Distributed-Automata and Simple Test Tube Systems

A Project Report
Submitted in partial fulfillment of the requirements of the degree of

Bachelor of Technology
in
Computer Science and Engineering

by

Prahladh Harsha

under the guidance of

Prof. Kamala Krithivasan

Department of Computer Science and Engineering
Indian Institute of Technology, Madras

1998

Certificate

This is to certify that the thesis entitled **Distributed-Automata and Simple Test Tube Systems** is a bona-fide record of the work done by **Prahladh Harsha** in the Department of Computer Science and Engineering, Indian Institute of Technology, Madras, under my guidance and supervision, in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering.

Place: Madras [Prof. Kamala Krithivasan]

Date:

Acknowledgements

I would like to express my sincere thanks to my guide, Prof Kamala Krithivasan for her constant support and advice. I am grateful to her for her encouragement which especially helped me overcome my initial hesitation before beginning this project and also get over my periods of slackness. I thank her for her excellent handling of the course *Theoretical Foundations of Computer Science-2*, the course which prompted me to choose the field, Formal Languages and Automata Theory for my project.

I would also like to express my thanks to Prof S.V.Raghavan, Head of the Department of Computer Science, Madras for letting me use the facilities, but for which this project would have been impossible. I am thankful to Prof C. Pandurangan for making the Theory Lab a wonderful place to work in. I am grateful to all the members of the faculty of the Department of Computer Science for their guidance. I am indebted to Dr. D.Janakiram, my Faculty Advisor for always being there whenever I needed help or advice. I would like to make a special mention of the Library, both the Departmental Library and the Theory Lab Library for their excellent facilities.

I also thank my batch-mates, the class of 98, my hostelmates, my lab-associates and other friends for their company, over the last 4 years of my stay in IIT, Madras, which helped me get over my occasional blues. I am greatly indebted to all others who have been either directly or indirectly involved in the completion of this project.

Abstract

In classical Formal Language Theory, "centralised" grammar systems and automata were the standard models for computation devices. With the rapid growth of computer networks, distributed databases, etc, distributed computation has begun to play a major role in modern computer science. The theory of grammar systems and distributed-automata was thus developed as a model for distributed computation.

In this thesis, we introduce the notion of distributed-automata. Distributed-Automata are a group of automata working in unison to accept one language. We build the theory of distributed-automata for specific automata like FSA, PDA and k-turn PDA. We then analyse the acceptance capacity of each of these automata. We present proofs that distributed FSA do not have any additional power over "centralised" FSA while both distributed PDAs and k-turn PDAs with just two components working together are as powerful as Turing Machines.

We then extend the definition of test tube systems to simple H systems and analyse their generative capacity. Distributed simple H systems are a group of simple H systems working in parallel with an architecture similar to that of PC grammar systems. Though simple H systems generate only regular languages, we show that when these systems work in parallel, with the aid of morphisms and intersection with regular sets, they are able to characterise all context-free languages.

Contents

Chapter 1

Introduction

1.1 Distributed Grammar Systems and Automata

In classical formal language and automata theory, grammars and automata were modeling classical computing devices. Such devices were "centralised" - the computation was 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. Analysing such computations in computer networks, distributed databases, etc., leads to notions such as distribution, parallelism, concurrency and communication. The theory of grammar systems and distributed-automata is thus developed as a model for distributed computation where these notions can be defined and analysed.

A grammar system is a *set* of grammars, working in unison, according to a specified protocol, to generate *one* language. Similarly distributed-automata is a *set* of automata, working together, according to specific rules, to accept *one* language. There are many reasons to consider such a generative (or acceptance) mechanism: to model distribution, to increase the generative (acceptance) power, to decrease the (descriptional) complexity, ... The crucial element in these systems is the protocol of cooperation. The theory of grammar systems may be seen as the "grammatical" theory of cooperation protocols.

One can broadly distinguish two basic classes of grammar systems: *sequential* and *parallel*. A *cooperating distributed* (CD) grammar system is a sequential system. In this system all the component grammars have a common sentential form. Initially, this is a common axiom. At each moment only one grammar is active - it rewrites the current sentential form, The matters such as which component grammar can become active at a given moment, and when an active grammar becomes inactive leaving the current sentential form to other component grammars is decided by the cooperation protocol. In a *parallel communicating* (PC) grammar system, each component has its own sentential form. Within each time unit (there is a clock common to all component grammars) each component applies a rule, rewriting its own sentential form. The key feature of a PC grammar system is its "communication through queries" mechanism.

In this report, the notion of CD grammar systems is extended to automata. A set of automata working in unison is considered and the increase in acceptance power is analysed. Then, distributed simple splicing systems are discussed whose architecture is similar to that of PC grammar systems.

The rest of this report is organised as follows: Section 1.1 gives a literature survey of the work in this area. In the other two sections of this chapter, we present the Formal language prerequisites and give a brief introduction to CD and PC grammar systems. Chapter 2 deals with Distributed-Automata. In Section 2.2, we give the formal definition of distributed FSA and show that its acceptance capacity is similar to that of FSAs. In Section 2.3, we proceed to define Distributed PDA. In Section 2.4, we discuss a restricted form of distributed PDAs - k-turn distributed PDAs, its properties and show that it is as powerful as a Turing Machine. Chapter 3 deals with simple test-tube systems. In the first three sections of this chapter, existing definitions and properties of splicing systems, simple H systems and Test tube systems are presented. In Section 3.4, we define Simple test-tube systems and present a proof that under morphisms and intersection with regular sets, the family of languages generated by this system includes CF.

1.2 Related Work

A comprehensive treatment of grammar systems and a survey of the recent developments in this area can be found in [7]. The study of CD grammar systems was initiated in [1]. The explicit notion of cooperating grammar systems was introduced in [11]. [2] introduces a problem solving architecture similar to PC grammar systems. This also discusses many relationships of CD and PC grammar systems to issues related to distribution, cooperation, parallelism in artificial intelligence, cognitive psychology, robotics, complex systems study etc. [6] and [18] give information about the initial development of CD and PC grammar systems.

The concept of *splicing* and the definitions of *H systems* were introduced in [8]. The regularity of H systems has been proved in [5] and [15], while *Extended H systems* have shown to be computationally complete in [13]. *Simple H systems* were first analysed in [10]. The extension of distributed grammar systems to splicing systems with an architecture similar to that of PC grammar systems has been discussed in [3] and [12].

1.3 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.*. A morphism $h : V^* \rightarrow U^*$, for arbitrary alphabets U, V, is called a *coding* if $h(a) \in U$ for each $a \in V$, and a *weak coding* if $h(a) \in U \cup \{\lambda\}$, for each $a \in V$.

FIN, REG, LIN, CF, CS and RE denote the families of finite, regular, linear, context-free, context-sensitive, and recursively enumerable languages respectively.

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 occurences of non-terminals. A language is said to be non-terminal bounded if there exists a non-terminal bounded grammar that generates it.

A *finite state automaton* (FSA) is a system $M = (Q, V, q_o, F, \delta)$ where Q is the finite set of states, V is the alphabet, $q_o (\in Q)$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta : Q \times V \rightarrow P_f(Q)$ is the transition mapping. ($P_f(X)$ denotes the set of finite subsets of X).

The transition mapping defines the transitions in the following manner where $q, q' \in Q, w \in V^*$ and $a \in V$.

$$(q, aw) \vdash (q', w) \text{ iff } q' \in \delta(q, a)$$

Let $\vdash^*$ denote the reflexive and transitive closure of $\vdash$. The language accepted by the FSA is defined as follows:

$$L(M) = \{w \in V^* | (q_o, w) \vdash^* (q_f, \lambda) \text{ for some } q_f \in F\}$$

A *push down automata* (PDA) is a system $M = (Q, V, \Gamma, q_o, Z_o, F, \delta)$ where Q is the finite set of states, V is the input alphabet, Γ is the stack alphabet, $q_o (\in Q)$ is the initial state, $Z_o (\in \Gamma)$ is the start stack symbol, $F \subseteq Q$ is the set of final states and $\delta : Q \times (V \cup \{\lambda\}) \times \Gamma \rightarrow P_f(Q \times \Gamma^*)$ is the transition mapping.

