Tree Transducers and Tree Insertion Systems

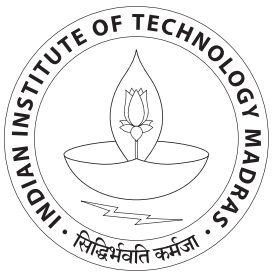
A Project Report

submitted by

SUNIL K.S

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

April 2011

THESIS CERTIFICATE

This is to certify that the thesis entitled **Tree Transducers and Tree Insertion Systems**, submitted by **Sunil K.S**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Kamala Krithivasan
Research Guide
Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

My stay at IIT Madras was enriched by the timely guidance and help from a large number of people. I wish to express my immense grattitude to the following people who haven helped me in various ways during my stay here:

My giude , Prof. Kamala Krithivasan, for her invaluable giudance and support without which it would not have been possible to put together this report. She has been a continuous source of motivation and encouragement. I am especially gratful for immence amount of patience she has had through all the stages of my work. Also I would like to thank her for keeping the Theoretical Computer Science (TCS-II) Lab a ver lively and homely place. The wonderful atmosphere encourages me to come and spend a lot of time in the lab. The lab is an informal place to work in. I have really enjoyed my time here.

Our Head of the Department, Prof. Siva Ram Murthy C, for all encouragements and support he has given me to do my project.

Our Faculty advisor Dr. Madhu Mutyam, Project co-ordinator Dr. Narayanaswamy N, for their timely advices and support to complete my project in time.

Dr. Jaylal Sharma, for his valuable suggestions and advices. His renowned course on complexity theory has helped me a lot in my work.

All faculty members of our departmet, for their help and encouragements.

Ms.Saradhambal and Ms.Radhaikumar, for patiently listening to my troubles every time.

Department librarians Ms.Prema and Mr.Balasundaram, for all their help.

Ms.Lisha Varghese, my under graduate teacher, for her constant support, encouragements and valuable suggestions without which it would not have been do my work with this much concentration and interest.

My labmate Ajeesh Ramanujan, for all the help and support.

My dear friends Jethin, Pranav, Shyam, Sirosh, Aravind, Jobin and Abhijith, for making my stay at IIT memorable.

My B.Tech classmates Arun Babu and Subin George, for taking a special care about me during the entire period of my stay in Chennai.

Amma, Valyammachi, Ammavan, Ammayi, Kannan, Chechi, Chettan and Sreekuttan, for always ensuring that I didnt feel lost here, for their tender concern they have for me, and most of all, for their unwavering belief in me, at all times.

My best friends Raghesh, Balagopal and Jyothikrishna, for making sure I never really felt lonely while I was here. I thank them most of all, for always boosting my spirits, especially in the most trying of times. I could not have asked for a better group of friends.

ABSTRACT

KEYWORDS: Markov Decision Processes, Symmetries, Abstraction

Tree Transducers

Applications

Insertion deletion System

In trees

Various Types

Comparison with Grammars and Automata

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
RTFM	Read the Fine Manual

NOTATION

r	Radius, m
α	Angle of thesis in degrees
β	Flight path in degrees

CHAPTER 1

INTRODUCTION

1.1 Theoretical Computer Science

Language is a systematic means of communicating by the use of sounds or conventional symbols. All language have its own alphabet from which words are formed. A meaningful collection of words is known as a sentence. For every lanuage there is some rules for sentence formation. ie., Grammar is the system of relationships between elements of the sentence that links the sounds to the meanings.

A formal language is a set of words, i.e. finite strings of letters, symbols, or tokens. The set from which these letters are taken is called the alphabet over which the language is defined. A formal language is often defined by means of a formal grammar. Formal languages are studied in computer science and linguistics; the field of formal language theory studies the purely syntactical aspects of such languages.

A formal grammar (sometimes simply called a grammar) is a set of rules for forming strings in a formal language. The rules describe how to form strings

from the language's alphabet that are valid according to the language's syntax. A grammar does not describe the meaning of the strings or what can be done with them in whatever contextonly their form.

1.1.1 Importance

1.1.2 Applications

1.2 Transducers

1.2.1 Tree Transducers

Different Types

1.2.2 Applications

1.3 Insertion Deletion Systems

1.3.1 In Strings

1.3.2 In Trees

Difference when compared to string insertion

1.4 Motivation for this Work

1.5 Contributions of the Report

1.6 Outline of the Report

CHAPTER 2

PRELIMINARIES

2.1 Basics of Formal Language

The basic data structure or input to grammars or automaton are strings. Strings are defined over a finite alphabet. Alphabet may vary depending upon application. Elements of an alphabets are called symbols. Usually we denote the basic alphabet set either as Σ or T. For example,

$\Sigma_1 = \{a, b\}$

$\Sigma_2 = \{0, 1, 2\}$ are few examples for alphabet set.

A string or word is a finite sequence of symbols from that alphabet, usually written as concatenated symbols and not seperated by commas or space. If w is a string over an alphabet Σ, then the length of w written as $len(w)$ or $|w|$ is the number of symbols it contains. For example, if $\Sigma_1 = \{a, b\}$, a string *abbab* is a string or word over Σ with $|w|$=5. If $|w|$=0, then w is called as emty string denoted either as λ or ϵ.

2.1.1 Grammars

Definition 1. *A Grammar is a construct* $G = (N, T, S, P)$ *where,*

- N *is a finite set of non terminals,*
- T *is a finite set of terminals,*
- $S \in N$ *is the start symbol,*
- P *is a finite set of production rules.*

Chomsky's classifiaction of grammars.

- Type 0 or Phrase structured grammar
- Type 1 or context sensitive grammar
- Type 2 or Context free grammar
- Type 3 or Regular grammar

2.1.2 Regular Expressions

Definition 2. *Let* Σ *be an alphabet. For each* $a \in \Sigma$, $\boldsymbol{a}$ *is a regular expression representing the regular set* $\{a\}$. ϕ *is a regular expression representing the empty set.* ϵ *is a regular expression representing the set* $\{\epsilon\}$. *If* r_1 *and* r_2 *are regular expression representing the regular sets* R_1 *and* R_2 *respectively, then* $r_1 + r_2$ *is a regular expression representing* $R_1 \cup R_2$. $r_1.r_2$ *is a regular expression representing* R_1R_2. r_1^* *is a regular expression representing* R_1^*.

Any expression obtained from $\phi, \epsilon, a(a \in \Sigma)$, *using the above operations and paranthesis where required, is a regular expression.*

Example 1. $a(bc)^*bcd$ *represents the regular set:* $\{a(bc)^n d | n \geq 1\}$

2.1.3 Automata

2.2 Insertion-deletion System

Insertion-deletion systems are one of the models studied inspired by biology [2]. The operation of insertion and deletion on strings have some relevances to some phenomena in human genetics [3]. A DNA strand can be inserted into/deleted from another strand. The idea of insertion-deletion has been extended to arrays also [5]. In this paper we consider the insertion systems for trees. Trees are important data structures and find use in many applications from the description of parse trees to representation of XML and DTD [6]. Considering insertion systems in trees can have profound applications in such areas [1].

2.2.1 DNA Computing

DNA is the molecule which plays the central role in *DNA* computing. *DNA* is a *polymer*(large molecule) which strung together form *monomers* (small molecules) called *deoxyribonucleotides(nucleotides)*. *DNA* is a crucial molecule in living cells and it has a fascinating structure which supports two most important functions of *DNA*: coding for production of protiens and self replication so that an exact copy is passed to the offspring cell.

Each nucleotide consists of three components: a sugar, a phosphate group and a nitrogenous base. The sugar(*deoxyribose*) has five carbons which are numbered from 1′ to 5′. The phosphate group is attached to the 5′ carbon and the base is attached to 1′ carbon. Within the sugar structure there is a *hydroxyl group(OH)* attached to the 3′ carbon.

Different nucleotides differs only by their bases which come in two sorts: *purines and pyramidines*. *aDENINE(a)* and *guanine(G)* are purine bases and *cytosine(C)* and *thymine(T)* are pyramidine bases.

Nucleotides can be linked together in two ways.

- The 5′-phoshpate group of one nucleotide is joined with the 3′-hydroxyl group of other forming a *phosphodiester* bond which is a string(*covalent*) bond
- The base of one nucleotide interacts with the base of other to form a weak(*hydrogen*) bond. Here *A* and *T* can pair together and *C* and *G* can pair together- no other pairing are possible.

  Fusing two single stranded *DNA* molecules by complementary base pairing is known as *annealing*.

Since the hydrogen bond between complementary bases is weaker than the phosphodiester bond between the consecutive nucleotides within one strand, we can seperate two strnds of *DNA* without breaking the single strand. This seperation can be acheived by heating *DNA* solution to a temperature between 85°C and 90°C which is the melting temperature of *DNA*. This process of seperating *DNA* strands by heating the solution is known as *denaturation*.

If the heated solution is allowed to cool down slowly, the seperated strands fuse again by hydrogen bonds. This process is called *renaturation(reannealing)*.

Enzymes are protiens that catalyze chemical reactions taking place in living cells. A class of enzymes called *(DNA) polymerases* is able to add nucleotides to an existing *DNA* molecule. To do so, they require an existing single stranded templates which prescribes the chain of nucleotides to be added and an already existing sequence*(primer)* which is bounded to a part of the template with the 3′ end available for extension.

2.3 Tree Language

Let N be the set of positive integers. Then the set of finite strings over N is denoted by N^*. The empty string is denoted by ϵ. A **ranked alphabet** Σ is a finite set of symbols together with a function $Rank$: $\Sigma \rightarrow N$. For $f \in \Sigma$, the value $Rank(f)$ is called the rank of f. For every $n \geq 0$, we denote by Σ_n the set of all symbols of rank n. Elements of rank $0, 1, \cdots, n$ are respectively called constants, unary, $\cdots$, n-ary symbols.

A **tree t** [4] over an alphabet Σ is a partial mapping $t : N^* \rightarrow \Sigma$ that satisfies the following conditions:

- *dom(t)* is a finite, prefix-closed subset of N^*, and
- for each $p \in dom(t)$, if $Rank(t(p)) = n > 0$, then $\{i | p.i \in dom(t)\} = \{1, 2, \cdots, n\}$

Each $p \in dom(t)$ is called a **node** of t. The node with domain element Σ is the **root**. For a node p, we define the i^{th} *child* of p to be the node $p.i$, and we define the i^{th} *subtree* of p to be the tree t' such that $t'(p) = t(p.i.p'), \forall p' \in dom(t)$. We denote by $T(\Sigma)$ the set of all trees over the alphabet Σ. The **size** of a tree is the number of elements in $dom(t)$. The **height** of a tree t is $max\{|w| : w \in dom(t)\}$. Given a finite tree t, the **frontier** of t is the set $\{p \in dom(t) | \forall n \in N, p.n \notin dom(t)\}$. A tree with root a and subtrees $t_1, t_2, \cdots, t_r$ is represented by $a(t_1, t_2, \cdots, t_r)$.

Example 2. *Let* $\Sigma = \{a, b, c, h\}, a, b \in \Sigma_2, c \in \Sigma_1, h \in \Sigma_0$.
A tree over Σ *and its diagrammatic representation is shown in Figure 2.1*

Let t be the tree $a(b(b(h,h),h),c(h))$.
$dom(t) = \{\epsilon, 1, 1.1, 1.1.1, 1.1.2, 1.2, 2, 2.1\}$.
$size(t) = 8$.
$height(t) = 3$.
$frontier(t) = \{1.1.1, 1.1.2, 1.2, 2.1\}$.

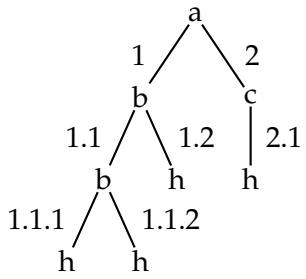


Figure 2.1: A tree and its diagrammatic representation

2.3.1 Tree Grammars

Regular Tree Grammars

A Regular Tree Grammar (RTG) [4] is a construct $G = (N, T, S, P)$ where,

- N is a finite set of non terminals,
- T is a finite set of terminals,
- $S \in N$ is the start symbol,
- P is a finite set of production rules.
  Each rule in P is of the form $X \rightarrow x(X_1, X_2, \cdots, x_p), p \geq 0$, where $X, X_1, X_2, \cdots, X_p \in N$ and $x \in T$

The **language generated** by an RTG G is represented by $L(G)$, is defined as the set of trees generated by G using productions rules in P, starting from S. A **regular tree language** is a language generated by a regular tree grammar.

