-1}}, Z_{qi}) \in \delta'_i(p_{i_j}, \lambda, Z_{qi})$ for $2 \leq j \leq k$.
6. $(q, \lambda) \in \delta_i(p_{i_1}, \lambda, Z_{qi})$.

Case 1 Let $q \notin F$

1. if $q \in N(Q_{T_i})$ and $s < k-1$ then $\{(r_{q_{x,s}}, X) \mid x$ is a finite integer $\geq 1\} \cup \{([p, s+1], YX)\} \in \delta'_i([q,s], a, X)$ if $(p, YX) \in \delta_i(q, a, X)$.
2. if $q \in N(Q_{T_i})$ and $s = k-1$ then $(p, YX) \in \delta'_i([q,s], a, X)$ if $(p, YX) \in \delta_i(q, a, X)$.
3. if $q \notin N(Q_{T_i})$ and $s < k-1$ then $([p, s+1], YX) \in \delta'_i([q,s], a, X)$ if $(p, YX) \in \delta_i(q, a, X)$.
4. if $q \notin N(Q_{T_i})$ and $s = k-1$ then $(r_{qi}, X) \in \delta_i([q,s], a, X)$.

Case 2 Let $q \in F$

1. if $q \in N(Q_{T_i})$ and $s < k-1$ then $(f_{k-s+1}, X) \in \delta'_i([q,s], a, X)$.
2. if $q \in N(Q_{T_i})$ and $s = k-1$ then $(p, YX) \in \delta'_i([q,s], a, X)$ if $(p, YX) \in \delta'_i((q, a, X)$.
3. if $q \notin N(Q_{T_i})$ and $s < k-1$ then $\{(f_{k-s+1}, X), ([p, s+1], YX)\} \in \delta'_i([q,s], a, X)$ if $(p, YX) \in \delta_i((q, a, X)$.
4. if $q \notin N(Q_{T_i})$ and $s = k-1$ then $\{(f_1, X), (r_{qi}, X)\} \in \delta'_i([q,s], a, X)$.

δ'_{n+1} definition is as follows,

1. $(d_2, Z_{qi}Z_{n+1}) \in \delta'_{n+1}(r_{qi}, \lambda, Z_{n+1})$.
2. $(d_{i+1}, Y) \in \delta'_{n+1}(d_i, \lambda, Y)$ for $2 \leq i \leq k-1$.
3. $(r'_{qi}, Z_{n+1}) \in \delta'_{n+1}(d_k, \lambda, Z_{qi}Z_{n+1})$.

δ'_{n+2} is defined as,

1. $(f_{i-1}, Z_{n+2}) \in \delta'_{n+2}(f_i, \lambda, Z_{n+2})$ for $2 \leq i \leq k$.

The set of final states F' is defined as

$$F' = \{f_1\} \cup \{p \in F \mid \exists q \; s.t \; \delta_i([q, k-1], a, X) \text{ contains } p \text{ for some } i, \; a \in V \text{ and } X \in \Gamma'_i\}$$

The states $\{p_{i_1}, p_{i_2}, \cdots, p_{i_k}\}$ are added to the component i to make sure that each component does not do less than k number of processing before proceeding on to the next component. The new component Q_{n+1} is added in order to make sure that a component does not go beyond k number of steps in a component. Its stack stores the state with which the system has to carry on processing after being "dummy" in the component Q_{n+1}. The component Q_{n+2} make sure that the system goes to the final state only after k number of processing in the component i.
From the construction itself it should be clear that the language generated by the constructed n-PDA in $= k$-mode is nothing but $L(M)$. □
In a similar manner we can prove.

Theorem 10 *For every distributed PDA in $\leq k$-mode there is an equivalent distributed PDA in t-mode.*

Theorem 11 *For every distributed PDA in $\geq k$-mode there is a distributed PDA in t-mode.*

The converse of the above two theorems follows from the equivalence of n-PDA in $= k$-mode and t-mode. Upto this we have considered the language accepted by the "final state acceptance" in a n-PDA. Similarly we can define language accepted by "empty store". The language accepted by n-PDA by "empty store acceptance" is defined as follows,

Definition 8 *The language accepted by the n-PDA $M = (Q, V, \Gamma, \Delta, q_0, Z, F)$ by "empty store acceptance"is defined as follows,*

$$N(M) = \{w \in V^* \mid (q_o, w, Z_1, Z_2, \cdots, Z_n, i') \vdash^* (q, \lambda, \lambda, \lambda, \cdots, \lambda(n \; times), i)$$

$$\text{for some } q \in Q_i, \; 1 \leq i, i' \leq n, \quad \text{and } q_0 \in Q_{i'}\}$$