The transition mapping defines the transitions in the following manner where $q, q' \in Q; w \in V^*; \alpha, \beta \in \Gamma^*; Z \in \Gamma$ and $a \in V$.

$$(q, aw, Z\alpha) \vdash (q', w, \beta\alpha) \text{ iff } (q', \beta) \in \delta(q, a, Z)$$

Let $\vdash^*$ denote the reflexive and transitive closure of $\vdash$. The language accepted by the PDA is defined as following:

$L(M) = \{w \in V^* | (q_o, w, Z_o) \vdash^* (q_f, \lambda, \gamma)$ for some $q_f \in F$ and $\gamma \in \Gamma^*\}$

A PDA is said to perform a turn if the stack size reduces (ie., $(q, aw, \alpha) \vdash (q', w, \beta)$ and $|\alpha| > |\beta|$ where $q, q' \in Q; w \in V^*; a \in V; \alpha, \beta \in \Gamma^*$) Under the above definition of PDAs, there is no upper bound on the number of turns the PDA can perform in order to accept a string. 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 denoted by the symbol *k-turn PDA*. One can easily observe that the language accepted by 1-turn PDAs is the set of linear languages(LIN). It is known that the family of languages that are accepted by k-turn PDA, $k \geq 1$ are the non-terminal bounded languages.

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

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

Proofs of the following representation of RE can be found in [4] and [17].

Theorem 1 *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, $T \subseteq V_1$.*

1.4 CD and PC Grammar systems

1.4.1 CD grammar systems

As mentioned earlier, CD grammar systems are a sequential set of component grammars having a single sentential form. At any instant, one of the components rewrites this sentential form and the decision regarding which component is the active one is made according to a cooperation protocol. The formal definition is as follows:

Definition 1 *A cooperating distributed (CD) grammar system of degree* $n, n \geq 1$*, is a construct*

$$\Gamma = (N, T, S, P_1, \ldots, P_n),$$

where N, T *are disjoint alphabets,* $S \in N$ *and* $P_1, \ldots, P_n$ *are finite sets of rewriting rules over* $N \cup T$*.*

The elements of N *are nonterminals, those of* T *are terminals;* $P_1, \ldots, P_n$ *are called components of the system.*

There are various modes of derivation which define the cooperation protocol possible between the component grammars, such as (i) each grammar works for the maximum extent possible on the sentential form, (ii) each grammar works arbitrarily long on the sentential form, (iii) each grammar works either for exactly (or atmost or atleast) k steps on the sentential form ,etc. We present here only the definition of the first mode of derivation. The others can be suitably defined.

Definition 2 *For each* $i, 1 \leq i \leq n$*, the terminating derivation by the i-th component, denoted by* $\Rightarrow^t_{P_i}$*, is defined by*

$$x \Rightarrow^t_{P_i} y \textit{ iff } x \Rightarrow_{P_i} y \textit{ and there is no } z \in V^* \textit{ with } y \Rightarrow_{P_i} z$$

Then the language generated by the CD system working in this mode is

Definition 3 *The language generated by CD grammar system* $\Gamma == (N, T, S, P_1, \ldots, P_n)$ *in the terminating derivation mode is*

$$L_t(\Gamma) = \{w \in T^* | \quad S \Rightarrow^t_{P_{i_1}} w_1 \Rightarrow^t_{P_{i_2}} \ldots \Rightarrow^t_{P_{i_m}} w_m = w, \\ m \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq m\}$$

It is easily seen that for CD grammar systems working in any of the modes mentioned above and having regular, linear, context-sensitive or type-0 components, the generative power does not change. However CD grammar systems with context-free components do generate languages not in CF.

1.4.2 PC grammar systems

PC grammar systems are a set of component grammars working in parallel. Each of these components works on its own sentential form, rewriting the form at each global time-unit. The components communicate through a queuing mechanism and the strings generated by component 1 are said to be the strings generated by the system.

Definition 4 *A parallel communicating (PC) grammar system of degree* $n, n \geq 1$, *is a* $(n+3)$*-tuple*

$$\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n))$$

where N *is a nonterminal alphabet,* T *is a terminal alphabet,* $K = \{Q_1, \ldots, Q_n\}$ *(the sets* N, K, T *are mutually disjoint),* P_i *is a finite set of rewriting rules over* $N \cup K \cup T$ *and* $S_i \in N$ *for all* $1 \leq i \leq n$.

Let $V_\Gamma = N \cup K \cup T$. The sets $P_i, 1 \leq i \leq n$ are called the *components* of the system, and the elements $Q_1, \ldots, Q_n$ of K are called the *query symbols.* The following definition tells us how the sentential forms develop in each of the components and how the query mechanism is carried out.

Definition 5 *Given a PC grammar system* $\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n))$ *for two n-tuples* $(x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n)$ *with* $x_i, y_i \in V_\Gamma^*, 1 \leq i \leq n$, *where* $x_i \notin T^*$, *we write* $(x_1, x_2, \ldots, x_n) \Rightarrow (y_1, y_2, \ldots, y_n)$ *if one of the following conditions hold.*

(i) For each $i, 1 \leq i \leq n, |x_i|_K = 0, 1 \leq i \leq n$, *and for each* $i, 1 \leq i \leq n$, *we have either* $x_i \Rightarrow y_i$ *by a rule in* P_i *or* $x_i = y_i \in T^*$.

(ii)There is $i, 1 \leq i \leq n$, *such that* $|x_i|_K > 0$. *Let, for each such* $i, x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}, t \geq 1$, *for* $z_j \in (N \cup T)^*, 1 \leq j \leq t+1$. *If* $|x_{i_j}|_K = 0$, *for all* $j, 1 \leq j \leq t$, *then* $y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$ *and* $y_{i_j} = S_{i_j}, 1 \leq j \leq t$. *If for some* $j, 1 \leq j \leq t, |x_{i_j}|_K \neq 0$, *then* $y_i = x_i$. *For all* $i, 1 \leq i \leq n$, *such that* y_i *is not specified above, we have* $y_i = x_i$.

And the language generated by this grammar system is:

Definition 6 *The language generated by a PC grammar system* Γ *as mentioned above is*

$$L(\Gamma) = \{x \in T^* \mid (S_1, \ldots, S_n) \Rightarrow^* (x, \alpha_2, \ldots, \alpha_n), \alpha_i \in V_\Gamma^*, 2 \leq i \leq n\}$$

The component P_1 is called the *master* of the system. In the case of PC grammar systems, an increase is noticed in the generative power even when the component grammars are regular, context-free or context-sensitive.

Chapter 2

Distributed-Automata

2.1 Introduction

Distributed-Automata (also called n-*automata*) is a set of n component automata working together, which process the tape containing the input string in a sequential manner. These component automata share a common state space. One of these component automata is designated as the initial automaton which begins processing the tape. A set of states called *transition states* is associated with each of these components. Whenever a transition state is reached, a non-deterministic jump from the present automata to any of the component automata (including the present one) can be performed. The formal definition of the automata system will be given depending on the type of the component automata (FSA, PDA, . . .) employed.

2.2 Distributed FSA

n- *finite state automata* (n-FSA) is a set of n (or less) number of component FSAs working in unison processing an input string. The formal definition is as follows:

Definition 7 *An n-FSA is a* 6*-tuple* $\Gamma = (Q, V, \Delta, q_0, F, Q_T)$ *of degree* $m, m \leq n$ *where*

Q	*is a finite set of states*
V	*is a finite set of alphabets*
Δ	*is a m-tuple* $(\delta_1, \ldots, \delta_m)$ *of state transition functions*
	where each $\delta_i : Q \times (V \cup \{\lambda\}) \to P_f(Q), 1 \leq i \leq m$
$q_0 \in Q$	*is the initial state*
$F \subseteq Q$	*is the set of final acce Distributed-Automata (also called* n-automata*) is a set of* n *comp*
Q_T	*is a m-tuple* $(Q_1, \ldots, Q_m)$ *of transition states*
	where each $Q_i \subseteq Q, 1 \leq i \leq m.$

Each of the component FSAs of the n-FSA is of the form $M_i = (Q, V, \delta_i, q_0, F), 1 \leq i \leq m$

Initially the component automaton M_1 begins processing the string as if it were a "stand-alone" FSA. On arriving at any transition state ($q \in Q_1$), a non-deterministic choice is made among the m-component FSAs and the processing of the string is continued according to the transition function of that component. When a transition state in this component is reached, once again a non-deterministic jump is performed and the processing is continued.