Example 3. $G_{r_2} = (N, T, S, P)$ *where,* $N = \{S, B, B', C, H\}$, $T = \{a_2, b_1, c_1, h_0\}$

$P = \{S \rightarrow a(B, C), B \rightarrow b(B'), B' \rightarrow b(B)|b(H), C \rightarrow c(C)|c(H), H \rightarrow h\}$

$L(G_{r_2}) = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%2 = 0\}$

Context-free Tree Grammars

A Context-free Tree Grammar (*CTG*) [4] is a construct $G = (S, N, \mathcal{F}, R)$ where the rules have the form $X(x_1, x_2, \cdot, x_n) \rightarrow t$ with t a tree of $T(\mathcal{F} \cup N \cup \{x_1, x_2, \cdot, x_n\})$, $\{x_1, x_2, \cdot, x_n\} \in \mathcal{X}$ where $\mathcal{X}$ is the set of reserved variables with $\mathcal{X} \cap (\mathcal{F} \cup) N) = \phi$, $\mathcal{X}$ a non-terminal of arity n.

The **language generated** by an *CTG* G is represented by $L(G)$, is defined as the set of trees generated by G using productions rules in P, starting from S. A **context-free tree language** is a language generated by a context-free tree grammar.

Example 4. *The grammar of axiom Prog, set of non-terminals* $\{Prog, Nat, Fact()\}$*, the set of terminals* $\{0, s, if(,), eq(,), not(), times(,), dec()\}$ *and rules*

$$Prog \rightarrow Fact(Nat)\ , \qquad Nat \rightarrow 0\ ,\ Nat \rightarrow s(Nat)$$

$$Fact(x) \rightarrow if(eq(x, 0), s(0))\ , \quad Fact(x) \rightarrow if(not(eq(x, 0)), times(x, Fact(dec(x))))$$

where $\mathcal{X} = x$ *is a context-free tree grammar. Here the last rule is the classical definition of the factorial function.*

2.3.2 Tree Automata

A ***non deterministic botttom-up finite tree automata*** (NFTA) [4] over an alphabet Σ is a tuple
$D = (Q, \Sigma, Q_f, \Delta)$ where,

- Q is a finite set of states,
- Σ is a ranked input alphabet,
- $Q_f \subseteq Q$ is a set of final states,

- Δ is a finite set of transition rules.

Each transition rule is a triple of the form $((q_1, q_2, \cdots, q_n), f, q)$ where $q_1, q_2, \cdots, q_n, q \in Q, f \in \Sigma_n$, i.e. $Rank(f) = n$. We use $f(q_1, q_2, \cdots, q_n) \rightarrow q$ to denote that $((q_1, q_2, \cdots, q_n), f, q) \in \Delta$. If $Rank(f) = 0$,i.e. f is a constant, then we use rules of the form $f \rightarrow q$. The epsilon rules are denoted by rules of the form $q_i \rightarrow q_j$. A ***run*** of A over a tree $t \in T(\Sigma)$ is a mapping $r : dom(t) \rightarrow Q$ such that for each node $p \in dom(t)$ where $q = r(p)$, we have that if $q_i = r(pi)$ for $1 \leq i \leq n$ then Δ has the rule $t(p)(q_1, q_2, \cdots, q_n) \rightarrow q$.

Example 5. *The tree automata $D_{r_4} = (Q, \Sigma, Q_f, \Delta)$ accepts trees with odd number of a's over the alphabet $\Sigma = \{a_2, b_2, c_0\}$ where $Q = \{e, o\}, Q_f = \{o\}$ and Δ contains following transitions.*

$$c \rightarrow e \quad b(e,e) \rightarrow e \quad b(o,o) \rightarrow e \quad a(e,o) \rightarrow e \quad a(o,e) \rightarrow e$$
$$b(e,o) \rightarrow o \quad b(o,e) \rightarrow o \quad a(e,e) \rightarrow o \quad a(o,o) \rightarrow o$$

Distributed Tree Automata

Distributed Nondeterministic Tree Automata (DNTA) In this section we define distributed nondeterministic tree automata(DNTA), the different modes of acceptance of DNTA and discuss the power of different modes of acceptance.

Definition 3. *A DNTA is a 4-tuple $D = (K, \Sigma, F, \Delta)$ where,*

- *K is an n-tuple $(K_1, K_2, \cdots, K_n)$ where each K_i is a set of states of the i^{th} component;*
- *Σ is a finite set of ranked alphabet;*
- *$F \subseteq \bigcup_i K_i$ is the set of final states;*

- Δ *is a n-tuple* $(\delta_1, \delta_2, \cdots, \delta_n)$ *of state transistion function where each* δ_i *is a set of transition rules of the* i^{th} *component having the form* $f(q_1, q_2, \cdots, q_n) \rightarrow q, f \in \Sigma_n, q_1, q_2, \cdots, q_n \in K_i, q \in \bigcup_i K_i$ *or* $q_i \rightarrow q_j$.

In the case of DNTA, 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 $*$-mode, t-mode, $\leq k$-mode, $\geq k$-mode, and $= k$-mode, where k is a positive integer. Description of each of the above modes of acceptance is as follows:

***t*-mode acceptance :**An automaton that has a leaf transition rule begins processing the input tree. Suppose that the system starts from the component i. The control stays in component i as long as it can follow the transition rules in component i. Otherwise, it transfers the control to some other component j, $j \neq i$ which has the transition function to proceed. If more than one component succeeds, then the selection of j is done nondeterministically. The process is repeated and we accept the tree if the system reaches any one of the final states. It does not matter which component the system is in while accepting.

Definition 4. *The instantaneous description(ID) of a DNTA* $D = (K, \Sigma, F, \Delta)$ *working in t-mode is given by a triple* (B, t, i) *where* $B \subseteq \bigcup_i K_i$ *and it denotes the current active state set of the whole system,* $t \in T(\Sigma \cup \bigcup_i K_i)$ *and* $i, 1 \leq i \leq n$ *the index of the component in which the system is currently in.*

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

i. $(B, t, i) \vdash_t (B', t', i)$ if there is a rule of the form $a(q_1, q_2, \cdots, q_n) \rightarrow q' \in \delta_i$ such that t' is obtained from t by replacing a subtree of t of the form $a(t_1, t_2, \cdots, t_n)$ by $q'(t_1, t_2, \cdots, t_n)$, where $a \in \Sigma_n, n \geq 0, t_1, t_2, \cdots, t_n \in T(\bigcup_i K_i), r(root(t_1)) = q_1, r(root(t_2)) = q_2, \cdots,$
$r(root(t_n)) = q_n, q_1, q_2, \cdots, q_n \in B$ and B' is the set of active state set after performing the transition.

ii. $(B, t, i) \vdash_t (B, t, j)$ iff component i does not have a transition to proceed and component j has a transition to proceed.

The reflexive and transitive closure of $\vdash_t$ is denoted by $\vdash_t^*$. We now give an example of a distributed bottom up tree automata working in t-mode.

Example 6. *Consider the language* $L_1 = \{a(bd(g^j d)^i(f), ce(h^k e)^i(f)), i, j, k \geq 1\}$ *over* $\Sigma = \{a, b, c, d, e, f, g, h\}, a \in \Sigma_2, b, c, d, e, g, h \in \Sigma_1, f \in \Sigma_0$.

We define a distributed tree automaton $D_1 = (K, \Sigma, \{q_a\}, \Delta)$ *working in t-mode as follows. The components are defined as follows*

- *Component* 1
  - $K_1 = \{q_f, q_g, q_1, q_2\}$
  - $\delta_1 = \{d(q_f) \to q_g, g(q_g) \to q_1, g(q_1) \to q_1, q_2 \to q_f\}$
- *Component* 2
  - $K_2 = \{q_f, q_e, q_1, q_2\}$
  - $\delta_2 = \{e(q_f) \to q_e, h(q_e) \to q_2, h(q_2) \to q_2, q_1 \to q_f\}$
- *Component* 3
  - $K_3 = \{q_f, q_a, q_b, q_c, q_d, q_e, q_1, q_2\}$
  - $\delta_3 = \{f \to q_f, b(q_1) \to q_b, c(q_2) \to q_c, a(q_b, q_c) \to q_a\}$

The processing starts in component 3 , with the two leaves using the rule $f \to q_f$. As further processing is not possible in component 3, processing continues with 2 or 1. Then it alternates between 1 and 2 processing d's, g's, e's and f's. Finally when the labels are b and c, processing takes the tree to q_b and q_c and in component 3 state q_a is reached by the root.

CHAPTER 3

EARLIER WORK

3.1 Insertion deletion System

The operations of insertion and deletion are fundamental in formal language theory, and generative mechanisms based on them have been considered (with linguistic motivation) since a long time ago, see [7] .

In general form, an insertion operation means adding a substring to a given string in a specified context, while a deletion operation means removing a substring of a given string from a specified context. A finite set of insertion-deletion rules, together with a set of axioms provide a language generating device (an *Insdel system*): starting from the set of initial strings and iterating insertion-deletion operations as defined by the given rules we obtain a language. The number of axioms, the length of the inserted or deleted strings, as well as the length of the contexts where these operations take place are natural descriptional complexity measures in this framework. As expected, insertion and deletion operations with context dependence are very powerful, leading to characterizations of recursively

enumerable languages.

The paper [8] contains an unexpected result: context-free insertion-deletion systems with one axiom are already universal, they can generate any recursively enumerable language. Moreover, this result can be obtained by inserting and deleting strings of a rather small length, at most three. The same paper stated an open question about context-free insertion-deletion systems having rules dealing with strings of length at most two.

In [9] the author answer this open question and showed that if the length of the inserted and deleted string is at most two, then such systems generate a particular subset of the family of context-free languages. They also showed that the traditional complexity measures for insertion-deletion systems, in particular the size of contexts, need a revision and we propose new measures based on the total weight.

The insertion and deletion operations with context dependendence are given in [2]. Here the authors explained how a string is inserted/deleted between two given strings using the concept of *DNA* computing.

The *insertion* process can be described as follows.

Suppose if we add a single stranded *DNA* sequence of the form $5' - x_1uvx_2z - 3'$ to a test tube which contains another single stranded *DNA* sequence of the form $3' - \bar{u}\bar{y}\bar{v} - 5'$ where x_1, x_2, u, v, z are strings, $\bar{u}, \bar{v}$ are the Wartson-Crick complements of the strings u, v and $\bar{y}$ is the complement of some new string y. Then the *DNA* strand will *anneal*, $\bar{u}$ will stick to u and $\bar{v}$ to v, folding $\bar{y}$ as shown in figure **??**. If we cut the double stranded subsequence uv by using a restricyion enzyme: adding

$\bar{z}$, which acts as *primer*, we will get a complete double stranded sequence. We can now seperate the two strands by melting the solution and hence we obtain two strands x_1uyvx_2z and $\bar{x_1}\bar{u}\bar{y}\bar{v}\bar{x_2}\bar{z}$. Hence the string y has been inserted between strings u and v.

Similarly by using a mismatching annealing, deletion operation which is controlled by a context, can also perform theoretically. The different steps in deletion process is shown in figure **??**.

Definition 5. *An Insdel system [9] is a construct $\gamma = (V, T, A, I, D)$, where V is an alphabet, $T \subseteq V$, A is a finite language over V , and I, D are finite sets of triples of the form (u, α, v), of strings over V . The elements of T are terminal symbols (in contrast, those of VT are called nonterminals), those of A are axioms, the triples in I are insertion rules, and those from D are deletion rules. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted in between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed from the context (u, v). Stated otherwise, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \rightarrow u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \rightarrow uv$. We denote by $\Rightarrow_{ins}$ the relation defined by an insertion rule (formally, $x \Rightarrow_{ins} y$ iff $x = x_1uvx_2$, $y = x_1u\alpha vx_2$, for some $(u, \alpha, v) \in I$ and $x_1, x_2 \in V^*$) and by $\Rightarrow_{del}$ the relation defined by a deletion rule (formally, $x \Rightarrow_{del} y$ iff $x = x_1u\alpha vx_2$, $y = x_1uvx_2$, for some $(u, \alpha, v) \in D$ and $x_1, x_2 \in V^*$). We refer by $\Rightarrow$ to any of the relations $\Rightarrow_{ins}$, $\Rightarrow_{del}$, and denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$ (as usual, $\Rightarrow^*$ is the transitive closure of $\Rightarrow$).*

The language generated by γ is defined by $L(\gamma) = \{w \in T^* | x \Rightarrow^* w$, for some $x \in A\}$.