The equivalence of acceptance by final state and empty store in a n-PDA in t-mode is proved by the following theorems,

Theorem 12 *If L is $L(M_2)$ for some n-PDA M_2, then L is $N(M_1)$ for some n-PDA M_1.*

Proof. Let $M_2 = (Q, V, \Gamma, \Delta, q_0, Z, F)$ be a n-PDA in t-mode where the acceptance is by final state. Let

$$Q = (Q_1, Q_2, \cdots, Q_n)$$
$$\Gamma = (\Gamma_1, \Gamma_2, \cdots, \Gamma_n)$$
$$\Delta = (\delta_1, \delta_2, \cdots, \delta_n)$$
$$Z = (Z_1, Z_2, \cdots, Z_n)$$

The n-PDA $M_1 = (Q', V, \Gamma', \Delta', q_0, Z', \phi)$ in t-mode where the acceptance is by empty store is constructed as follows: $Q' = (Q'_1, Q'_2, \cdots, Q'_n)$ and $\Delta' = (\delta'_1, \delta'_2, \cdots, \delta'_n)$

1. $Q'_i \;=\; Q_i \cup \{q_i\},\ 1 \le i \le n$
2. $\Gamma'_i \;=\; \Gamma_i \cup \{Z'_i\}$
3. For each i δ'_i includes all elements of δ_i
4. $\delta'_i(q, \lambda, Z'_i)$ contains $(q, Z_i Z'_i)$
5. For all $i(1 \le i \le n)$ if $q \in F \cap Q_i$ then $\delta'_i(q, \lambda, X) \;=\; (q_1, X)$
6. For $1 \le i \le n-1$
   (a) $\delta'_i(q_i, \lambda, X)$ contains $(q_i, \lambda),\ X \in \Gamma_i$
   (b) $\delta'_i(q_i, \lambda, Z_i)$ contains (q_{i+1}, λ)
7. $\delta'_n(q_n, \lambda, Z'_n)$ contains (q_n, λ)

Whenever the system enters a final state string read by the system should be accepted by the system. i.e. the stacks of the components should be emptied. For this, as soon as the system enters the final state it has the possibility of going to the first component through the state q_1. When in the state q_1 the system empties the first component stack and enters the second component through the state q_2 and the procedure is repeated. In the state q_n the system empties the stack of the nth component. It is straight forward to prove that $L(M_2) \;=\; N(M_1)$. □

Theorem 13 *If L is $N(M_1)$ for some n-PDA M_1 in t-mode, then L is $L(M_2)$ for some $n+1$-PDA M_2 in t-mode.*

Proof. Let $M_1 \;=\; (Q, V, \Gamma, \Delta, q_0, Z, \phi)$ be a n-PDA in t-mode. Let

$$Q \;=\; (Q_1, Q_2, \cdots, Q_n)$$

$$\Gamma \;=\; (\Gamma_1, \Gamma_2, \cdots, \Gamma_n)$$

$$\Delta \;=\; (\delta_1, \delta_2, \cdots, \delta_n)$$

$$Z \;=\; (Z_1, Z_2, \cdots, Z_n)$$

The n+1-PDA $M_2 \;=\; (Q', V, \Gamma', \Delta', q_0, Z', \{q_f\})$ where $Q' \;=\; (Q'_1, Q'_2, \cdots, Q'_n, Q'_{n+1})$, $\Gamma' \;=\; (\Gamma'_1, \Gamma'_2, \cdots, \Gamma'_n, \Gamma'_{n+1})$, $\Delta' \;=\; (\delta'_1, \delta'_2, \cdots, \delta'_n, \delta'_{n+1})$ and $Z' \;=\; (Z'_1, Z'_2, \cdots, Z'_n, Z'_{n+1})$ is constructed as follows:

1. $Q'_i \;=\; Q_i \cup \{r'_{qi} \mid q \in Q_i\} \cup \{q_i\},\ 1 \le i \le n$
2. $Q'_{n+1} \;=\; \{r_{qi} \mid q \in \cup_i Q_i\} \cup \{q_f, q_g\}$
3. $\Gamma'_i \;=\; \Gamma_i \cup \{Z'_i\},\ 1 \le i \le n$
4. $\Gamma'_{n+1} \;=\; \{r_{qi} \mid q \in \cup_i Q_i\} \cup \{Z'_{n+1}\}$
5. $\delta'_i(q, \lambda, Z'_i)$ contains $(q, Z_i Z'_i)$
6. δ'_i includes all elements of δ_i