Definition 8 *The instantaneous description of the n-FSA (ID) is given by a* 3*-tuple* (q, w, i)*, where* $q \in Q, w \in V^*, 1 \leq i \leq m.$

The transitions between the IDs is defined as follows:

$$(q, aw, i) \vdash (q', w, i) \quad \text{iff} \quad q' \in \delta_i(q, a) \text{ where } q, q' \in Q,$$
$$a \in V \cup \{\lambda\}, w \in V^*, 1 \leq i \leq m$$

$$(q, w, i) \vdash (q, w, j) \quad \text{iff} \quad q \in Q_i \text{ where } q \in Q, w \in V^*,$$
$$1 \leq i, j \leq m$$

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

Definition 9 *The language accepted by the n-FSA $\Gamma = (Q, V, \Delta, q_0, F, Q_T)$ is defined by*

$$L(\Gamma) = \{w \in V^* \mid (q_0, w, 1) \vdash^* (q_f, \lambda, i) \textit{ for some } q_f \in F, 1 \leq i \leq m\}$$

If all the states can perform as transition states (ie., $Q_i = Q, 1 \leq i \leq m$), then the term Q_T is not explicitly mentioned in the definition of the n-FSA. If n (ie., the upper bound on the degree of the n-FSA) is not explicitly mentioned, then the system is called *distributed FSA* and is denoted by the symbol $*$-FSA.

One can easily verify that having an n-FSA does not increase the acceptance capacity of the FSA as is seen from the following proof.

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

Proof

Let $\Gamma = (Q, V, \Delta, q_0, F, Q_T)$ be a n-FSA of degree $m, m \leq n$
where $\Delta = (\delta_1, \ldots, \delta_m)$ and $Q_T = (Q_1, \ldots, Q_m)$

Consider the FSA $M = (Q', V, \delta, q_0', F)$ where
$Q' = \{[q, i] \mid q \in Q, 1 \leq i \leq m\}$
$q_0 = [q_0, 1]$
$F' = \{[q_f, i] \mid q_f \in F, 1 \leq i \leq m\}$
δ contains the following transitions and nothing more
(i) For each $q_k \in \delta_i(q_j, a), q_j, q_k \in Q, 1 \leq i \leq m, a \in V \cup \{\lambda\}$
$[q_k, i] \in \delta([q_j, i], a)$
(ii) For each $q \in Q_i, 1 \leq i \leq m$ $[q, j] \in \delta([q, i], \lambda), 1 \leq j \leq m$

One can easily see that $L(M) = L(\Gamma)$

Thus, $L(\Gamma) \in REG$ $\square$

2.3 Distributed PDA

n- push down automata (n-PDA) is a set of n (or less) number of component PDAs working in unison processing an input string. It is to be noted that each of the component PDAs has its own stack. It is the presence of these multiple stack in n-PDA which imparts it its additional acceptance power.The formal definition is as follows:

Definition 10 *An n-PDA is a 8-tuple* $T = (Q, V, \Gamma, \Delta, q_0, , Z, F, Q_T)$ *of degree* $m, m \leq n$ *where*

Q	*is a finite set of states*
V	*is a finite set of input alphabet*
Γ	*is a m tuple* $(\Gamma_1, \ldots, \Gamma_m)$
	where each $\Gamma_i, 1 \leq i \leq m$ *is a finite set of stack alphabet*
Δ	*is a m-tuple* $(\delta_1, \ldots, \delta_m)$ *of state transition functions*
	where each $\delta_i : Q \times (V \cup \{\lambda\}) \times \Gamma \to P_f(Q \times \Gamma^*), 1 \leq i \leq m$
$q_0 \in Q$	*is the initial state*
Z	*is a m tuple* $(Z_1, \ldots, Z_m)$
	where each $Z_i \in \Gamma_i, 1 \leq i \leq m$ *is the start symbol of stack i*
$F \subseteq Q$	*is the set of final accepting states*
Q_T	*is a n-tuple* $(Q_1, \ldots, Q_m)$ *of transition states*
	where each $Q_i \subseteq Q, 1 \leq i \leq m$

Each of the component PDAs of the n-PDA is of the form $M_i = (Q, V, \Gamma_i, \delta_i, q_0, Z_i, F), 1 \leq i \leq m$. The n-PDA can also be denoted as $T = (Q, V, q_0, F, Q_T, (\Gamma_1, \delta_1, Z_1), \ldots (\Gamma_m, \delta_m, Z_m))$

The transitions and instantaneous descriptions of the systems are extensions of those of the normal PDA and are defined in a manner similar to those of the n-FSA.

Definition 11 *The instantaneous description of the n-PDA of degree m is a* $(m+3)$*-tuple* $(q, w, \alpha_1, \ldots, \alpha_m, i)$*, where* $q \in Q, w \in V^*, \alpha_k \in \Gamma^*, 1 \leq i, k \leq m$

The transitions between the IDs is defined as follows:

$$\begin{array}{rcl} (q, aw, \alpha_1, \ldots, X\alpha_i, \ldots, \alpha_m, i) & \vdash & (q', w, \alpha_1, \ldots, \beta\alpha_i, \ldots, \alpha_m, i) \\ & \text{iff} & (q', \beta) \in \delta_i(q, a, X) \\ & & q, q' \in Q, a \in V \cup \{\lambda\}, w \in V^*, 1 \leq i \leq m \\ & & \alpha_1, \ldots, \alpha_m, \beta \in \Gamma^*, X \in \Gamma \\ (q, w, \alpha_1, \ldots, \alpha_m, i) & \vdash & (q, w, \alpha_1, \ldots, \alpha_m, i) \\ & \text{iff} & q \in Q_i, q \in Q, w \in V^*, 1 \leq i \leq m \\ & & \alpha_1, \ldots, \alpha_m \in \Gamma^* \end{array}$$

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

Definition 12 *The language accepted by the n-FSA* $T = (Q, V, \Gamma, \Delta, q_0, , Z, F, Q_T)$ *is defined by*

$$\begin{array}{rl} L(T) = \{w \in V^* \mid & (q_0, w, Z_1, \ldots, Z_m, 1) \vdash^* (q_f, \lambda, \alpha_1, \ldots, \alpha_m, i) \\ & \textit{for some } q_f \in F, \alpha_1, \ldots, \alpha_m \in \Gamma^*, 1 \leq i \leq m\} \end{array}$$

As in the case of n-FSA, if all the states can perform as transition states (ie., $Q_i = Q, 1 \leq i \leq m$), then the term Q_T is not explicitly mentioned in the definition of n-PDA. If n (ie., the upper bound on the degree of the n-PDA) is not explicitly mentioned, then the system is called *distributed PDA* and is denoted by the symbol $*$-PDA.

We are aware that a two-stack machine can simulate a Turing Machine (TM). Hence a 2-PDA is as powerful as a TM. We present below a formal proof of the same.

Theorem 3 *For any $L \in RE$, there is a 2-PDA T such that $L = L(T)$.*

Proof

Let $M = (Q, V, \Gamma, \delta, q_0, B, F)$ be a TM such that $L(M) = L$. Let R, L, L' be symbols such that $R, L, L' \notin V$. We shall also assume that $Z_1, Z_2 \notin \Gamma$ and $s_0 \notin Q$.