An Insdel system $\gamma = (V, T, A, I, D)$ is said to be of weight $(n, m; p, q)$ if

$n = max\{|\alpha|| (u, \alpha, v) \in I\}$,

$m = max\{|u||(u, \alpha, v)Ior(v, \alpha, u) \in I\}$,

$p = max\{|\alpha||(u, \alpha, v) \in D\}$,

$q = max\{|u||(u, \alpha, v) \in Dor(v, \alpha, u) \in D\}$,

The total weight of γ *is the sum* $m + n + p + q$.

Example 7. *Consider the Insdel system* $ID = (T, T, A, I, \phi)$, *with* $T = \{0, 1\}$, $A = \{01\}$ *and* $I = \{(0, 01, 1)\}$. *Here at each derivation step the string* 01 *is inserted in the middle of the word obtained on the previous step because this is the only place where the context* 01 *occurs. Since we start from* 01 *it is clear that* $L(ID) = \{0^n 1^n | n \geq 1\}$. *Here the weight of the system is (2, 1 ; 0, 0) and has the total weight equal to 3.*

We denote by $INS_n^m DEL_p^q$, for $n, m, p, q \geq 0$, the family of languages $L(\gamma)$ generated by Insdel systems of weight $(n, m; p, q)$ such that $n \leq n$, $m \leq m$, $p \leq p$, $q \leq q$. If some of the parameters n, m, p, q is not specified, then we write instead the symbol . Thus, $INS_*^0 DEL_*^0$ denotes the family of languages generated by context-free Insdel systems, i.e., with insertion rules of the form $(\epsilon, \alpha, \epsilon) \in I$ and deletion rules of the form $(\epsilon, \alpha, \epsilon) \in D$, where ϵ denotes the empty string.

3.1.1 Comparison with grammar systems

We denote by $CF\ INSDEL_{m,p}$ the family of context-free Insdel systems having the length of the inserted string at most m and the length of the deleted string at most p. For a system $ID \in CF\ INSDEL_{m,p}$ we shall also say that ID is of size (m, p).

Context-free Insdel systems have an interesting particularity: insertions and deletions are uncontrolled and may happen at any time at any place in a string. This fact can give an impression that such systems cannot be controlled in order to perform computations. However, this affirmation is not true. In [8] it is shown

that, in spite of the above remark, such systems are able to simulate an arbitrary Chomsky grammar.

Example 8. *Consider a context-free Insdel system* $ID_2 = (V, T, \{S\}, I, D)$ *of size (3, 2) with* $T = \{a, b\}$, $V = T \cup \{S, S'\}$, $I = \{S'aSb, S'ab\}$ *and* $D = SS'$. *The computation in this system goes as follows. Given a word* w_1Sw_2 *(initially* $w_1, w_2 = \epsilon$*) the string* $S'aSb$ *is inserted after* S, *which produces* $w_1SS'aSbw_2$. *After that, the deletion rule eliminates* SS *and word* w_1aSbw_2 *is obtained. It is easy to see that this corresponds to rewriting rule* $S \rightarrow aSb$. *If the string* $S'aSb$ *is not inserted immediately after* S, *then symbol* S *cannot be eliminated, hence it will not be possible to generate a terminal string.*

Therefore, system ID_2 *simulates the grammar with productions* $\{S \rightarrow aSb, S \rightarrow ab\}$. *Hence,* $L(ID_2) = \{a^n b^n | n \geq 1\}$.

Insdel systems of a sufficiently large weight can characterize *RE*, the family of recursively enumerable languages.

The following results about the comparison of insdel system recursively enumerable languagesare given in [2] with proofs.

- $RE = INS_3^2 DEL_3^0$
- $RE = INS_1^2 DEL_1^1$
- $RE = INS_1^1 DEL_2^0$
- $RE = INS_*^1 DEL_0^0$
- $INS_2^2 DEL_0^0$ contains non semilinear languages.

In [10] the author gave the following results

from [2; 8; 9; 10], we can summerize the results as in table **??**.

No.	Total weight	(n, m; p, q)	Family generated	References
1	6	(3, 0; 3, 0)	RE	[8]
2	5	(1, 2; 1, 1)	RE	[3; 2]
3	5	(1, 2; 2, 0)	RE	[3; 2]
4	5	(2, 1; 2, 0)	RE	[3; 2]
5	5	(1, 1; 1, 2)	RE	[1]
6	5	(2, 1; 1, 1)	RE	[1]
7	5	(2, 0; 3, 0)	RE	[8]
8	5	(3, 0; 2, 0)	RE	[8]
9	4	(1, 1; 2, 0)	RE	[2]
10	4	(1, 1; 1, 1)	RE	[1]
11	4	(2, 0; 2, 0)	$\subset$ CF	[9]
12	m+1	(m, 0; 1, 0)	$\subset$ CF	[9]
13	p+1	(1, 0; p, 0)	$\subset$ REG	[9]
14	4	(1, 2; 1, 0)	?	

Table 3.1: Comparison of insdel system and recursively enumerable languages

3.1.2 Contextual Grammars

3.1.3 Tree Adjoining Grammars(TAG)

3.1.4 Contextual Tree Adjoining Grammars(CTAG)

3.2 Tree Transducer

3.2.1 Top-down Tree Transducers

Example 9. *The top-down tree transducer for number convertion*

$q(crore2(x1, x2)) \to 10000000(q(x1), q(x2))$	$q(crore1(x1)) \to 10000000(q(x1), 100000(0, 0))$	$q(lakh2(x1, x2)) \to 100000(q(x1), q(x2))$
$q(lakh1(x1)) \to 100000(q(x1), 1000(0, 0))$	$q(thousand2(x1, x2)) \to 1000(q(x1), q(x2))$	$q(thousand1(x1)) \to 1000(q(x1), 100(0, 0))$
$q(hundred2(x1, x2)) \to 100(q(x1), q(x2))$	$q(hundred1(x1)) \to 100(q(x1), 10(0, 0))$	$q(-ty2(x1, x2)) \to 10(2 + (x1), 1 + (x2))$
$q(-ty1(x1)) \to 10(2 + (x1), 0)$	$q(-teen1(x1)) \to 10(1, 3 + (x1))$	$q(zero) \to 10(0, 0)$
$q(one) \to 10(0, 1)$	$q(two) \to 10(0, 2)$	$q(three) \to 10(0, 3)$
$q(four) \to 10(0, 4)$	$q(five) \to 10(0, 5)$	$q(six) \to 10(0, 6)$
$q(seven) \to 10(0, 7)$	$q(eight) \to 10(0, 8)$	$q(nine) \to 10(0, 9)$
$q(ten) \to 10(1, 0)$	$q(eleven) \to 10(1, 1)$	$q(twelve) \to 10(1, 2)$
$1 + (zero) \to 0$	$1 + (one) \to 1$	$1 + (two) \to 2$
$1 + (three) \to 3$	$1 + (four) \to 4$	$1 + (five) \to 5$
$1 + (six) \to 6$	$1 + (seven) \to 7$	$1 + (eight) \to 8$
$1 + (nine) \to 9$	$2 + (twen-) \to 2$	$2 + (thir-) \to 3$
$2 + (four) \to 4$	$2 + (fif-) \to 5$	$2 + (six) \to 6$
$2 + (seven) \to 7$	$2 + (eigh-) \to 8$	$2 + (nine) \to 9$
$3 + (thir-) \to 3$	$3 + (four) \to 4$	$3 + (fif-) \to 5$
$3 + (six) \to 6$	$3 + (seven) \to 7$	$3 + (eigh-) \to 8$
$3 + (nine) \to 9$		

Table 3.2: Transition table for number convertion.

Figure 3.1: Top-down Transducer for Number Conversion

3.2.2 Extended Top-down Tree Transducers

Example 10. *The Extended top-down tree transducer for Arabic to English Convertion*

3.2.3 Extended Multi Bottom-up Tree Transducers

Example 11. *The Extended Multi Botttom-up tree transducer for Copying*

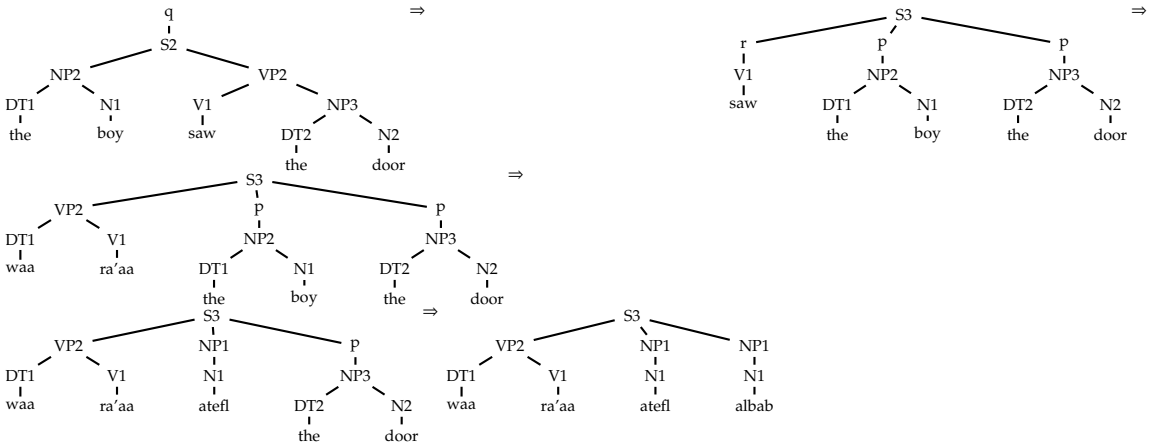


Figure 3.2: Extended Top-down Transducer for Arabic to English Translation

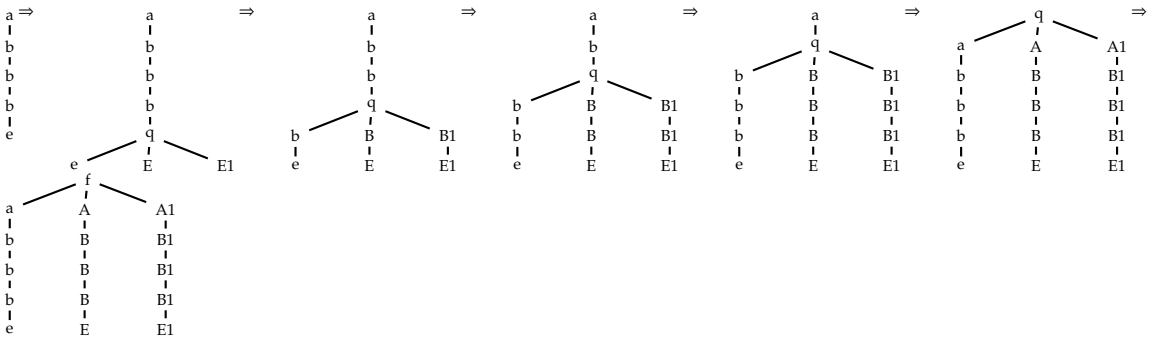


Figure 3.3: Extended Multi Bottom-up Transducer for Copying

$q(S2(x1, VP2(x2, x3))) \rightarrow S3(r(x2), p(x1), p(x3))$ $\quad r(V1(saw)) \rightarrow VP2(DT1(waa), V1(ra'aa))$
$p(NP2(DT1(the), N1(boy))) \rightarrow NP1(N1(atefl))$ $\quad p(NP3(DT2(the), N2(door))) \rightarrow NP1(N1(albab))$

Table 3.3: Transition table for Arabic to English Convertion.

$a(q(x1, x2, x3)) \rightarrow q(a(x1), A(x2), A1(x3))$ $\quad b(q(x1, x2, x3)) \rightarrow q(b(x1), B(x2), B1(x3))$
$q(x1, x2, x3) \rightarrow f(x1, x2, x3)$ $\quad e \rightarrow q(e, E, E1)$

Table 3.4: Transition table for Copying.

3.3 Learning of Finite State Automata

CHAPTER 4

WORK DONE

4.1 Tree Insertion Systems

4.1.1 Introduction

Similarly a tree, say t can be inserted as a sub-tree of another tree, say T based on some context. The insertion may be *fixed arity* or *variable arity*. Here arity refers to the arity of nodes in tree T. In this paper we consider the insertion of trees into trees and call it as *tree insertion system*.