7. For $1 \leq i \leq n$, $\delta'_i(q, \lambda, Z'_i)$ contains (r_{qi}, Z'_i)
8. $\delta'_{n+1}(r_{qi}, \lambda, Z'_{n+1})$ contains $(q_1, Z_{qi}Z_{n+1})$
9. $\delta'_i(q_i, \lambda, Z'_i)$ contains $(q_{i+1}, Z'_i),\ 1 \leq i \leq n-1$
10. $\delta'_i(q_i, \lambda, X)$ contains $(q_g, X),\ 1 \leq i \leq nX \in \Gamma_i$
11. $\delta'_n(q_n, \lambda, Z'_n)$ contains (q_f, λ)
12. $\delta'_{n+1}(q_g, \lambda, Z_{qi})$ contains (r'_{qi}, λ)
13. $\delta'_i(r'_{qi}, \lambda, X)$ contains $(q, X),\ 1 \leq i \leq n$

Whenever the system's stacks are empty the system enters the new final state q_f included in the newly added component Q'_{n+1}. For this, if the system in the state q in the component i sees its stack is empty the system enters the state r_{qi} which is only in the newly added state Q'_{n+1}. In the component Q'_{n+1} the state r_{qi} is put into its stack to store the information which state and which component the system has to go if the system sees that some stacks are non-empty. After stacking the state r_{qi}, the system uses the states q_j to see whether the stack of the component j is empty or not. If it is empty it goes to the next component to check the emptiness of the next component. If not, it enters the state q_g which is in the component $n+1$ to get the information from which state and component it has to continue the processing. This work is done by the state r'_{qi}.

It is straight forward to prove that $L(M_2) = N(M_1)$. □

The equivalence of the "empty store acceptance" and the "final state acceptance" in the other modes can be proved similarly.

5. Distributed k-turn PDA

In this section, we consider a restricted version of the distributed PDA discussed in the previous section. Here we consider only *-mode acceptance. Also, we take $Q = Q_1 = Q_2 = \cdots = Q_n$. In this restricted version of distributed PDA, the stacks of the component PDAs can perform atmost a k number of turns while accepting a string. A n-PDA in which the stack of each of the components can perform atmost k turns each is called a *k-turn n-pushdown automata* (k-turn n-PDA). If n is not explicitly mentioned, then the system is called *k-turn distributed PDA* and is denoted by the symbol k-turn $*$-PDA. The following corollary immediately follows from this definition of k-turn n-PDA and the fact that the family of languages accepted by k-turn PDAs is the family of non-terminal bounded languages.

Corollary 1 *For every non terminal bounded language L, there exists a k-turn n-PDA M for some $k \geq 1$ such that $L(M) = L$.*

In the case of k-turn n-PDA, we note that each of the k-turn component PDAs of the entire system can be replaced by k "1-turn component PDA"s (with the order in which the component PDAs are being used being coded in the state information). With this observation we have the following theorem.

Theorem 14 *For any k-turn *-PDA M_1, there exists a 1-turn *-PDA M_2 such that $L(M_1) = L(M_2)$.*

This theorem tells us that we can restrict our attention to 1-turn *-PDA as far as analyzing accepting power is concerned.

It is of interest to note that 1-turn *-PDA accept certain languages which are not context-free in nature.

Example 4 : Consider

$$\begin{aligned} M_1 &= (\{q_1, \ldots, q_6\}, \{a, b, c\}, q_1, \{q_6\}, \\ &\quad (\{Z, a, b, c\}, \delta_1, Z), (\{Z, a, b, c\}, \delta_2, Z)) \end{aligned}$$

where

$$\begin{aligned} (r1) &: \delta_1(q_1, a, X) = \{(q_1, aX)\}, X \in \{Z, a\} \\ (r2) &: \delta_1(q_1, b, a) = \{(q_2, \lambda)\} \\ (r3) &: \delta_2(q_2, \lambda, X) = \{(q_3, bX)\}, X \in \{Z, b\} \\ (r4) &: \delta_1(q_3, b, a) = \{(q_2, \lambda)\} \\ (r5) &: \delta_1(q_3, c, Z) = \{(q_4, Z)\} \\ (r6) &: \delta_2(q_4, \lambda, b) = \{(q_5, \lambda)\} \\ (r7) &: \delta_2(q_5, c, b) = \{(q_5, \lambda)\} \\ (r8) &: \delta_2(q_5, \lambda, Z) = \{(q_6, \lambda)\} \end{aligned}$$