Consider a 2-PDA $T = (Q', V, s_0, F', Q_T, (\Gamma_1 \cup \{Z_1\}, \delta_1, Z_1), (\Gamma_1 \cup \{Z_1\}, \delta_1, Z_1))$ where

$$\begin{aligned}
Q' &= \{s_0, s_1, s_2\} \cup \{[A], [A'] \mid A \in \Gamma\} \cup \\
&\quad \{q', [q, A], [q, R], [q, LA], [q, L'A] \mid q \in Q, A \in \Gamma\} \\
Q_T &= (Q_1, Q_2) \text{ where} \\
&\quad Q_1 = \{s_2\} \cup \{[A], [A'] \mid A \in \Gamma\} \cup \\
&\quad \{[q, R], [q, L'A] \mid q \in Q, A \in \Gamma\} \\
&\quad Q_2 = \{s_1\} \cup \{q', [q, A] \mid q \in Q, A \in \Gamma\} \\
F' &= \{q'_f \mid q_f \in F\}
\end{aligned}$$

δ_1 and δ_2 are as specified below

$(R_1) : \delta_1(s_0, a, A) = \{(s_0, aA)\}, a \in V, A \in \Gamma \cup \{Z_1\}$

$(R_2) : \delta_1(s_0, \lambda, A) = \{(s_1, A)\}, A \in \Gamma \cup \{Z_1\}$

$(T_1) : \delta_1(s_1, \lambda, A) = \{([A], \lambda)\}, A \in \Gamma$

$(T_2) : \delta_2([A], \lambda, X) = \{(s, AX)\}, A \in \Gamma, X \in \Gamma \cup \{Z_2\}$

$(T_3) : \delta_1(s_1, \lambda, Z_1) = \{(s_2, Z_1)\}$

$(T_{4a}) : \delta_2(s_2, \lambda, A) = \{([A'], \lambda)\}, A \in \Gamma$

$(T_{4b}) : \delta_2(s_2, \lambda, Z_2) = \{([B'], \lambda)\}$

$(T_5) : \delta_1([A'], \lambda, Z_1) = \{q'_0, AZ_1)\}, A \in \Gamma$

For each rule $r^R \in \delta$ of the form $\delta(q_j, X) = (q_k, Y, R)$, we have the following rules for the 2-PDA.

$(r^R_1) : \delta_1(q'_j, \lambda, X) = \{([q_k, R], Y)\}$

$(r_{2a}^R) : \delta_2([q_k, R], \lambda, A) = \{([q_k, A], \lambda)\}, A \in \Gamma,$

$(r_{2b}^R) : \delta_2([q_k, R], \lambda, Z) = \{([q_k, B], \lambda)\}$

$(r_3^R) : \delta_1([q_k, A], \lambda, X) = \{(q_k', AX), X \in \Gamma \cup \{Z_1\}$

For each rule $r^L \in \delta$ of the form $\delta(q_j, X) = (q_k, Y, L)$, we have the following rules for the 2-PDA.

$(r_1^L) : \delta_1(q_j', \lambda, X) = \{([q_k, LY], \lambda)\}$

$(r_{2a}^L) : \delta_1([q_k, LY], \lambda, X) = \{([q_k, L'Y], X)\}$

$(r_{2b}^L) : \delta_1([q_k, LY], \lambda, Z) = \{([q_k, L'Y], BZ)\}$

$(r_3^L) : \delta_2([q_k, L'Y], \lambda, X) = \{(q_k', YX)\}, X \in \Gamma_2 \cup \{Z_2\}$

The rules (R_1) and (R_2) read the input into the stack of PDA 1 (Note : If the input is not completely read, i.e., Rule (R_2) is performed before the entire string is read, then the string can never be read compeletely).

The rules (T_1) and (T_2) transfer the entire input to the stack of the 2^{nd} PDA. When the entire input is transfered completely, rule (T_3) is performed.

The topmost symbol on the stack of the PDA 1 corresponds to the symbol being scanned by the TM M.

Thereafter the transitions of the TM are carried out by the rules (r_p) or (r_l) as the case may be. Rules $(r_R2)(b)$ and $(r_L2)(b)$ are added so that when the TM reads beyond the already scanned portion of the tape, an extra B (blank symbol) is added to the right or the left of the tape as the case may be. In this manner each action of the TM is simualted by this machine. Hence it can be easily seen that

$$L(M) = L(T)$$

Thus proved. $\square$

2.4 Distributed k-turn PDA

2.4.1 Distributed k-turn PDA - Definition

We now consider a restricted version of the distributed PDA discussed in the previous section. In this restricted version of pushdown automata, the stacks of the component PDAs can perform atmost a finite 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. Throughout this section, we shall consider only those k-turn n-PDA whose transition state space is the entire state space. 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 an k-turn n-PDA Γ such that $L(\Gamma) = L$.*

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 PDAs (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 4 *For any k-turn *-PDA Γ_1, there exists a 1-turn *-PDA Γ_2 such that $L(\Gamma_1) = L(\Gamma_2)$.*

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

It is of interest to note that 1-turn *-PDA accept certain languages which are non context-free in nature.

Example 1 Consider

$$\begin{aligned}\Gamma_1 &= (\{q_1, \ldots, q_6\}, \{a, b, c\}, q_1, \{q_6\},\\ &\quad (\{Z, a, b, c\}, \delta_1, Z), (\{Z, a, b, c\}, \delta_2, Z))\end{aligned}$$

where

$(r1) : \delta_1(q_1, a, X) = \{(q_1, aX)\}, X \in \{Z, a\}$

$(r2) : \delta_1(q_1, b, a) = \{(q_2, \lambda)\}$

$(r3) : \delta_2(q_2, \lambda, X) = \{(q_3, bX)\}, X \in \{Z, b\}$

$(r4) : \delta_1(q_3, b, a) = \{(q_2, \lambda)\}$

$(r5) : \delta_1(q_3, c, Z) = \{(q_4, Z)\}$

$(r6) : \delta_2(q_4, \lambda, b) = \{(q_5, \lambda)\}$

$(r7) : \delta_2(q_5, c, b) = \{(q_5, \lambda)\}$

$(r8) : \delta_2(q_5, \lambda, Z) = \{(q_6, \lambda)\}$

In this automata, 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

$$L(\Gamma_1) = \{a^n b^n c^n \mid n \geq 1\}$$

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

Example 2 Consider

$$\begin{aligned}\Gamma_2 &= (\{q_{10}, q_{11}, q_{20}, q_{2a}, q_{2b}, q_f\}, \{a, b, c\}, q_{10}, \{q_f\},\\ &\quad (\{Z, a, b\}, \delta_1, Z), (\{Z, a, b\}, \delta_2, Z))\end{aligned}$$

where

$(r1) : \delta_1(q_{10}, p, X) = \{(q_{10}, pX)\}, X \in \{Z, a, b\}, p \in \{a, b\}$

$(r2) : \delta_1(q_{10}, c, a) = \{(q_{2a}, \lambda)\}$

$(r3) : \delta_1(q_{10}, c, b) = \{(q_{2b}, \lambda)\}$

$(r4) : \delta_2(q_{2a}, \lambda, X) = \{(q_{11}, aX)\}, X \in \{Z, a, b\}$

$(r5) : \delta_2(q_{2b}, \lambda, X) = \{(q_{11}, bX)\}, X \in \{Z, a, b\}$

$(r6) : \delta_1(q_{11}, \lambda, a) = \{(q_{2a}, \lambda)\}$

$(r7) : \delta_1(q_{11}, \lambda, b) = \{(q_{2b}, \lambda)\}$

$(r8) : \delta_1(q_{11}, \lambda, Z) = \{(q_{20}, Z)\}$

$(r9) : \delta_2(q_{20}, a, a) = \{(q_{20}, \lambda)\}$

$(r10) : \delta_2(q_{20}, b, b) = \{(q_{20}, \lambda)\}$

$(r11) : \delta_2(q_{20}, \lambda, Z) = \{(q_f, \lambda)\}$

In this automata, 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 stack 2 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 reminder 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 is entered and the string $wcw, w(\in \{a, b\}^*)$ is accepted. Thus

$$L(\Gamma_2) = \{wcw \mid w \in \{a, b\}^*\}$$

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

2.4.2 Properties of Distributed 1-turn PDA

In this subsection, we discuss a few of the closure properties of the family of languages accepted by 1-turn *-PDAs. We then present the equivalence of this family of languages with RE.