Definition 6. *The* blocked tree insertion system *is a tuple* $\Gamma = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

- Σ *is a finite set of* ranked alphabets.
- $\mathcal{A} = \{A_1 \cup A_2 \cup \cdots \cup A_m\}$, *where each* $A_i, 1 \leq i \leq m$ *is a finite set of axioms. With each set* A_i *is associated a* **flag** F_i *which is a triple* $[x_i, y_i, z_i]$, *where* $x_i, y_i, z_i \in \{-1, 0, 1, \cdots, k\}$, *for some fixed k, are integers, which plays some role in language generation unless* $x_i = y_i = z_i = -1$. *(For each insertion from* A_i, *the* x_i *value gets incremented if* $x_i \leq y_i$ *and the* x_i *value gets decremented if* $x_i > y_i$. *The* x_i *will be set to* z_i *if one insertion from* A_i *happens when* $x_i = y_i$. *The tree insertion system is said to be* **stable**, *if* $x_i = y_i$ *for all flags with* $x_i \leq y_i$ *initially and* $x_i \neq y_i$ *for all flags with* $x_i > y_i$ *initially.)*

- $\mathcal{A}' \subseteq \mathcal{A}$ *is a finite set of* initial axioms.
- $R = \{r_1, r_2, \cdots, r_n\}$ *is a finite set of* insertion rules

  Each r_i*, for* $1 \leq i \leq n$ *is of the form* $(\chi, C_1, C_2, \cdots, C_p)$ *where,*
  - $\chi = (root, left, right)$ *which represents a context.*
    - *root is any node in the tree*
    - $left$ *is* i^{th} *child of root.*
    - *right is* $(i+1)^{th}$ *child of root.* $0 \leq i \leq arity(root)$ *and* $p \leq arity(root)$ *(*$-$ *checks for the absence of a child).*
  - $C_i = (X, rt', k), 1 \leq i \leq p, X \in \mathcal{A}$
    - *rt' is the root of the tree to be attached.*
    - *k is the position at which rt' is to get attached.* $1 \leq k \leq arity(root)$ *and it is between the nodes* $left$ *and right.*

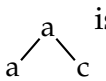
  As examples, $\chi = (a, b, c)$ denotes a node with label a having a node with label b as i^{th} child and a node with label c as $(i+1)^{th}$ child for $1 \leq i < arity(a)$.
  $\chi = (a, -, -)$ denotes a leaf node with label a.

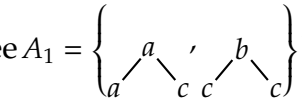
The **derivation step** is described as follows.

If $r = (\chi, C_1, C_2, \ldots, C_p)$ is a rule with $\chi = (a, b, c)$ and $C_i = (X, d, k)$ where X is an axiom with trees having root with label d and t is a tree with root p(domain) having label a. Then $t \Rightarrow t'$ by rule r where $t'(p.k)$ is the tree with root label d, $t'(p.i)$ is tree with root label b and $t'(p.(i+p+1))$ is tree with root label c.

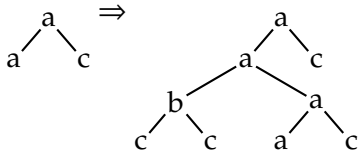
$\overset{*}{\Rightarrow}$ is the reflexive transitive closure of $\Rightarrow$.

The flag associated with X is also updated when r is applied. For each insertion from X, the x_i value gets incremented if $x_i \leq y_i$, the x_i value gets decremented if $x_i > y_i$ and the x_i will be set to z_i if $x_i = y_i$. We describe the derivation informally with an example.

Suppose 

is a tree $A_1 =$ 

be the axiom and $r_1 = ((a, -, -), (A_1, b, 1), (A_1, a, 2))$

be the insertion rule. Then by using r_1 

Here, at leaf node a, a subtree with root b is attached as the first child and another subtree with root a is attached as the second child.

The **language generated** by a tree insertion system Γ, represented by $L(\Gamma)$, is the set of trees, with each node having children exactly equal to its arity, derivable in Γ, when it is in *stable stae*, from an initial axiom, using rules of Γ.

$$L(\Gamma) = \left\{ t | S \overset{*}{\Rightarrow} t, S \text{ in some } A_i \in \mathcal{A}'. \text{ Each node of } t \text{ has children exactly as its arity and } \Gamma \text{ is in stable state} \right\}$$

Example 12. $L_{r_1} = \{a(b^i(h), c^j(h)), i, j \geq 1\}$

$\Gamma_{r_1} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1, A_2\}$

$F_1 = \{-1, -1, -1\}, F_2 = \{-1, -1, -1\}$ $\mathcal{A}' = \{A_1\}$, *where*

$A_1 = \left\{ \begin{array}{c} a \\ b \quad c \end{array} \text{ , } \begin{array}{ccc} & a & \\ b & & c \\ h & & h \end{array} \right\}$, $A_2 = \left\{ b \text{ , } c \text{ , } h \right\}$

$R = \{r_1, r_2, r_3, r_4\}$, *where*

$r_1 = (\chi_1, C_1)$, *where* $\chi_1 = (b, -)$ *and* $C_1 = (A_2, b, 1)$

$r_2 = (\chi_2, C_2)$, *where* $\chi_2 = (c, -)$ *and* $C_2 = (A_2, c, 1)$

$r_3 = (\chi_1, C_3)$ *where* $C_3 = (A_2, h, 1)$

$r_4 = (\chi_2, C_3)$

Example 13. $L_{r_2} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%2 = 0\}$

$\Gamma_{r_2} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1, A_2\}$,$F_1 = [-1, -1, -1]$, $F_2 = [-1, -1, -1]$, $\mathcal{A}' = \{A_1\}$, *where*

$A_1 = \left\{ a(b(b), a(c(b(h)), c(h))) \right\}$, $A_2 = \left\{ b(b), c, h \right\}$

$R = \{r_1, r_2, r_3, r_4\}$, *where*

$r_1 = (\chi_1, C_1)$, $r_2 = (\chi_2, C_2)$, $r_3 = (\chi_1, C_3)$, $r_4 = (\chi_2, C_3)$, *where*

$\chi_1 = (b, -)$, $\chi_2 = (c, -)$, $C_1 = (A_2, b, 1)$, $C_2 = (A_2, c, 1)$, $C_3 = (A_2, h, 1)$

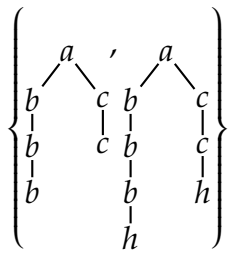
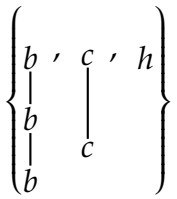
The *tree insertion* system Γ_{r_2}, with some minor changes in its axioms can be used to generate the following regular tree languages.

- $L_{r_3} = \{a(b^i(h), c^j(h)), i, j \geq 1, j\%2 = 0\}$
- $L_{r_4} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%2 \neq 0\}$
- $L_{r_5} = \{a(b^i(h), c^j(h)), i, j \geq 1, j\%2 \neq 0\}$
- $L_{r_6} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%2 = 0 \text{ and } j\%2 = 0\}$
- $L_{r_7} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%2 = 0 \text{ and } j\%2 \neq 0\}$
- $L_{r_8} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%2 \neq 0 \text{ and } j\%2 = 0\}$
- $L_{r_9} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%2 \neq 0 \text{ and } j\%2 \neq 0\}$

Example 14. $L_{r_5} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%3 = 0, j\%2 = 0\}$

$\Gamma_{r_5} = (F, \mathcal{A}, \mathcal{A}', R)$ *where,*

$F = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1^{[-1,-1,-1]}, A_2^{[-1,-1,-1]}\}$, $\mathcal{A}' = \{A_1\}$, *where*

$A_1 =$  , $A_2 =$ 

$R = \{r_1, r_2, r_3, r_4\}$, *where*

$r_1 = (\chi_1, C_1)$, *where* $\chi_1 = (b, -)$ *and* $C_1 = (A_2, b, 1)$

$r_2 = (\chi_2, C_2)$, *where* $\chi_2 = (c, -)$ *and* $C_2 = (A_2, c, 1)$

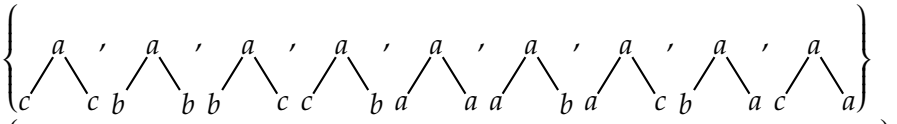
$r_3 = (\chi_1, C_3)$ *where* $C_3 = (A_2, h, 1)$

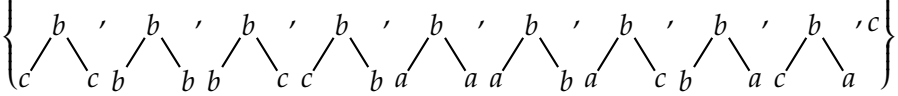
$r_4 = (\chi_2, C_3)$

Example 15. $L_{r_3} = \{t | t \text{ has a as root}\}$

$\Gamma_{r_3} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1, A_2\}$, $F_1 = [-1, -1, -1], F_2 = [-1, -1, -1]$, $\mathcal{A}' = \{A_1\}$, *where*

$A_1 =$ 

$A_2 =$ 

R contains rules,

$r_1 = (\chi_1, U_1, U_2)$, $r_2 = (\chi_2, U_1, U_2)$, *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -, -)$, $U_1 \in \{C_1, C_3\}, U_2 \in \{C_2, C_4\}$

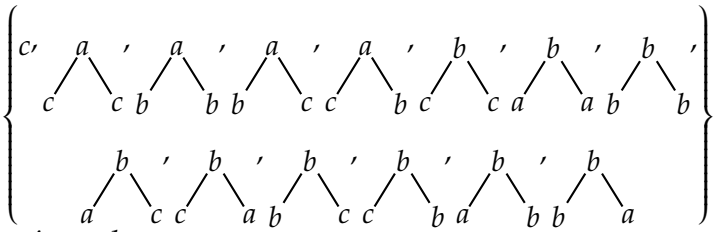
$C_1 = (A_1, a, 1), C_2 = (A_1, a, 2), C_3 = (A_2, V_1, 1), C_4 = (A_2, V_1, 2)$, $V_1 \in \{b, c\}$

Example 16. $L_{r_6} = \{t|$ *no a in t has an a as child.*$\}$

$\Gamma_{r_6} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1\}$

$F_1 = [-1, -1, -1]$ $\mathcal{A}' = \{A_1\}$

$A_1 =$ 

R contains rules,

$r_1 = (\chi_1, C_1, C_2)$,$r_2 = (\chi_2, C_3, C_4)$, *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -, -)$

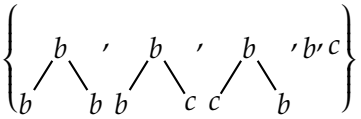
$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2), C_3 = (A_1, V_2, 1), C_4 = (A_1, V_2, 2)$

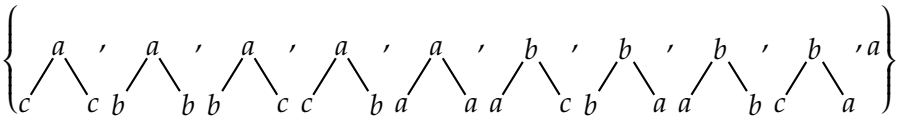
$V_1 \in \{c, b\}, V_2 \in \{a, b, c\}$

Example 17. $L_{r_4} = \{t|n_a(t)\%2 \neq 0\}$

$\Gamma_{r_4} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1, A_2\}$,$F_1 = [-1, -1, -1], F_2 = [0, 1, 2]$, $\mathcal{A}' = \{A_1, A_2\}$, *where*

$A_1 =$ 

$A_2 =$ 

R contains rules,

$r_1 = (\chi_1, U_1, U_2))$, $r_2 = (\chi_2, U_1, U_2)$, *where*

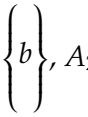
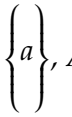
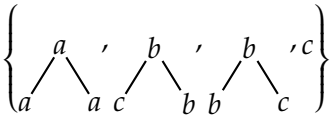
$\chi_1 = (a, -, -), \chi_2 = (b, -, -)$, $U_1 \in \{C_1, C_3\}, U_2 \in \{C_2, C_4\}$

$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2), C_3 = (A_2, V_2, 1), C_4 = (A_2, V_2, 2)$, $V_1 \in \{c, b\}, V_2 \in \{a, b\}$

Example 18. $L_{r_4} = \{t | (n_a(t)\%3) = 0$ *and* $(n_b(t)\%2) = 0\}$

$\Gamma_{r_4} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1, A_2, A_3\}$,$F_1 = [0, 0, 1], F_2 = [0, 0, 2], F_3 = [-1, -1, -1]$, $\mathcal{A}' = \{A_1, A_2\}$, *where*