In this automaton, the prefix a^n is first pushed onto the stack of component 1 by rule $(r1)$. On encountering b^m, rules $(r2), (r3), (r4)$, for each 'b' read off a character from the input string, pop an 'a' from the stack of component 1 and push a 'b' on the stack of component 2. Rule $(r5)$ checks whether an exact matching takes place (ie., $n = m$). Rules $(r6)$ and $(r7)$ similarly match the substring c^p with b^m. If an exact matching occurs (checked by rule $(r8)$), the final state q_6 is entered and the string $a^n b^n c^n$ is accepted. Thus

$$L(M_1) = \{a^n b^n c^n \mid n \geq 1\}$$

We observe that both the component PDAs perform 1 turn each in this example.

Example 5 : Consider

$$\begin{aligned} M_2 &= (\{q_{10}, q_{11}, q_{20}, q_{2a}, q_{2b}, q_f\}, \{a, b, c\}, q_{10}, \{q_f\}, \\ &\quad (\{Z, a, b\}, \delta_1, Z), (\{Z, a, b\}, \delta_2, Z)) \end{aligned}$$

where

$$\begin{aligned} (r1) &: \delta_1(q_{10}, p, X) = \{(q_{10}, pX)\}, X \in \{Z, a, b\}, p \in \{a, b\} \\ (r2) &: \delta_1(q_{10}, c, a) = \{(q_{2a}, \lambda)\} \\ (r3) &: \delta_1(q_{10}, c, b) = \{(q_{2b}, \lambda)\} \\ (r4) &: \delta_2(q_{2a}, \lambda, X) = \{(q_{11}, aX)\}, X \in \{Z, a, b\} \\ (r5) &: \delta_2(q_{2b}, \lambda, X) = \{(q_{11}, bX)\}, X \in \{Z, a, b\} \end{aligned}$$

$$\begin{aligned}
(r6) &\quad : \quad \delta_1(q_{11}, \lambda, a) = \{(q_{2a}, \lambda)\} \\
(r7) &\quad : \quad \delta_1(q_{11}, \lambda, b) = \{(q_{2b}, \lambda)\} \\
(r8) &\quad : \quad \delta_1(q_{11}, \lambda, Z) = \{(q_{20}, Z)\} \\
(r9) &\quad : \quad \delta_2(q_{20}, a, a) = \{(q_{20}, \lambda)\} \\
(r10) &\quad : \quad \delta_2(q_{20}, b, b) = \{(q_{20}, \lambda)\} \\
(r11) &\quad : \quad \delta_2(q_{20}, \lambda, Z) = \{(q_f, \lambda)\}
\end{aligned}$$

In this automaton, rule $(r1)$ pushes the string $w(\in \{a, b\}^*)$ onto the stack of component 1. On encountering a 'c' in the input string (rules $(r2), (r3)$), the string w is reversed into the stack of component 2 by a series of pops and pushes (rules $(r4), (r5), (r6), (r7)$). When the string is completely reversed (indicated by rule $(r8)$), the remainder of the input string is matched with the contents of the stack of component 2 (rules $(r9), (r10)$). If a perfect match occurs, then the final state q_f is entered and the string $wcw, w(\in \{a, b\}^*)$ is accepted. Thus

$$L(M_2) = \{wcw \mid w \in \{a, b\}^*\}$$

As in the previous example, we observe that both the component PDAs perform 1 turn each in this example.

We need the following closure properties of distributed k-turn PDAs to show their equivalence with TMs

Theorem 15 *The family of languages accepted by* 1*-turn *-PDAs is closed under the following operations.*
(i) Morphism
(ii) Intersection with Regular Sets

(i) **Morphism:** Let
$M = (Q, V_1, q_0, F, (\Gamma_1, \delta_1, Z_1), \ldots, (\Gamma_n, \delta_n, Z_n))$ be any 1-turn n-PDA of and $h : V_1 \to V_2$ be any morphism.
Construct a 1-turn n-PDA as shown below:
$M' = (Q', V_2, q_0, F, (\Gamma_1, \delta'_1, Z_1), \ldots, (\Gamma_n, \delta'_n, Z_n))$
where

$$\begin{aligned}
Q' = Q \cup \{[q, a, X, i, j] \mid \quad & |h(a)| > 1, h(a) = a_0 a_1 \ldots a_p \in V_2^*, 0 \leq i < p, \\
& q \in Q, a \in V_1, X \in \Gamma_j, 1 \leq j \leq m\}
\end{aligned}$$