Theorem 5 *The family of languages accepted by* 1*-turn *-PDAs is closed under the following operations.*

(i) Union

(ii) Concatenation

(iii) Intersection

(iv) Morphism

(v) Intersection with Regular Sets

Proof

(i) **Union:** Let

$$\Gamma_1 = (Q_1, V, q_{10}, F_1, (\Gamma_{11}, \delta_{11}, Z_{11}), \ldots, (\Gamma_{1n_1}, \delta_{1n_1}, Z_{1n_1}))$$
$$\Gamma_2 = (Q_2, V, q_{20}, F_2, (\Gamma_{21}, \delta_{21}, Z_{21}), \ldots, (\Gamma_{2n_2}, \delta_{2n_2}, Z_{2n_2}))$$

be any two 1-turn *-PDAs. We can assume $Q_1 \cap Q_2 = \Phi$. Also assume $q_{00} \notin Q_1 \cup Q_2$. Consider the 1-turn *-PDA given by

$$\begin{aligned}\Gamma = (Q_1 \cup Q_2, V, q_{00}, F_1 \cup F_2, \quad & (\Gamma_{11}, \delta'_{11}, Z_{11}), \ldots, (\Gamma_{1n_1}, \delta'_{1n_1}, Z_{1n_1}),\\ & (\Gamma_{21}, \delta'_{21}, Z_{21}), \ldots, (\Gamma_{2n_2}, \delta'_{2n_2}, Z_{2n_2}))\end{aligned}$$

where

$$\begin{aligned}\delta'_{11} &= \delta_{11} \cup \{(q_{00}, \lambda, Z_{11}) \rightarrow \{(q_{10}, Z_{11}), (q_{20}, Z_{11})\}\\ \delta'_{ij} &= \delta_{ij} \text{ for other } i \text{ and } j.\end{aligned}$$

Γ is so constructed that both the automata systems Γ_1 and Γ_2 are simulated by it. The initial non-deterministic choice between the two automata Γ_1 and Γ_2 is made by the additional rules added to δ_{11}. We thus easily see that

$$L(\Gamma) = L(\Gamma_1) \cup L(\Gamma_2)$$

It is to be noted that $deg(\Gamma) = deg(\Gamma_1) + deg(\Gamma_2)$.

(ii) **Concatenation:** Let

$$\Gamma_1 = (Q_1, V, q_{10}, F_1, (\Gamma_{11}, \delta_{11}, Z_{11}), \ldots, (\Gamma_{1n_1}, \delta_{1n_1}, Z_{1n_1}))$$
$$\Gamma_2 = (Q_2, V, q_{20}, F_2, (\Gamma_{21}, \delta_{21}, Z_{21}), \ldots, (\Gamma_{2n_2}, \delta_{2n_2}, Z_{2n_2}))$$

be any two 1-turn *-PDAs. We can assume $Q_1 \cap Q_2 = \Phi$. Consider the 1-turn *-PDA given by

$$\begin{aligned}\Gamma = (Q_1 \cup Q_2, V, q_{10}, F_2, \quad & (\Gamma_{11}, \delta'_{11}, Z_{11}), \ldots, (\Gamma_{1n_1}, \delta'_{1n_1}, Z_{1n_1}), \\ & (\Gamma_{21}, \delta'_{21}, Z_{21}), \ldots, (\Gamma_{2n_2}, \delta'_{2n_2}, Z_{2n_2}))\end{aligned}$$

where

$$\begin{aligned}\delta'_{11} &= \delta_{11} \cup \{(q_f, \lambda, X) \rightarrow \{(q_{20}, X) \mid q_f \in F, X \in \Gamma_{11}\} \\ \delta'_{ij} &= \delta_{ij} \text{ for other } i \text{ and } j.\end{aligned}$$

In this case the construction of Γ is such that the initial processing of the input string (ie., processing of the substring w_1 of string $w = w_1w_2$) is performed as if the automata system were Γ_1. On reaching a final state in Γ_1, the additional rules added to δ_{11} enable a jump to the initial state of the system Γ_2. From this point onwards, the remaining substring (ie., substring w_2) is processed as if the automata system were Γ_2. We thus have

$$L(\Gamma) = L(\Gamma_1)L(\Gamma_2)$$

We again note that by this construction $deg(\Gamma) = deg(\Gamma_1) + deg(\Gamma_2)$.

(iii) **Intersection:** To prove closure under intersection, we give a brief description of the construction of the 1-turn *-PDA Γ, which accepts the intersection of the languages accepted by the two 1-turn *-PDAs Γ_1 and Γ_2, as the exact details of the construction are too cumbersome.

Γ has three more components than the sum of the components of both Γ_1 and Γ_2. Γ behaves in the following manner: the input string is initially pushed into one of the stacks of the three additional components. This string is then reversed and a copy of this reversed string is pushed into each of the stacks of the other two additional components. The reversed strings are in the proper order, ie., the input string being read left to right the tape coincides with the string read top to bottom in these stacks. Γ_1 processes the string in one of these stacks. If Γ_1 reaches a final state and the entire string in that stack is processed, then the initial state of Γ_2 is entered. Γ_2 then processes the string in the other stack in a manner similar to Γ_1. Thus Γ accepts only those strings which are accepted by both Γ_1 and Γ_2.

(iv) **Morphism:** Let

$\Gamma = (Q, V_1, q_0, F, (\Gamma_1, \delta_1, Z_1), \dots, (\Gamma_n, \delta_n, Z_n))$ be any 1-turn *-PDA of degree m and $h : V_1 \to V_2$ be any morphism.

Construct a 1-turn *-PDA of degree m as shown below:

$\Gamma' = (Q', V_2, q_0, F, (\Gamma_1, \delta'_1, Z_1), \dots, (\Gamma_n, \delta'_n, Z_n))$

where

$$Q' = Q \cup \{[q, a, X, a_i, j] \mid \quad |h(a)| > 1, h(a) = a_0 a_1 \dots a_n \in V_2^*, 0 \leq i < n, \\ q \in Q, a \in V_1, X \in \Gamma_j, 1 \leq j \leq m\}$$

The rules of Γ 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 $h(a) = a_0 a_1 \dots a_n \in V_2^*$

$([q, a, X, a_0, j], X) \in \delta'_j(q, a_0, X)$

$([q, a, X, a_i, j], X) \in \delta'_j([q, a, X, a_{i-1}, j], a_i, X), 1 \leq i < n$

$(q', \gamma) \in \delta'_j([q, a, X, a_{n-1}, j], a_i, X)$

Γ' is constructed in such a manner that Γ' responds to $h(w), (w \in V_1^*)$ in exactly the same way as Γ does to w. Thus we have

$$L(\Gamma') = h(L(\Gamma))$$

We note that by virtue of this construction $deg(\Gamma') = deg(\Gamma)$.

(v) **Intersection with Regular sets:** Though closure under intersection with regular sets immediately follows from (iii) and the observation that REG is a subset of the family of languages accepted by 1-turn *-PDAs, we give an an independent proof of this fact so as to show that in this case no extra components are required as is required for general intersection.

Let

$\Gamma = (Q_1, V, q_{10}, F_1, (\Gamma_1, \delta_1, Z_1), \ldots, (\Gamma_n, \delta_n, Z_n))$ be any 1-turn *-PDA of degree m and $M = (Q_2, V, q_{20}, F_2, \delta)$ be any FSA.