$A_1 =$  , $A_2 =$  , $A_3 =$ 

R contains rules,

$r_1 = (\chi_1, U_1, U_2))$, $r_2 = (\chi_2, U_1, U_2)$, *where*

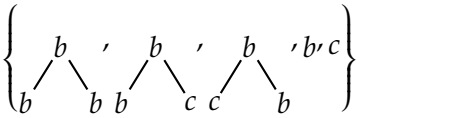
$\chi_1 = (a, -, -), \chi_2 = (b, -, -)$, $U_1 \in \{C_1, C_3\}, U_2 \in \{C_2, C_4\}$

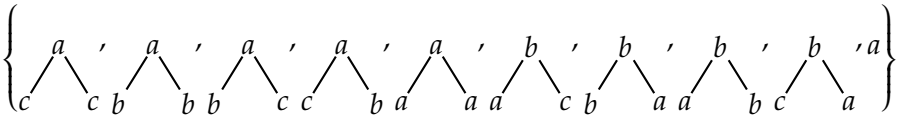
$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2), C_3 = (A_2, V_2, 1), C_4 = (A_2, V_2, 2)$, $V_1 \in \{c, b\}, V_2 \in \{a, b\}$

Example 19. $L_{r_4} = \{t | ((n_a(t) - 2)\%3) = 0\}$

$\Gamma_{r_4} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1, A_2\}$,$F_1 = [-1, -1, -1], F_2 = [0, 2, 4]$, $\mathcal{A}' = \{A_1, A_2\}$, *where*

$A_1 =$ 

$A_2 =$ 

R contains rules,

$r_1 = (\chi_1, U_1, U_2))$, $r_2 = (\chi_2, U_1, U_2)$, *where*

$\chi_1 = (a, -, -)$, $\chi_2 = (b, -, -)$, $U_1 \in \{C_1, C_3\}$, $U_2 \in \{C_2, C_4\}$

$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2), C_3 = (A_2, V_2, 1), C_4 = (A_2, V_2, 2)$, $V_1 \in \{c, b\}, V_2 \in \{a, b\}$

4.1.2 Equivalence with Regular Tree Grammar

Tree grammars are generating devices which is used for generating trees. The language generated by regular tree grammar is called *regular tree language*.

Tree automata are accepting devices for trees. Finite tree automata are generalizations of word automata. While a word automaton accepts a word, a tree automaton accepts a tree. Finite tree automata can be either bottom-up or top-down [4]. A *top-down tree automaton* starts its computation at the root of the tree and then simultaneously works down the paths of the tree level by level. A *bottom-up tree automaton* starts its computation in the leaves of the input tree and works its way up towards the root. A finite tree automaton can be either deterministic or non-deterministic. Deterministic top-down automata are strictly less expressive than non-deterministic top-down automata. For the bottom-up case, deterministic bottom-up tree automata are just as powerful, from the point of view of language

equivalence, as non-deterministic bottom-up tree automata. Non-deterministic top-down tree automata are equivalent to non-deterministic bottom-up tree automata.

4.1.3 Insertion System to Regular Tree Grammar

Method of conversion

For a given tree insertion system $\Gamma = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ we can construct an equivalent regular tree grammar $G = (N, T, S, \bar{P})$.

First we consider the simple case where the flag $F_i = [-1, -1, -1], \forall A_i \in \mathcal{A}$

- $T = \Sigma$
- N contains the start symbol S initially and more symbols of the form $M', M^{2'}, \cdots, M^{k'}$ for some fixed k, are added to N as we proceed to define the rules.
- $\forall A_i \in \mathcal{A}'$, if $t \in A_i$ with root with label p having arity $m > 0$ and children with label $p_1, p_2, \cdots, p_m$, then $S \to p(P_1, P_2, \cdots, P_m)$ will be a production rule and if there is a node in t with label q and arity r having children with labels $q_1, q_2, \cdots, q_r$, then $Q \to q(Y_1, Y_2, \cdots, Y_r)$ will be a production rule where $Y_i = Q_i, \forall q_i$ with label not equal to q and $Y_i = Q_i', \forall q_i$ with label equal to q. $Q_i' \to q(Y_1, Y_2, \cdots, Y_r)$ with $Y_i = Q_i^{2'}, \forall Y_i = Q_i$.

  In general $Q_i^{n'} \to q(Y_1, Y_2, \cdots, Y_r)$ with $Y_i = Q_i^{(n+1)'}, \forall Y_i = Q_i^{n'}$.

  If $m = 0$, $S \to P$ is a production rule.
- $\forall r_i \in R, r_i = (\chi, C_1, C_2, \cdots, C_k)$ where $\chi = (p, left, right)$, $C_j = (X_j, rt_j, k'), \forall j \leq k$ with $rt_j \in \Sigma, X_j \in \mathcal{A}, P^{n'} \to p(RT_j)$, where RT_j is the non terminal corresponding to rt_j, is a production rule.
- $\forall t \in \mathcal{A} - \mathcal{A}'$ with root node having label p and arity 0, then $P \to p$ is a production rule.
- N will includes all such P_i , Q_i and Q_i'.

It can easily be proved by induction that $L(G) = L(\Gamma)$.

Example 20. $L_{r_1} = \{a(b^i(h), c^j(h)), i, j \geq 1\}$

$\Gamma_{r_1} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1, A_2\}$

$F_1 = \{-1, -1, -1\}, F_2 = \{-1, -1, -1\}$ $\mathcal{A}' = \{A_1\}$, *where*

$A_1 = \left\{ \begin{array}{ccccc} a & , & a & & \\ b \diagup \diagdown c & & b \diagup \diagdown & & c \\ & & h & & h \end{array} \right\}$, $A_2 = \left\{ \begin{array}{ccccc} b & , & c & , & h \end{array} \right\}$

$R = \{r_1, r_2, r_3, r_4\}$, *where*

$r_1 = (\chi_1, C_1)$, *where* $\chi_1 = (b, -)$ *and* $C_1 = (A_2, b, 1)$

$r_2 = (\chi_2, C_2)$, *where* $\chi_2 = (c, -)$ *and* $C_2 = (A_2, c, 1)$

$r_3 = (\chi_1, C_3)$ *where* $C_3 = (A_2, h, 1)$

$r_4 = (\chi_2, C_3)$

The production rules are

$\bar{P} = \{S \rightarrow a(B, C), B \rightarrow b(B)|b(H), C \rightarrow c(C)|c(H), H \rightarrow h\}$

The regular tree grammar corresponding to Γ_{r_2} *is* $G_{r_2} = (\{S, B, C, H\}, \{a, b, c, h\}, S, \bar{P})$

Example 21. $L_{r_2} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%2 = 0\}$

$\Gamma_{r_2} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1, A_2\}$,$F_1 = [-1, -1, -1], F_2 = [-1, -1, -1]$, $\mathcal{A}' = \{A_1\}$, *where*

$A_1 = \left\{ \begin{array}{ccccc} a & , & a & & \\ b \diagup \diagdown c & & b & & c \\ | & & | & & | \\ b & & b & & h \\ & & | & & \\ & & h & & \end{array} \right\}$, $A_2 = \left\{ \begin{array}{ccccc} b & , & c & , & h \\ | & & & & \\ b & & & & \end{array} \right\}$

$R = \{r_1, r_2, r_3, r_4\}$, *where*

$r_1 = (\chi_1, C_1)$, $r_2 = (\chi_2, C_2)$, $r_3 = (\chi_1, C_3)$, $r_4 = (\chi_2, C_3)$, *where*

$\chi_1 = (b, -)$, $\chi_2 = (c, -)$, $C_1 = (A_2, b, 1)$, $C_2 = (A_2, c, 1)$, $C_3 = (A_2, h, 1)$

The production rules are

$\bar{P} = \{S \to a(B, C), B \to b(B'), B' \to b(B)|b(H), C \to c(C)|c(H), H \to h\}$

The regular tree grammar corresponding to Γ_{r_2} *is* $G_{r_2} = (\{S, B, B', C, H\}, \{a, b, c, h\}, S, \bar{P})$

Example 22. $L_{r_5} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%3 = 0, j\%2 = 0\}$

$\Gamma_{r_5} = (F, \mathcal{A}, \mathcal{A}', R)$ *where,*

$F = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1^{[-1,-1,-1]}, A_2^{[-1,-1,-1]}\}$, $\mathcal{A}' = \{A_1\}$, *where*

$A_1 = \left\{ \begin{array}{ccccc} a & , & a & & \\ b \diagup \diagdown c & & b \diagup \diagdown c & & \\ b \quad c & & b \quad c & & \\ b & & b & & \\ & & h & & \end{array} \right\}$, $A_2 = \left\{ \begin{array}{ccccc} b & , & c & , & h \\ b & & c & & \\ b & & & & \end{array} \right\}$

$R = \{r_1, r_2, r_3, r_4\}$, *where*

$r_1 = (\chi_1, C_1)$, *where* $\chi_1 = (b, -)$ *and* $C_1 = (A_2, b, 1)$

$r_2 = (\chi_2, C_2)$, *where* $\chi_2 = (c, -)$ *and* $C_2 = (A_2, c, 1)$

$r_3 = (\chi_1, C_3)$ *where* $C_3 = (A_2, h, 1)$

$r_4 = (\chi_2, C_3)$

The production rules are

$\bar{P} = \{S \to a(B, C), B \to b(B'), B' \to b(B''), B'' \to b(B)|b(H), C \to c(C'), C' \to c(C)|c(H), H \to h\}$

The regular tree grammar corresponding to Γ_{r_2} *is* $G_{r_2} = (\{S, B, B', B'', C, H\}, \{a, b, c, h\}, S, \bar{P})$

The above case will not take care of languages like trees with node labels a, b, and c where number of $a's$ is odd. For such cases we give the construction below.

- $T = \Sigma$
- N contains the start symbol S initially and more symbols of the form $M', M^{2'}, \cdots, M^{k'}$ for some fixed k, are added to N as we proceed to define the rules.
- Let $AJ \subseteq \mathcal{A}'$, where for each $A_i \in AJ, x_i = y_i = z_i$ in F_i and $AI = (\mathcal{A}' - AJ)$, where for each $A_i \in AI$,$n_i = |x_i - y_i|$. (This is used to take care of some constraint on the number of a particular terminal symbol $\sigma_k \in \Sigma$).
- $\forall A_i \in \mathcal{A}$, if there is a node with label p arity 0 is in A_i, then $P \to p$ is a production rule.
- $\forall A_i \in AJ$
  If $\exists A_l \in AI$ with $(x_l \neq y_l)$ in F_l, $t \in A_i$ with root node with label p having arity m and children with label $p_1, p_2, \cdots, p_m$, where for some $j \leq m$, p_j is the label of root node of some $t' \in A_l$, then from $S \to p(P_1, P_2, \cdots, P_m)$ write j production rules, with $P_k = S, \forall k \leq j$.
  If $m = 0$, $S \to P$ will be a production rule.

  $\forall A'_l \in AI$, if $(x'_l = y'_l)$ in F'_l then $S \to p(P_1, P_2, \cdots, P_m)$ will also be a production rule.

  N will includes all such P_i.
- $\forall A_i \in AI$ with $n_i = 1$
  If $t \in A_i$ with root with label p having arity m and children with label $p_1, p_2, \cdots, p_m$, then $S \to p(P_1, P_2, \cdots, P_m)$ will be a production rule and if there is a node in t with label q and arity r having children with labels $q_1, q_2, \cdots, q_r$, then, $Q \to q(Q_1, Q_2, \cdots, Q_r)$ will be a production rule.
  If $m = 0$, $S \to P$ will be a production rule.

  $\forall r_i \in R, r_i = (\chi, C_1, C_2, \cdots, C_{k'})$ where $\chi = (p, left, right)$, $C_j = (X_j, rt_j, k'')$, $\forall j \leq k'$ with $rt_j \in \Sigma, X_j \in \mathcal{A}$

If $\forall j \leq k', X_j \notin AI, P \rightarrow p(RT_1, RT_2, \cdots, RT_k)$ will be a production rule,

If $\exists X_j \in AI, j \leq k$,

- $P \rightarrow p(Y_1, Y_2, \cdots, Y_k)$ will be a production rule with $Y_j = RT_j, \forall j$ where $X_j \notin AI$ and $Y_j = RT'_j, \forall j$ where $X_j \in AI$.

  If $rt_j = \sigma_j$ where $X_j \in AI$ then $RT'_{\sigma_k} \rightarrow rt_j(Y_1, Y_2, \cdots, Y_{p'})$ will be a production rule with $Y_{p''} = S$ for some $p'' \leq p'$, where p' is the arity of rt_j.