The rules of M' are given by
(1) $\delta'_j(q, \lambda, X) = \delta_j(q, \lambda, X), q \in Q, X \in \Gamma_j, 1 \leq j \leq m$
(2) If $(q', \gamma) \in \delta_j(q, a, X), q \in Q, a \in V_1, X \in \Gamma_j, \gamma \in \Gamma_j^*$
  (i) If $h(a) = \lambda$, then $(q', \gamma) \in \delta'_j(q, \lambda, X)$
  (ii) If $|h(a)| = 1$, then $(q', \gamma) \in \delta_j(q, h(a), X)$

(iii) If $|h(a)| > 1$, say $h(a) = a_0a_1 \dots a_p \in V_2^*$
$([q, a, X, 0, j], X) \in \delta'_j(q, a_0, X)$
$([q, a, X, i, j], X) \in \delta'_j([q, a, X, i-1, j], a_i, X), 1 \leq i < p$
$(q', \gamma) \in \delta'_j([q, a, X, p-1, j], a_p, X)$
M' is constructed in such a manner that M' responds to $h(w), (w \in V_1^*)$ in exactly the same way as M does to w. Thus we have

$$L(M') = h(L(M))$$

(ii) **Intersection with Regular sets:** Let $M = (Q_1, V, q_{10}, F_1, (\Gamma_1, \delta_1, Z_1), \dots, (\Gamma_n, \delta_n, Z_n))$ be any 1-turn n-PDA and $M_2 = (Q_2, V, q_{20}, F_2, \delta)$ be any FSA.

Consider the 1-turn n-PDA

$$M' = (Q_1 \times Q_2, V, [q_{10}, q_{20}], F_1 \times F_2, (\Gamma_1, \delta'_1, Z_1), \dots, (\Gamma_n, \delta'_n, Z_n))$$

The rules of M' are given by:

(1) If $(q'_1, \gamma) \in \delta_j(q_1, \lambda, X), q_1, q'_1 \in Q_1, \gamma \in \Gamma_j^*, X \in \Gamma_j, 1 \leq j \leq m$, then $([q'_1, q_2], \gamma) \in \delta_j([q_1, q_2], \lambda, X)$ for all $q_2 \in Q_2$.

(2) If $(q'_1, \gamma) \in \delta_j(q_1, a, X)$ and $q'_2 \in \delta(q_2, a), q_1, q'_1 \in Q_1, q_2, q'_2 \in Q_2, a \in V, \gamma \in \Gamma_j^*, X \in \Gamma_j, 1 \leq j \leq m$, then $([q'_1, q'_2], \gamma) \in \delta_j([q_1, q_2], a, X)$.

It can easily be seen by this construction that

$$L(M') = L(M) \cap L(M_2).$$

□

For two morphisms $h_1, h_2 : V_1^* \to V_2^*$, we define the equality set by

$$EQ(h_1, h_2) = \{x \in V_1^* | h_1(x) = h_2(x)\}$$

The following theorem gives a representation of RE in terms of equality sets [7, 9].

Theorem 16 *Each language $L \in RE$, $L \subseteq T^*$, can be written in the form $L = pr_T(EQ(h_1, h_2) \cap R)$, where $R \subseteq V_1^*$ is a regular language and $h_1, h_2 : V_1^* \to V_2^*$ are two λ - free morphisms, $T \subseteq V_1$.*

The following theorem shows that equality sets are accepted by 1-turn 2-PDAs.

Theorem 17 *For any 2 morphisms $h_1, h_2 : V_1 \to V_2$, there exists a 1-turn 2-PDA M such that $L(M) = EQ(h_1, h_2)$.*

Proof. Construct the 1-turn 2-PDA

$$M = (Q, V_1, q_0, \{q_f\}, (V_2 \cup \{Z\}, \delta_1, Z), (V_2 \cup \{Z\}, \delta_2, Z))$$
where $Q = \{q_0, q_1, q_f\} \cup \{[a] \mid a \in V_1\} \cup \{[X'] \mid X \in V_2 \cup \{Z\}\}$