Consider the 1-turn *-PDA of degree m

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

The rules of Γ' are given by:

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

It can easily be seen by this construction that

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

□

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

Theorem 6 *For any 2 morphisms $h_1, h_2 : V_1 \to V_2$, there exists a 1-turn 2-PDA Γ such that $L(\Gamma) = EQ(h_1, h_2)$.*

Proof: Construct the 1-turn 2-PDA

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

The rules of Γ are given as follows:

$(r1) : \delta_1(q_0, a, X) = \{[a], h_1(a)X)\}, a \in V_1, X \in V_2 \cup \{Z\}$

$(r2) : \delta_2([a], \lambda, X) = \{q_0, h_2(a)X)\}, a \in V_1, X \in V_2 \cup \{Z\}$

$(r3) : \delta_1(q_0, \lambda, X) = \{[X'], \lambda\}, X \in V_2 \cup \{Z\}$

$(r4) : \delta_1(q_1, \lambda, X) = \{[X'], \lambda\}, X \in V_2 \cup \{Z\}$

$(r5) : \delta_2([X'], \lambda, X) = \{q_1, \lambda\}, X \in V_2$

$(r6) : \delta_2([Z'], \lambda, Z) = \{q_f, \lambda\}$

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 (Rule $(r6)$). Thus we have

$$L(\Gamma) = EQ(h_1, h_2)$$

This theorem coupled with characterisation of RE by equality sets (1) and the closure properties under homomorphism and intersection with regular sets (5) proves that the family of languages accepted by 1-turn 2-PDA includes the whole of RE. We thus have,

Theorem 7 *For each $L \in RE$, there exists a 1-turn 2-PDA Γ such that $L(\Gamma) = L$ and conversely, For each 1-turn 2-PDA Γ, we have $L(\Gamma) \in RE$.*

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

Chapter 3

Simple Test Tube Systems

3.1 Introduction

In this chapter, a symbol processing mechanism having the architecture of a PC grammar system with communication by command, but with the components being test tubes working as simple splicing schemes is presented. The communication is performed by redistributing the contents of the tubes according to a specific protocol. The increase in generative capacity of simple test tube systems over simple H systems is discussed.In the next section, the basic definitions (H systems, EH systems and Simple H systems) and notations employed are outlined. Section 3.3 deals with test tube systems. In Section 3.4, simple test-tube systems are introduced and the inclusion of CF in the family of languages accepted by these systems is proven.

3.2 Splicing Systems

3.2.1 Splicing - Definition

Definition 13 *A splicing rule (over an alphabet V) is a string $r = u_1\#u_2\$u_3\#u_4$, where $u_i, 1 \leq i \leq 4$, and $\#, \$$ are special symbols not in V. For such a rule r and* *the strings $x, y, z \in V^*$, the splicing operation is as given follows:*

$$\begin{aligned}(x, y) \Rightarrow_r z \quad \textit{iff} \quad & x = x_1u_1u_2x_2, y = y_1u_3u_4y_2 \\ & z = x_1u_1u_4y_2 \\ & \textit{for some } x_1, x_2, y_1, y_2 \in V^*\end{aligned}$$

x, y are then said to be *spliced* at the *sites* u_1u_2, u_3u_4 respectively to obtain the string z. The strings x, y are called the *terms* of splicing. When understood from context, the index r is omitted from $\Rightarrow_r$

A *splicing scheme* (or an *H scheme*) is a pair $\sigma = (V, R)$ where V is an alphabet and R is a set of splicing rules (over V). For a language $L \subseteq V^*$, the following are defined:

$$\sigma(L) = \{w \in V^* | (x, y) \Rightarrow_r w \text{ for } x, y \in L, r \in R\}$$

The following are also defined for the language L

$$\begin{aligned}\sigma^0 &= L \\ \sigma^{i+1}(L) &= \sigma^i(L) \cup \sigma(\sigma^i(L)) \\ \sigma^*(L) &= \cup_{i \geq 0} \sigma^i(L)\end{aligned}$$

Thus, $\sigma^*(L)$ is the smallest language containing L and closed under the splicing operation.

3.2.2 H systems and EH systems

Definition 14 *An* extended splicing system *is a quadruple*

$$\gamma = (V, T, A, R)$$

where V is an alphabet, $T \subseteq V$ (the terminal alphabet), $A \subseteq V^*$ the set of axioms and $R \subseteq V^*\#V^*\$V^*\#V^*$ the set of rules. The pair $\sigma = (V, R)$ is called the underlying H schema *of γ. The language generated by γ is defined by*

$$L(\gamma) = \sigma^*(A) \cap T^*$$

An H system $\gamma = (V, T, A, R)$ is said to be of *type* (F_1, F_2) for two families of languages F_1, F_2 if $A \in F_1, R \in F_2$. $EH(F_1, F_2)$ is used to denote the family of languages generated by extended H systems of type (F_1, F_2). An H system $\gamma = (V, T, A, R)$ with $V = T$ is said to be *non-extended;*, here the H system is denoted by $\gamma = (V, A, R)$. The family of languages generated by non-extended H systems of type (F_1, F_2) is denoted by $H(F_1, F_2)$. Obviously, $H(F_1, F_2) \subseteq EH(F_1, F_2)$.

The following results regarding H systems and EH systems can be found in [5], [15]; [14] and [13] respectively.

Theorem 8 *(i)* $H(FIN, FIN) \subseteq REG$.
(ii) $EH(FIN, FIN) = REG$
(iii)$EH(FIN, REG) = RE$

Thus these EH systems are as powerful as Turing Machine. However it is not realistic to deal with infinite number of rules even if they were a regular set of rules. For this purpose several variants were proposed. One such variant is the working of several these extended H systems in unison in a manner similar to PC grammar systems. These parallel Extended systems are called test-tube systems, which will be discussed in detail in the next section.

3.2.3 Simple H systems

Definition 15 *A simple H system is a triple*

$$\Gamma = (V, A, M)$$

where V *is an alphabet,* $A \subseteq V^*$ *is a finite set of axioms and* $M \subseteq V$

The elements of M are called *markers*. One can consider four ternary relations on the language V^*, corresponding to the splicing rules of the forms

$$a\#\$a\#, \#a\$\#a, a\#\$\#a, \#a\$a\#$$

where a is an arbitrary element of M. These four rules are respectively called splicing rules of types $(1,3),(2,4),(1,4),(2,3)$.

Clearly, rules of types $(1,3)$ and $(2,4)$ define the same operation: for $x,y,z \in V^*$ and $a \in M$ we obtain

$$\begin{aligned}(x,y) \Rightarrow^a_{(1,3)} \quad \text{iff} \quad & x = x_1 a x_2, y = y_1 a y_2, z = x_1 a y_2,\\ & \text{for some } x_1,x_2,y_1,y_2 \in V^*.\end{aligned}$$

For the other types, the splicing is defined as follows:

$$\begin{aligned}(x,y) \Rightarrow^a_{(1,4)} \quad \text{iff} \quad & x = x_1 a x_2, y = y_1 a y_2, z = x_1 a a y_2,\\ & \text{for some } x_1,x_2,y_1,y_2 \in V^*.\\ (x,y) \Rightarrow^a_{(2,3)} \quad \text{iff} \quad & x = x_1 a x_2, y = y_1 a y_2, z = x_1 y_2,\\ & \text{for some } x_1,x_2,y_1,y_2 \in V^*.\end{aligned}$$

Then for $L \subseteq V^*$ and $(i,j) \in \{(1,3),(1,4),(2,3)\}$, the following is defined

$$\sigma_{(i,j)}(L) = \{w \in V^* | (x,y) \Rightarrow^a_{(i,j)} w \text{ for some } x,y \in L, a \in M\}$$

$$\begin{aligned}\sigma^0_{(i,j)}(L) &= L\\ \sigma^{k+1}_{(i,j)}(L) &= \sigma^k_{(i,j)}(L) \cup \sigma_{(i,j)}(\sigma^k_{(i,j)}(L)), k \geq 0\\ \sigma^*_{(i,j)}(L) &= \cup_{k\geq 0}\sigma^k_{(i,j)}(L)\end{aligned}$$

Definition 16 *The language generated by* $\Gamma = (V,A,M)$ *in case* (i,j) *is defined by*

$$L_{(i,j)}(\Gamma) = \sigma^*_{(i,j)}(A)$$

As in all the cases mentioned, both the axiom set and the rule set is finite, it follows from [5] and [15] that for any simple H system Γ, the languages $L_{(i,j)}(\Gamma)$ are regular. It has been shown in [10] that each two of the three family of languages obtained in this way are incomparable.