  If $rt_j \neq \sigma_k$ where $X_j \in AI$ then $RT'_j \rightarrow rt_j(Y_1, Y_2, \cdots, Y_{p'})$ will be a production rule $Y_{p''} = RT'_{\sigma_k}$ for some $p'' \leq p'$.

N will includes all such P_i, Q_i, RT'_i and Y_i.

- $\forall A_i \in AI$ with $n_i \neq 1$
  If $t \in A_i$ with root with label $p \neq \sigma_i$ having arity m and children with label $p_1, p_2, \cdots, p_m$, with some $p_j = \sigma_k, j \leq m$ then $S \rightarrow p(P_1, P_2, \cdots, P_m)$, where both $P_j = P_{\sigma_k}$ and $P_j = S$ are production rules. Then $P_{\sigma_k} \rightarrow S$ and $P_{\sigma_k} \rightarrow p_{\sigma_k}(Y_1, Y_2, \cdots, Y_{m'})$ will be production rules where m' is the arity of node with label p_{σ_k} and for some $m'' \leq m', Y_{m''} = P'_{\sigma_k}$. $P'_{\sigma_k} \rightarrow p_{\sigma_k}(Y_1, Y_2, \cdots, Y_{m'})$ and for some $m'' \leq m', Y_{m''} = P^{2'}_{\sigma_k}$. In general $P^{q'}_{\sigma_k} \rightarrow p_{\sigma_k}(Y_1, Y_2, \cdots, Y_{m'})$ and for some $m'' \leq m', Y_{m''} = P^{(q+1)'}_{\sigma_k}$ and $P^{z'_i}_{\sigma_k} \rightarrow p_{\sigma_k}(Y_1, Y_2, \cdots, Y_{m'})$ and for some $m'' \leq m', Y_{m''} = S$.

  If $m = 0, S \rightarrow P$ will be a production rule.

  If $t \in A_i$ with root node with label $p = \sigma_k$ having arity m and children with label $p_1, p_2, \cdots, p_m$, with some $p_j, j \leq m$ is the label of root node of some $t' \in AI$, then $S \rightarrow p(P_1, P_2, \cdots, P_m)$ will be a production rule with $P_j = P'_j$ and $P'_j \rightarrow p_j(Y_1, Y_2, \cdots, Y_{m'})$ where m' is the arity of node with label p_j and for some $m'' \leq m', Y_{m''} = P'_{\sigma_k}$.

  $\forall r_i \in R, r_i = (\chi, C_1, C_2, \cdots, C'_k)$ where $\chi = (p, left, right)$, $C_j = (X_j, rt_j, k''), \forall j \leq k'$ with $rt_j \in \Sigma, X_j \in \mathcal{A}$

  If $\forall j, X_j \in AJ$, with $p \neq \sigma_k$ then $P \rightarrow p(RT_1, RT_2, \cdots, RT_m)$ will be a production rule where m is the arity of node with label p.

  If $\forall j, X_j \in AI$, if $p \neq \sigma_k$ then $P \rightarrow p(RT'_1, RT'_2, \cdots, RT'_m)$ will be a production rule where m is the arity of node with label p.

  If $p = \sigma_k$ then $P \rightarrow p(Y_1, Y_2, \cdots, Y_k)$ is a production rule with $Y_j = RT_j, \forall j$ where $X_j \in AJ$ $Y_j = RT'_j, \forall j$ where $X_j \in AI$.

  If $rt_j = \sigma_k$ where $X_j \in AI$ then $RT'_{\sigma_k} \rightarrow rt_j(Y_1, Y_2, \cdots, Y_{p'})$ will be a production rule with $Y_{p''} = RT^{2'}_{\sigma_k}$ for some $p'' \leq p'$, where p' is the arity of rt_j.

  In general, for some $p'' \leq p$, $RT^{(z_j-1)'}_{\sigma_k} \rightarrow rt_j(Y_1, Y_2, \cdots, Y_{p'})$, with $Y_{p''} = RT^{(z_i)'}_{\sigma_k}$, and $RT^{z'_i}_{\sigma_k} \rightarrow rt_j(Y_1, Y_2, \cdots, Y_{p'})$, with $Y_{p''} = S$.

If $rt_j \neq \sigma_k$ where $X_j \in AI$ then $RT'_j \rightarrow rt_j(Y_1, Y_2, \cdots, Y_{p'})$ will be a production rule $Y_{p''} = RT'_{\sigma_k}$ for some $p'' \leq p'$.
N will includes all such $P'_i s$, $RT'_i s$ and $Y'_i s$.

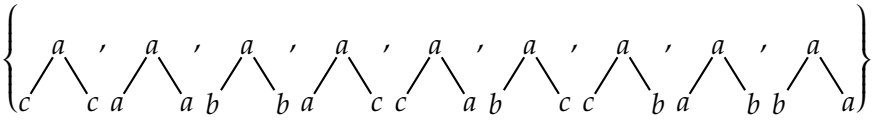
- If $\forall k, Y_k = RT'_j$ for some rt_j in $P \rightarrow p(Y_1, Y_2, \cdots, Y_r)$, then $P \rightarrow p(RT_1, RT_2, \cdots, RT_k)$ will be a production rule.

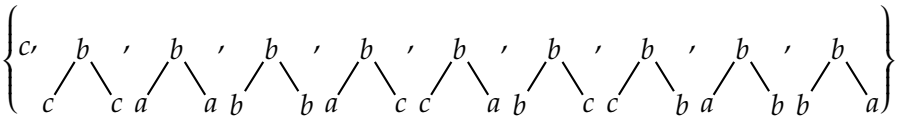
It can easily be proved by induction that $L(G) = L(\Gamma)$.

Example 23. $L_{r_3} = \{t | t \textit{ has a as root}\}$

$\Gamma_{r_3} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1, A_2\}$, $F_1 = [-1, -1, -1], F_2 = [-1, -1, -1]$, $\mathcal{A}' = \{A_1\}$

$A_1 =$ 

$A_2 =$ 

R contains rules,

$r_1 = (\chi_1, U_1, U_2)$, $r_2 = (\chi_2, U_1, U_2)$, *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -, -)$, $U_1 \in \{C_1, C_3\}, U_2 \in \{C_2, C_4\}$

$C_1 = (A_1, a, 1), C_2 = (A_1, a, 2), C_3 = (A_2, V_1, 1), C_4 = (A_2, V_1, 2)$

$V_1 \in \{b, c\}$

The production rules are

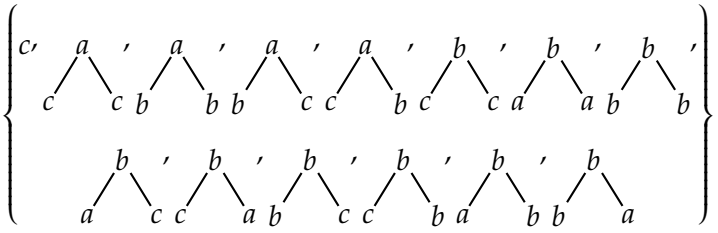
$$\begin{aligned} P = \{S &\rightarrow a(C,C)|a(B,B)|a(B,C)|a(C,B)|a(A,A)|a(A,B)|a(A,C)|a(B,A)|a(C,A) \\ A &\rightarrow a(A,B)|a(A,C)|a(C,C)|a(C,A)|a(C,B)|a(B,C)|a(B,A)|a(A,A)|a(B,B) \\ B &\rightarrow b(A,B)|b(A,C)|b(C,C)|b(C,A)|b(C,B)|b(B,C)|b(B,A)|b(A,A)|b(B,B) \\ C &\rightarrow c\} \end{aligned}$$

The regular tree grammar corresponding to Γ_{r_3} *is* $G_{r_3} = (\{S, A, B, C\}, \{a, b, c\}, S, P)$

Example 24. $L_{r_6} = \{t|$ *no a in t has an a as child*.$\}$

$\Gamma_{r_6} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1\}$, $F_1 = [-1, -1, -1]$ $\mathcal{A}' = \{A_1\}$

$A_1 =$ 

R contains rules,

$r_1 = (\chi_1, C_1, C_2)$,$r_2 = (\chi_2, C_3, C_4)$, *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -, -)$

$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2), C_3 = (A_1, V_2, 1), C_4 = (A_1, V_2, 2)$

$V_1 \in \{c, b\}, V_2 \in \{a, b, c\}$

The production rules are

$$
\begin{aligned}
P = \{S &\rightarrow a(C,C)|a(B,B)|a(B,C)|a(C,B)|b(C,C)|b(A,A)|b(B,B)|b(A,C)|b(A,A)|b(B,A)|b(A,B)|b(C,B)|b \\
A &\rightarrow a(C,C)|a(C,B)|a(B,C)|a(B,B) \\
B &\rightarrow b(A,B)|b(A,C)|b(C,C)|b(C,A)|b(C,B)|b(B,C)|b(B,A)|b(A,A)|b(B,B) \\
C &\rightarrow c\}
\end{aligned}
$$

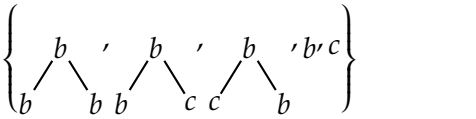
The regular tree grammar corresponding to Γ_{r_3} *is* $G_{r_3} = (\{S, A, B, C\}, \{a, b, c\}, S, P)$

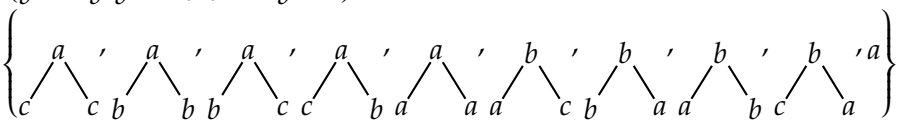
Example 25. $L_{r_4} = \{t | n_a(t)\%2 \neq 0\}$

$\Gamma_{r_4} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1, A_2\}$

$F_1 = [-1, -1, -1], F_2 = [0, 1, 0]$, $\mathcal{A}' = \{A_1, A_2\}$, *where*

$A_1 =$ 

$A_2 =$ 

R contains rules,

$r_1 = (\chi_1, U_1, U_2))$, $r_2 = (\chi_2, U_1, U_2)$, *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -, -)$, $U_1 \in \{C_1, C_3\}, U_2 \in \{C_2, C_4\}$

$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2), C_3 = (A_2, V_2, 1), C_4 = (A_2, V_2, 2)$

$V_1 \in \{c, b\}, V_2 \in \{a, b\}$

The production rules are

$$
\begin{aligned}
P = \{S &\rightarrow a(C,C)|a(B,B)|a(B,C)|a(C,B)|a(A,A)|b(A,C)|b(B,A)|b(A,B)|b(C,A)|A \\
&\quad b(S,B)|b(B,S)|b(S,C)|b(C,S) \\
A &\rightarrow a(C,C)|a(C,B)|a(B,C)|a(B,B)|a(C,A')|a(C,B')|a(B,A')|a(B,B')| \\
&\quad a(A',C)|a(A',B)|a(B',C)|a(B',B)|a(A',A')|a(A',B')|a(B',A')|a(B',B') \\
&\quad a(A,A)|a(B,B) \\
B &\rightarrow b(C,C)|b(C,B)|b(B,C)|b(B,B)|b(C,A')|b(C,B')|b(B,A')|b(B,B')| \\
&\quad b(A',C)|b(A',B)|b(B',C)|b(B',B)|b(A',A')|b(A',B')|b(B',A')|b(B',B') \\
&\quad b(A,A)|b(B,B) \\
A' &\rightarrow a(S,C)|a(S,B)|a(C,S)|a(B,S)|a(S,A')|a(A',S)|a(S,B')|a(B',S) \\
B' &\rightarrow b(A',C)|b(A',B)|b(C,A')|b(B,A')|b(A',A')|b(A,A)|b(B',A')|b(A',B') \\
C &\rightarrow c\}
\end{aligned}
$$

The regular tree grammar corresponding to Γ_{r_4} *is* $G_{r_4} = (\{S, A, A', B, B', C\}, \{a, b, c\}, S, P)$

Result 1: *Given a tree insertion system, we can construct an equivalent regular tree grammar.*

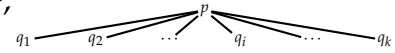
4.1.4 Finite state Bottom-up Tree Automata to Insertion System

Since there exists a bottom-up finite tree automata for accepting a regular tree language,it is enough to simulate that automata using the tree insertion system, to show the equivalence of tree insertion system and regular tree grammars.