The rules of M are given as follows:

$$\begin{array}{lcl}
(r1) & : & \delta_1(q_0, a, X) = \{[a], h_1(a)X)\}, a \in V_1, X \in V_2 \cup \{Z\} \\
(r2) & : & \delta_2([a], \lambda, X) = \{q_0, h_2(a)X)\}, a \in V_1, X \in V_2 \cup \{Z\} \\
(r3) & : & \delta_1(q_0, \lambda, X) = \{[X'], \lambda\}, X \in V_2 \cup \{Z\} \\
(r4) & : & \delta_1(q_1, \lambda, X) = \{[X'], \lambda\}, X \in V_2 \cup \{Z\} \\
(r5) & : & \delta_2([X'], \lambda, X) = \{q_1, \lambda\}, X \in V_2 \\
(r6) & : & \delta_2([Z'], \lambda, Z) = \{q_f, \lambda\}
\end{array}$$

On reading the input string $w(\in V_1^*)$, rules $(r1), (r2)$ push $h_1(w)$ and $h_2(w)$ onto the stacks of components 1 and 2 respectively. Rules $(r3), (r4), (r5)$ match the strings $h_1(w)$ and $h_2(w)$. On a perfect match, the system enters the final state q_f (rule $(r6)$). Thus we have

$$L(M) = EQ(h_1, h_2)$$

□

This theorem coupled with the characterization of RE by equality sets (Theorem 16) and the closure properties under morphism and intersection with regular sets (Theorem 15) proves that the family of languages accepted by 1-turn 2-PDA includes the whole of RE. We thus have,

Theorem 18 *For each $L \in RE$, there exists a 1-turn 2-PDA M such that $L(M) = L$ and conversely, for each 1-turn 2-PDA M, we have $L(M) \in RE$.*

Proof. The first statement in the theorem follows from the discussion above while the converse is a consequence of the Church-Turing Hypothesis. □

6. Conclusions

In this paper, we define the notion of distributed automata in the sequential sense and analyze the increase in acceptance power for FSA and PDA in different modes like t-mode, *-mode, $= k$-mode, $\leq k$-mode, and $\geq k$-mode. In the case of FSA, we prove that the distributed FSAs are no more powerful than the "centralized" FSAs in all of the above modes. However in the case of PDAs, we show that a distributed system with just two components accepts the family RE. In the case of distributed PDA's we also analyze the acceptance power in each of the above modes and we prove they are all equivalent. We also consider a restricted version of distributed PDA called k-turn distributed PDA and show that 1-turn 2-PDA are as powerful as Turing Machines.

In [1] Cooperated Distributed Grammars are considered. There it is found that the different modes of derivations are not equal in power, where as when we define automata, the different modes of acceptance have equivalent power.

There are several extensions worth pursuing. It would be interesting to study the acceptance power of the distributed PDA in different modes by restricting the system to be deterministic. The correspondence between distributed grammars and automata can be investigated. It would also be worthwhile to probe into the complexity(time and space) of these systems and also parallel communicating distributed automata.

References

1. J. Dassow, Gh. Paun and G. Rozenberg, “Grammar Systems”, chapter 4 in Vol 2 of *Handbook of Formal Languages*, springer-verlag, Hieldelberg, pp. 155-213, 1997.
2. E. Csuhaj Varju and J. Dassow, “On cooperating Distributed Grammar Systems”, *J. Inform. Process. Cybern., EIK*,26, pp. 49-63, 1990.
3. R. Meersman, G. Rozenberg and D. Vermeir, “Cooperating Grammar Systems”, *Proc. MFCS ’78,LNCS 64,* Springer-Verlag,Berlin, pp. 364-374, 1978.
4. Gh. Paun, L. Santaen, Parallel Communicating Grammar Systems: regular case, Ann. Univ. Buc., *Matem-Inform. series, Vol.38,* 1989, no. 2, 55-63.
5. E. Csuhaj-Varju, J.Dassow, J.Kleeman and Gh. Paun, *Grammar Systems. A Grammatical Approach to Distribution and cooperation,* Gordon and Breach, London, 1994.
6. G.Rozenberg and A.Salomaa, eds, *Handbook of Formal Languages*, 3 Volumes, Springer-Verlag, Hiedelberg, 1997.
7. A.Salomaa, *Jewels of Formal Language Theory*, Computer Science Press, Rockville, Maryland, 1981.
8. J.E.Hopcroft and J.D.Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 1979.
9. K. Cullik II, “A purely homomorphic characterization of recursively enumerable sets”, *Journal of the ACM*, 26(1979), 345-350.