3.3 Test Tube Systems

Definition 17 *A test tube (TT) system (of degree $n, n \geq 1$) is a construct*

$$\Gamma = (V, (A_1, R_1, V_1), \ldots, (A_n, R_n, V_n))$$

where V is an alphabet, $A_i \subseteq V^*, R_i \subseteq V^*\#V^*\$V^*\#V^*$ and $V_i \subseteq V$ for each $1 \leq i \leq n$

Each triple (A_i, R_i, V_i) is called a *component* of the system or a *tube.* A_i is the set of axioms of tube i, R_i is the set of splicing rules of the tube i and V_i is the *selector* of the tube i.

Let

$$B = V^* - \cup_{i=1}^{n} V_i^*$$

The pair $\sigma_i = (V, R_i)$ is the underlying H schema associated with the i-th component of the system.

An $n-$tuple $(L_1, \ldots, L_n)$, $L_i \subseteq V^*, 1 \leq i \leq n$, is called a *configuration* of the system; L_i is also called the *contents* of the i-th tube.

Definition 18 *For any two configurations $(L_1, \ldots, L_n), (L'_1, \ldots, L'_n)$, the following is defined*

$$\begin{aligned} (L_1, \ldots, L_n) \quad &\Rightarrow \quad (L'_1, \ldots, L'_n) \text{ iff for each } i, 1 \leq i \leq n \\ &L'_i = \left[\cup_{j=1}^{n}(\sigma_j^*(L_j) \cap V_i^*)\right] \cup (\sigma_i^*(L_i) \cap B) \end{aligned}$$

In other words, the contents of each tube is spliced according to the associated set of rules and the result is redistributed among the n tubes according to the selectors $V_1, \ldots V_n$. The part which cannot be redistributed remains in the tube.

Definition 19 *The language generated by the test tube system*
$\Gamma = (V, (A_1, R_1, V_1), \ldots, (A_n, R_n, V_n))$ *is*

$$L(\Gamma) = \{w \in V^* | \quad w \in L_1^{(t)} \text{ for some } (A_1, \ldots, A_n) \Rightarrow^* (L_1^{(t)} \ldots L_n^{(t)}), t \geq 0\}$$

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

Given two families of languages F_1, F_2, $TT_n(F_1, F_2)$ denotes the family of languages $L(\Gamma)$, for some $\Gamma = (V, (A_1, R_1, V_1), \ldots, (A_m, R_m, V_m))$ with $m \leq n, A_i \in F_1, R_i \in F_2$ for each $i, 1 \leq i \leq m$. Γ is then said to be of type (F_1, F_2). When n is not specified, Γ is said to belong to the family $TT_\infty(F_1, F_2)$.

The generative power of the splicing systems is increased when working in parallel as mentioned earlier. This is seen from the following results [3].

Theorem 9 *(i)* $TT_\infty(FIN, FIN) = RE$.
(ii) $TT_6(FIN, FIN)$ *contains non-recursive languages.*

3.4 Simple Test Tube Systems

In this section, we consider a set of simple H systems working in parallel. Each component of this system, referred to as a *Simple Test Tube* is a simple H system. As in simple H systems, simple test tubes can also work in any of the four modes(types) $(1,4), (1,3), (2,4), (2,3)$. The formal definition of simple test tube systems is given below.

Definition 20 *A simple test tube (or distributed simple H system) (STT) system (of degree* $n, n \geq 1$*) is a construct*

$$\Gamma = (V, (A_1, M_1, V_1), \ldots, (A_n, M_n, V_n))$$

where V *is an alphabet,* $A_i \subseteq V^*, M_i \subseteq V$ *and* $V_i \subseteq V$ *for each* $1 \leq i \leq n$

Each triple (A_i, R_i, V_i) is called a *component* of the system or a *simple tube.* A_i is the set of axioms of tube i, M_i is the set of markers of the tube i and V_i is the *selector* of the tube i.

The pair $\sigma_i = (V, A_i, M_i)$ is the underlying simple H schema associated with the i-th component of the system.

Definition 21 *For a STT system* $\Gamma == (V, (A_1, M_1, V_1), \ldots, (A_n, M_n, V_n))$*, depending on the type* $(1,4), (1,3), (2,3), (2,4)$*, we associate 4 different TT systems.*
(i) Type $(1,4) : \Gamma^{(1,4)} = (V, (A_1, R_1^{(1,4)}, V_1), \ldots, (A_n, R_n^{(1,4)}, V_n))$ *where each* $R_i^{(1,4)} = \{a\#\$\#a | a \in M_i\}, 1 \leq i \leq n$.
(i) Type $(1,3) : \Gamma^{(1,3)} = (V, (A_1, R_1^{(1,3)}, V_1), \ldots, (A_n, R_n^{(1,3)}, V_n))$ *where each* $R_i^{(1,3)} = \{a\#\$a\# | a \in M_i\}, 1 \leq i \leq n$.
(i) Type $(2,3) : \Gamma^{(2,3)} = (V, (A_1, R_1^{(2,3)}, V_1), \ldots, (A_n, R_n^{(2,3)}, V_n))$ *where each* $R_i^{(2,3)} = \{\#a\$a\# | a \in M_i\}, 1 \leq i \leq n$.
(i) Type $(2,4) : \Gamma^{(2,4)} = (V, (A_1, R_1^{(2,4)}, V_1), \ldots, (A_n, R_n^{(2,4)}, V_n))$ *where each* $R_i^{(2,4)} = \{\#a\$\#a | a \in M_i\}, 1 \leq i \leq n$.

Now that we have associated STT systems with TT systems, we can proceed with the actual working of the STT system.

Definition 22 *A STT system* $\Gamma = (V, (A_1, M_1, V_1), \ldots, (A_n, M_n, V_n))$ *working in type* $(i,j), (i,j) \in \{(1,4), (1,3), (2,3), (2,4)\}$ *is defined to be the TT system* $\Gamma^{(i,j)}$ *(as defined in (21)).*

Given a families of languages F, $STT_n^{(i,j)}(F)$ denotes the family of languages $L(\Gamma)$, for some STT system $\Gamma = (V, (A_1, M_1, V_1), \ldots, (A_m, M_m, V_m))$ working according to type (i,j) with $m \leq n, A_k \in F$, for each $k, 1 \leq k \leq m$ and $(i,j) \in \{(1,4), (1,3), (2,3), (2,4)\}$. Γ is then said to be of type (F). When n is not specified, Γ is said to belong to the family $STT_\infty^{(i,j)}(F)$.

It is interesting to note that though simple H systems do not generate languages beyond REG, distributed simple H systems do produce languages not in REG.

Theorem 10 $STT_4^{(2,3)}(FIN) - REG \neq \Phi$.

Proof Let Γ be the following STT system of type $(2,3)$.

$$\Gamma = (V, (A_1, M_1, V_1), \ldots (A_4, M_4, V_4))$$

where

$V = \{a, b, c, d, c', d'\}$

$A_1 = \{cabd, c'ac\}$

$M_1 = \{c\}$

$V_1 = \{a, b, c, d\}$

$A_2 = \{dbd'\}$

$M_2 = \{d\}$

$V_2 = \{a, b, c', d\}$

$A_3 = \{cc'\}$

$M_3 = \{c'\}$

$V_3 = \{a, b, c', d'\}$

$A_4 = \{d'd\}$

$M_4 = \{d'\}$

$V_4 = \{a, b, c, d'\}$

Let us consider the sequence of operations performed on the string of type ca^ib^id. We have such a string to start with namely, $cabd \in A_1$. In simple tube 1, $(c'ac, ca^ib^id) \Rightarrow c'a^{i+1}b^id$, which is filtered into simple tube 2. Tube 2 performs the following: $(c'a^{i+1}b^id, dbd') \Rightarrow c'a^{i+1}b^{i+1}d'$ which is then accepted by tube 3. $(cc', c'a^{i+1}b^{i+1}d') \Rightarrow ca^{i+1}b^{i+1}d'$ is then performed by tube 3, the product of which is filtered into tube 4. Tube 4 finally performs the operation $(ca^{i+1}b^{i+1}d', d'd) \Rightarrow ca^{i+1}b^{i+1}d$ which is accepted by simple tube 1. This process repeats and we thus note that.

$$L(\Gamma) \cap ca^+b^+d = \{ca^nb^nd | n \in N\}$$

Hence $L(\Gamma)$ is not a regular language.