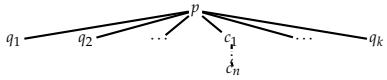
Method of conversion

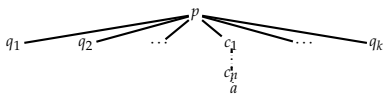
For a given deterministic bottom-up tree automata $D = (\Sigma', Q, Q_f, \Delta)$ we can construct a tree insertion system $\Gamma = (\Sigma, \mathcal{A}, \mathcal{A}', R)$, where

- $\Sigma = \Sigma'$
- If transitions are non-recursive

  For each transition of the form $p(q_1, q_2, \cdots, q_k) \rightarrow q_g$, where $q_1, q_2, \cdots, q_k, q_g \in Q, p \in \Sigma', k$ is the arity of node with label p and $q_g \in Q_f$, 

  will be in $\mathcal{A}'$

  $\forall q_i, 1 \leq i \leq k$, if $\{c_1(q_{i_1}) \rightarrow q_i, c_2(q_{i_2}) \rightarrow q_{i_1} \cdots c_n(q_{i_n}) \rightarrow q_{i_{n-1}}\} \in \Delta$ where $q_{i_n} = q_i$, $c_i \in \Sigma', 1 \leq i \leq n$ and $\forall i, j, q_{i_j} \in Q, 1 \leq j \leq n$,  will be in $\mathcal{A}'$

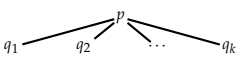
  If $\{c_1(q_{i_1}) \rightarrow q_i, c_2(q_{i_2}) \rightarrow q_{i_1} \cdots c_n(q_{i_n}) \rightarrow q_{i_{n-1}}, a \rightarrow q_{i_n}\} \in \Delta$ where $a, c_i \in \Sigma', 1 \leq i \leq n$ and $\forall i, j, q_{i_j} \in Q, 1 \leq j \leq n$,  will be in $\mathcal{A}'$

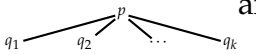
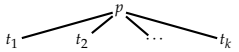
  For each transition of the form $p(q_1, q_2, \cdots, q_k) \rightarrow q_g$, where $q_1, q_2, \cdots, q_k, q_g \in Q$, $p \in \Sigma'$, k is the arity of node with label p and $q_g \in Q_f$ $\forall q_i, 1 \leq i \leq p$ if $\{c_1(q_{i_1}) \rightarrow q_i, c_2(q_{i_2}) \rightarrow q_{i_1} \cdots c_n(q_{i_n}) \rightarrow q_{i_{n-1}}\} \in \Delta$ where $q_{i_n} = q_i$, $c_i \in \Sigma', 1 \leq i \leq n$ and $\forall i, j, q_{i_j} \in Q, 1 \leq j \leq n$, $\begin{matrix} c_1 \\ \vdots \\ c_n \end{matrix}$ will be in $\mathcal{A} - \mathcal{A}'$

  If $\{c_1(q_{i_1}) \rightarrow q_i, c_2(q_{i_2}) \rightarrow q_{i_1} \cdots c_n(q_{i_n}) \rightarrow q_{i_{n-1}}, a \rightarrow q_{i_n}\} \in \Delta$ where $a, c_i \in \Sigma', 1 \leq i \leq n$ and $\forall i, j, q_{i_j} \in Q, 1 \leq j \leq n$, $\begin{matrix} c_1 \\ \vdots \\ c_n \\ a \end{matrix}$ will be in $\mathcal{A} - \mathcal{A}'$

  Set flag $F_i = [x_i, y_i, z_i]$ associated with each $A_i \in \mathcal{A}$ as [-1,-1,-1].
- If transitions are recursive

  Let AF and AN are two temporary axiom sets

For each transition of the form $p(q_1, q_2, \cdots, q_k) \rightarrow q_g$, where $q_1, q_2, \cdots, q_k, q_g \in Q$, $p \in \Sigma'$, k is the arity of node with label p,  will be in AF if $q_g \in Q_f$ and it will be in AN if $q_g \notin Q_f$.

Let $t =$  and $t' =$ 

$\forall t \in AF$, if all $q_1, q_2, \cdots, q_k \in Q_f$, then p will be in AF

$\forall t \in AN$, if all $q_1, q_2, \cdots, q_k \notin Q_f$, then p will be in AN

$\forall t \in AF$, t' will be in AF with t_i having the label of root node of some $t'' \in AF$, $\forall q_i \in Q_f$.

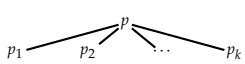
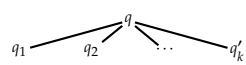
$\forall t \in AN$, t' will be in AN with t_i having the label of root node of some $t'' \in AN$, $\forall q_i \notin Q_f$.

$\forall t \in AF$, t' will be in AF with t_i having the label of root node of some $t'' \in AN$, $\forall q_i \notin Q_f$.

$\forall t \in AN$, t' will be in AN with t_i having the label of root node of some $t'' \in AF$, $\forall q_i \in Q_f$.

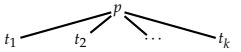
Set flag $F_{AN} = [x, y, z]$ associated with AN as [-1,-1,-1].

Set flag $F_{AF} = [x, y, z]$ associated with an axiom AF, which has a constraint on $\sigma_i \in \Sigma$, as follows.

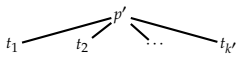
- $x = 0$.
- If $\exists t =$  $\in AF$, $k \geq 0$, if $p, p_1, p_2, \cdots, p_k \neq \sigma_i$ then $y_i = 0$.
- Else if $\exists t \in AF$, with n nodes of t have label σ_i, then $y_i = z_i = n$.
- If $t' =$  $\in AF$, $k' \geq 0$ having $m \geq n$ nodes of t are with label σ_i.
  - $z = (m - n) - 1$, if $y = 0$.
  - $z = (m - 1)$, if $y \neq 0$.

Now $\mathcal{A}' = AF$ and $\mathcal{A} = AF \cup AN$

- For each transitions of the form $a \rightarrow q_a \in \Delta$ where, $a \in \Sigma'$, $q_a \in Q$, a will be in $\mathcal{A}'$, if $q_a \in Q_f$ and will be in $\mathcal{A} - \mathcal{A}'$, if $q_a \notin Q_f$

- $\forall t \in \mathcal{A}$ which is of the form 

  $r = (\chi, C_1, C_2, \cdots, C_k) \in R$ where, $\chi = (p, -, \cdots, -)$ and $\forall i \leq k, C_i = (X, a, i)$ where, $X \in \mathcal{A}$, for some $t' \in X$,t' has a as root.

  And $\forall t_i, 1 \leq i \leq k$ which is of the form 

  $r = (\chi, C_1, C_2, \cdots, C_{k'}) \in R$ where, $\chi = (p', -, \cdots, -)$ and $\forall i \leq k', C_i = (X, a, i)$ where, $X \in \mathcal{A}$, for some $t' \in X, t'$ has a as root.

Example 26. $L_{r_1} = \{a(b^i(h), c^j(h)), i, j \geq 1\}$ *Consider the tree automata* $D_{r_1} = (Q, \Sigma', Q_f, \Delta)$ *which accepts the language* L_{r_1} *over the alphabet* $\Sigma' = \{a_2, b_1, c_1, h_0\}$ *where* $Q = \{q_a, q_b, q_c, q_h\}$, $Q_f = \{q_a\}$ *and* Δ *contains following transitions.*

$$h \to q_h \quad b(q_h) \to q_b \quad b(q_b) \to q_b \quad c(q_h) \to q_c \quad c(q_c) \to q_c \quad a(q_b, q_c) \to q_a$$

$$A_1 = \left\{ \begin{array}{ccc} a & , & a \\ b \diagup \diagdown c & & b \diagup \diagdown c \\ & & \vert \quad \vert \\ & & h \quad h \end{array} \right\}, A_2 = \left\{ c \ , \ \begin{array}{c} c \\ \vert \\ h \end{array} \ , \ b \ , \ \begin{array}{c} b \\ \vert \\ h \end{array} \ , \ h \right\}$$

R contains rules,

$r_1 = (\chi_1, C_1, C_2)$, $r_2 = (\chi_2, U_1)$, $r_3 = (\chi_3, U_2)$ *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -), \chi_3 = (c, -)$, $U_1 \in \{C_1, C_4\}, U_2 \in \{C_3, C_4\}$

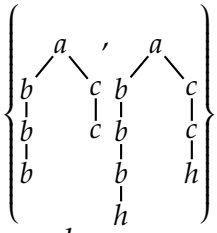
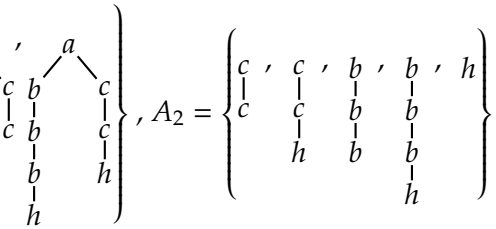
$C_1 = (A_2, b, 1), C_2 = (A_2, c, 2), C_3 = (A_2, c, 1), C_4 = (A_2, h, 1)$

So the tree insertion system equivalent to D_{r_1} *is* $\Gamma_{r_1} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1, A_2\}, F_1 = [-1, -1, -1], F_2 = [-1, -1, -1], \mathcal{A}' = \{A_1\}$

Example 27. $L_{r_5} = \{a(b^i(h), c^j(h)), i, j \geq 1, i\%3 = 0$ *and* $j\%2 = 0\}$ *Consider the tree automata* $D_{r_5} = (Q, \Sigma', Q_f, \Delta)$ *which accepts the language* L_{r_5} *over the alphabet* $\Sigma' = \{a_2, b_1, c_1, h_0\}$ *where* $Q = \{q_a, q_b, q_{b_1}, q_{b_2}, q_c, q_{c_1}, q_h\}$, $Q_f = \{q_a\}$ *and* Δ *contains following transitions.*

$$h \to q_h \quad b(q_h) \to q_{b_1} \quad b(q_{b_1}) \to q_{b_2} \quad b(q_{b_2}) \to q_b \quad b(q_b) \to q_{b_1}$$
$$c(q_h) \to q_{c_1} \quad c(q_{c_1}) \to q_c \quad c(q_c) \to q_{c_1} \quad a(q_b, q_c) \to q_a$$

$A_1 =$  , $A_2 =$ 

R contains rules,

$r_1 = (\chi_1, C_1, C_2)$, $r_2 = (\chi_2, U_1)$, $r_3 = (\chi_3, U_2)$ *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -), \chi_3 = (c, -)$, $U_1 \in \{C_1, C_4\}, U_2 \in \{C_3, C_4\}$

$C_1 = (A_2, b, 1), C_2 = (A_2, c, 2), C_3 = (A_2, c, 1), C_4 = (A_2, h, 1)$

So the tree insertion system equivalent to D_{r_5} *is* $\Gamma_{r_5} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1, A_2\}$,$F_1 = [-1, -1, -1], F_2 = [-1, -1, -1]$,$\mathcal{A}' = \{A_1\}$

Example 28. $L_{r_3} = \{t | t$ *has a as root*$\}$

Consider the tree automata $D_{r_3} = (Q, \Sigma', Q_f, \Delta)$ *which accepts the language* L_{r_3} *over the alphabet* $\Sigma' = \{a_2, b_2, c_0\}$ *where* $Q = \{q_a, q_b, q_c\}, Q_f = \{q_a\}$ *and* Δ *contains following transitions.*

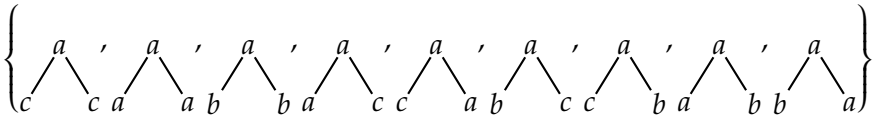
$c \to q_c$ $\quad b(q_c, q_c) \to q_b$ $\quad b(q_a, q_a) \to q_b$ $\quad b(q_b, q_b) \to q_b$ $\quad b(q_a, q_c) \to q_b$

$b(q_c, q_a) \to q_b$ $\quad b(q_b, q_c) \to q_b$ $\quad b(q_c, q_b) \to q_b$ $\quad b(q_a, q_b) \to q_b$ $\quad b(q_b, q_a) \to q_b$

$a(q_c, q_c) \to q_a$ $\quad a(q_a, q_a) \to q_a$ $\quad a(q_b, q_b) \to q_a$ $\quad a(q_a, q_c) \to q_a$ $\quad a(q_c, q_a) \to q_a$

$a(q_b, q_c) \to q_a$ $\quad a(q_c, q_b) \to q_a$ $\quad a(q_a, q_b) \to q_a$ $\quad a(q_b, q_a) \to q_a$

Create initial axiom A_1 *from transitions which directly leads to final state* q_a.