In fact, under morphism and intersection with regular sets, $STT_{\infty}^{(2,3)}(FIN)$ characterises the whole of CF.

Theorem 11 *For each $L \in CF$, there exists a STT system of type (2,3) Γ, a morphism h and regular set $R \in REG$ such that $L = h(L(\Gamma) \cap R)$.*

Proof Consider a CF grammar $G = (N.T, P, S)$ such that $L = L(G)$ in the Kurodo normal form where rules are of the form

$$(r): \quad A \rightarrow BC, A, B, C \in N$$

$$(r'): \quad A- > w, A \in N, w \in T^*$$

Let there be k rules of type (r) in the grammar G. Consider the STT system (of type (2,3))

$$\begin{aligned}\Gamma = (V, (A_1, M_1, V_1), \quad & (A_{r_1 1}, M_{r_1 1}, V_{r_1 1}), \ldots, (A_{r_1 5}, M_{r_1 5}, V_{r_1 5}), \\ & \vdots \\ & (A_{r_k 1}, M_{r_k 1}, V_{r_k 1}), \ldots, (A_{r_k 5}, M_{r_k 5}, V_{r_k 5}))\end{aligned}$$

where

$V = T \cup \{A_1, A_2 | A \in N\} \cup \{r | r : A \rightarrow BC \in P\}$

$A_1 = \{A_1 w A_2 | A \rightarrow w \in P\}$

$M_1 = \Phi$

$V_1 = T \cup \{S_1, S_2\}$

For each rule (r_i) of the type $(r_i) : A \rightarrow BC, 1 \leq i \leq k$, we have the following 5 simple tubes.

$A_{r_i 1} = \{B_2 r_i\}$

$M_{r_i 1} = \{B_2\}$

$V_{r_i 1} = T \cup \{B_1, B_2\}$

$A_{r_i2} = \{r_iC_1\}$

$M_{r_i2} = \{C_1\}$

$V_{r_i2} = T \cup \{C_1, C_2\}$

$A_{r_i3} = \{A_1B_1\}$

$M_{r_i3} = \{B_1\}$

$V_{r_i3} = T \cup \{B_1, r_i\}$

$A_{r_i4} = \{C_2A_2\}$

$M_{r_i4} = \{C_2\}$

$V_{r_i4} = T \cup \{r_i, C_2\}$

$A_{r_i5} = \Phi$

$M_{r_i5} = \{r_i\}$

$V_{r_i5} = T \cup \{A_1, A_2, r_i\}$

We shall show that if $A \Rightarrow^* w(\in T^*)$ for some $A \in N$, then A_1wA_2 is generated by one of the tubes. To start with we have for all $A \to w \in P$, A_1wA_2 as axioms in simple tube 1.

Suppose, we are able to generate strings $B_1w_1B_2$ and $C_1w_2C_2$ and we have the rule $(r_i) : A \to BC \in P, 1 \leq i \leq k$, we shall show that $A_1w_1w_2A_2$ is also generated by one of the simple tubes. Simple tube (r_i1) accepts $B_1w_1B_2$ and by splicing $(B_1w_1B_2, B_2r_i) \Rightarrow B_1w_1r_i$. $B_1w_1r_i$ is accepted by simple tube (r_i3) and by splicing $(A_1B_1, B_1w_1r_i) \Rightarrow A_1w_1r_i$ is generated. Similarly simple tube (r_i2) accepts $C_1w_2C_2$ and generates $r_iw_2C_2$ which is then further spliced in simple tube (r_i4) along with C_2A_2 to give $r_iw_2A_2$. Strings $A_1w_1r_i$ and $r_iw_2A_2$ are then spliced in simple tube (r_i5) to give $A_1w_1w_2A_2$.

Thus, all strings of the form $S_1wS_2, w \in L(G)$ are generated in due course and are filtered in simple tube 1. Consider the regular set $R = S_1T^*S_2$ and the morphism $h : T \cup \{S_1, S_2\} \to T$ defined by $h(a) = a, a \in T$ else $h(a) = \lambda$. We clearly see that

$$L = L(G) = h(L(\Gamma) \cap R)$$

Thus, $STT_{\infty}^{(2,3)}(FIN)$ characterises CF under intersection with regular sets and morphisms. $\square$

Chapter 4

Conclusions

In this thesis, we defined the notion of distributed-automata and analysed the increase in acceptance capacity for various types of automata like FSA, PDA, k-turn PDA. In the case of FSA, we showed that the distributed FSAs are no more powerful than the "centralised" FSAs. However for the case of PDAs and 1-turn PDAs, we proved that a distributed system with just two components accepts the family RE. It is an interesting problem to analyse the descriptional complexity of these devices.

We then extended the notion of parallel communicating grammar for simple H systems and defined distributed simple H systems (simple test tube systems). The increase in generative power of distributed simple H systems over simple H systems was then established. Simple test tube systems under morphisms and intersection with regular sets were proven to characterise the entire of CF. It is open for research to determine the exact nature of the family of languages generated by distributed simple H systems.

Bibliography

[1] E. Csuhaj-Varju, J. Dassow, On Cooperating Distributed Grammar Systems, *J. Inform. Process. Cybern., EIK,* 26(1990), 49-63.

[2] E. Csuhaj-Varju, J. Dassow, J Kelemen, Gh. Păun, *Grammar Systems, A grammatical Approach to Distribution and Cooperation,* Gordon and Breach, London, 1994.

[3] E. Csuhaj-Varju, L. Kari, Gh. Păun, Test Tube distributed systems based on splicing, *Computers and AI,* 15, 2-3(1996), 211-232.

[4] K. Culik II, A purely homomorphic characterisation of recursively enumerable sets, *Journal of the ACM,* 26(1979), 345-350.

[5] K. Culik II, T. Harju, Splicing Semigroups of Dominoes and DNA, *Discrete Appl. Math.,* 31(1991), 261-277.

[6] J. Dassow, J. Kelemen, Cooperating distributed grammar systems: a link between formal languages and artificial intelligence, *Bulletin of EATCS,* 45(1991), 131-145.

[7] J. Dassow, Gh. Păun, G. Rozenberg, Grammar Systems, chapter 4 in vol 2 of [16], 155-213.

[8] T. Head, Formal Language Theory and DNA: an analysis of the generative capacity of specific recombinant behaviours, *Bull. Math. Biology,* 49(1987), 737-759.

[9] T. Head, Gh. Păun, D. Pixton, language Theory and Molecular genetics, generative mechanisms suggested by DNA recombination, chapter 7 in vol 2 of [16], 295-360.

[10] A. Mateescu, Gh. Păun, G. Rozenberg, A. Salomaa, Simple Splicing Systems, *Discrete Appl. Math.,* 1997.

[11] R. Meersman, G. Rozenberg, D. Vermeir, Cooperating Grammar Systems, *Proc. MFCS '78, LNCS* 64, Springer-Verlag, Berlin, 1978, 364-374. .

[12] Gh. Păun, DNA Computing; Distributed Splicing System, in *Structures in Logic and Computer Science, A Selection of Essays in Honor A. Ehrenfeucht* (J. Mycielski, G. Rozenberg, A. Salomaa., eds) *Lecture Notes in Computer Science,* 1261, Springer-Verlag, 1997.

[13] Gh. Păun, Regular extended H systems are computationally universal, *J. Automata, Languages, Combinatorics,* 1, 1(1996), 27-36.

[14] Gh. Păun, G. Rozenberg, A. Salomaa, Computing by Splicing, *Theor. Computer Sci.,* 1996, 168, 2(1996), 321-336.

[15] D. Pixton, Regularity of Splicing languages, *Discrete Appl. Math.,* 1995, 69(1996), 101-124.

[16] G Rozenberg, A. Salomaa, Eds, *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Hiedelberg, 1997.

[17] A. Salomaa, *Jewels of Formal Language Theory,* Computer Science press, Rockville, Maryland, 1981.

[18] L. Săntean, Parallel Communicating Systems, *Bulletin of EATCS,* 42(1990), 160-171.