$A_1 =$ 

Create other axiom A_2 from transitions that leads to states which are the arguments of final state transitions.

$$A_2 = \left\{ c, \begin{array}{c} b \\ / \quad \backslash \\ c \quad c \end{array}, \begin{array}{c} b \\ / \quad \backslash \\ a \quad a \end{array}, \begin{array}{c} b \\ / \quad \backslash \\ b \quad b \end{array}, \begin{array}{c} b \\ / \quad \backslash \\ a \quad c \end{array}, \begin{array}{c} b \\ / \quad \backslash \\ c \quad a \end{array}, \begin{array}{c} b \\ / \quad \backslash \\ b \quad c \end{array}, \begin{array}{c} b \\ / \quad \backslash \\ c \quad b \end{array}, \begin{array}{c} b \\ / \quad \backslash \\ a \quad b \end{array}, \begin{array}{c} b \\ / \quad \backslash \\ b \quad a \end{array} \right\}$$

Since we could generate all trees in the axioms directly from the transitions itself, no need to set the tuple associated with the axioms.

Create rules for generating all trees in both A_1 and A_2.

R contains rules,

$r_1 = (\chi_1, U_1, U_2)$, $r_2 = (\chi_2, U_1, U_2)$, *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -, -), U_1 \in \{C_1, C_3\}, U_2 \in \{C_2, C_4\}$

$C_1 = (A_1, a, 1), C_2 = (A_1, a, 2), C_3 = (A_2, V_1, 1), C_4 = (A_2, V_1, 2)$

$V_1 \in \{b, c\}$

So the tree insertion system equivalent to D_{r_3} is $\Gamma_{r_3} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ where,

$$\Sigma = \{a_2, b_2, c_0\}, \mathcal{A} = \{A_1, A_2\}, F_1 = [-1, -1, -1], F_2 = [-1, -1, -1], \mathcal{A}' = \{A_1\}$$

Example 29. $L_{r_6} = \{t |$ *no a in t has an a as child.*$\}$

Consider the tree automata $D_{r_6} = (Q, \Sigma', Q_f, \Delta)$ which accepts the language L_{r_6} over the alphabet $\Sigma' = \{a_2, b_2, c_0\}$ where $Q = \{q_a, q_b, q_c\}, Q_f = \{q_a, q_b, q_c\}$ and Δ contains following transitions.

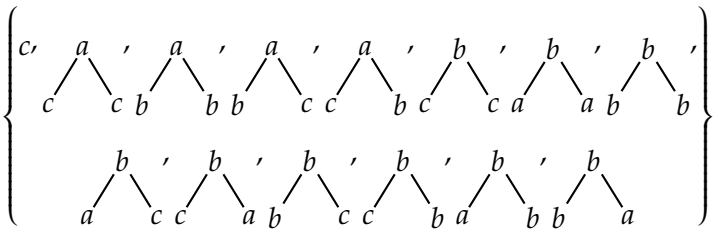
$c \to q_c$ $b(q_c, q_c) \to q_b$ $b(q_a, q_a) \to q_b$ $b(q_b, q_b) \to q_b$ $b(q_a, q_c) \to q_b$

$b(q_c, q_a) \to q_b$ $b(q_b, q_c) \to q_b$ $b(q_c, q_b) \to q_b$ $b(q_a, q_b) \to q_b$

$b(q_b, q_a) \to q_b$ $a(q_c, q_c) \to q_a$ $a(q_b, q_b) \to q_a$ $a(q_b, q_c) \to q_a$

$a(q_c, q_b) \to q_a$

Create initial axiom A_1 from transitions which directly leads to final state q_a.

$A_1 =$ 

There is no non-final state in this automata. So $\mathcal{A} = \mathcal{A}' = \{A_1\}$. Since we could generate all trees in the axioms directly from the transitions itself, no need to set the tuple associated with the axioms.

Create rules for generating all trees in A_1.

R contains rules,

$r_1 = (\chi_1, C_1, C_2)$,$r_2 = (\chi_2, C_3, C_4)$, *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -, -)$

$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2), C_3 = (A_1, V_2, 1), C_4 = (A_1, V_2, 2)$

$V_1 \in \{c, b\}, V_2 \in \{a, b, c\}$

So the tree insertion system equivalent to D_{r_6} is $\Gamma_{r_6} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ where,

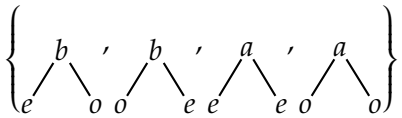
$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1\}$, $F_1 = [-1, -1, -1]$ $\mathcal{A}' = \{A_1\}$

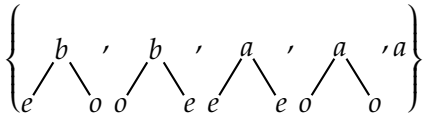
Example 30. $L_{r_4} = \{t | n_a(t)\%2 \neq 0\}$

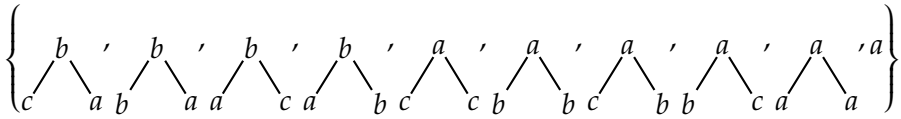
Consider the tree automata $D_{r_4} = (Q, \Sigma', Q_f, \Delta)$ *which accepts the language* L_{r_4} *over the alphabet* $\Sigma' = \{a_2, b_2, c_0\}$ *where* $Q = \{e, o\}, Q_f = \{o\}$ *and* Δ *contains following transitions.*

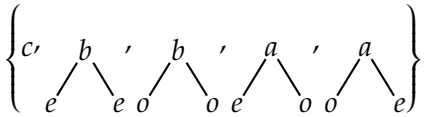
$$c \to e \quad b(e,e) \to e \quad b(o,o) \to e \quad a(e,o) \to e \quad a(o,e) \to e$$
$$b(e,o) \to o \quad b(o,e) \to o \quad a(e,e) \to o \quad a(o,o) \to o$$

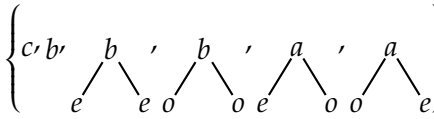
Here the transition shows a recursive behaviour and so it is not possible to generate all axioms for this language. Generate temporary axioms: AF, from final state transitions and AN, from non-final state transitions.

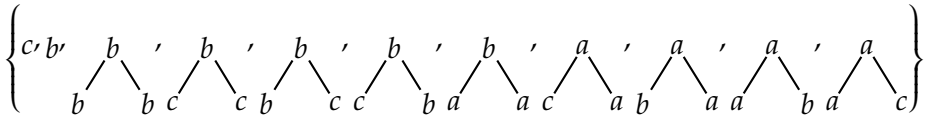
AF initially contains  *which leads to*





AN initially contains  *which leads to*

 *which then leads to*



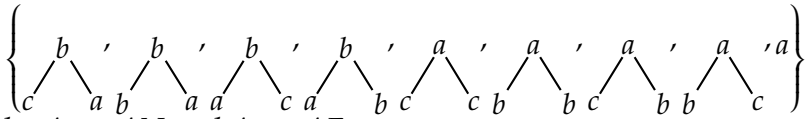
The two axiom sets AF and AN differs in the number of a's. So AF has some constraint on a.

Set the flag associated with AF.

Here $x = 0$.

Since all axioms in AF contains a, $n = 1$ *and one of the axiom contains 3 a's,* $m = 3$. *So* $y = 1$ *and* $z = 2$

In AF, split the axioms so that each axiom contains single a's.

$AF =$ 

Now let $A_1 = AN$ *and* $A_2 = AF$.

As both a and b can lead to final state directly, the insertion can be start from either A_1 *or* A_2. *Hence*

$\mathcal{A} = \{A_1, A_2\}$. $F_1 = [-1, -1, -1], F_2 = [0, 1, 2]$ $\mathcal{A}' = \{A_1, A_2\}$.

Create rules for generating all trees both in A_1 *and in* A_2.

R contains rules,

$r_1 = (\chi_1, U_1, U_2)$, $r_2 = (\chi_2, U_1, U_2)$, *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -, -), U_1 \in \{C_1, C_3\}, U_2 \in \{C_2, C_4\}$

$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2), C_3 = (A_2, V_2, 1), C_4 = (A_2, V_2, 2)$

$V_1 \in \{c, b\}, V_2 \in \{a, b\}$

So the tree insertion system equivalent to D_{r_4} *is* $\Gamma_{r_4} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

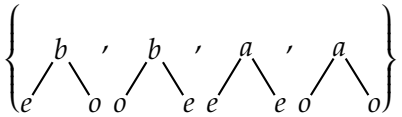
$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A}, \mathcal{A}'$ *and* R *as given above.*

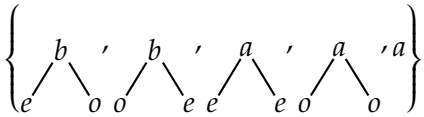
Example 31. $L_{r_8} = \{t | n_a(t)\%2 = 0\}$

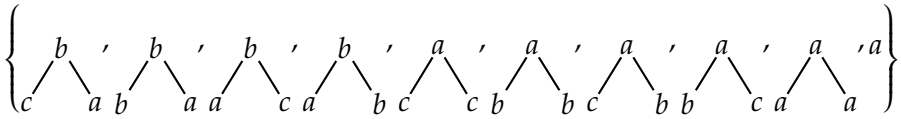
Consider the tree automata $D_{r_8} = (Q, \Sigma', Q_f, \Delta)$ *which accepts the language* L_{r_8} *over the alphabet* $\Sigma' = \{a_2, b_2, c_0\}$ *where* $Q = \{e, o\}, Q_f = \{e\}$ *and* Δ *contains following transitions.*

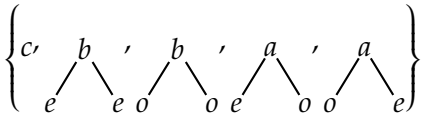
$$c \to e \quad b(e,e) \to e \quad b(o,o) \to e \quad a(e,o) \to e \quad a(o,e) \to e$$
$$b(e,o) \to o \quad b(o,e) \to o \quad a(e,e) \to o \quad a(o,o) \to o$$

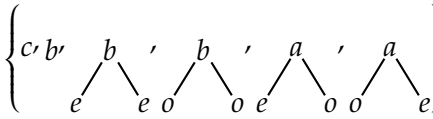
Here the transition shows a recursive behaviour and so it is not possible to generate all axioms for this language. Generate temporary axioms: AF, from final state transitions and AN, from non-final state transitions.

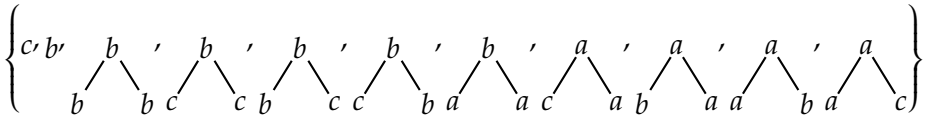
AN initially contains  *which leads to*

 *which then leads to*



AF initially contains  *which leads to*

 *which then leads to*



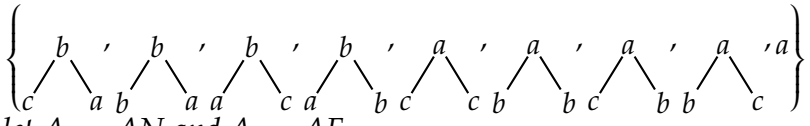
The two axiom sets AF and AN differs in the number of a's. So AF has some constraint on a.

Set the flag associated with AF.

Here $x = 0$.

Since AF contains some axioms without an a, $y = 0$ *and one of the axiom contains 2 a's,* $m = 2$. *So* $z = 1$

In AF, split the axioms so that each axiom contains single a's.

$AF =$ 

Now let $A_1 = AN$ *and* $A_2 = AF$.

As both a and b can lead to final state directly, the insertion can be start from either A_1 or A_2. Hence

$\mathcal{A} = \{A_1, A_2\}$. $F_1 = [-1, -1, -1], F_2 = [0, 0, 1]$ $\mathcal{A}' = \{A_1, A_2\}$.

Create rules for generating all trees both in A_1 and in A_2.

R contains rules,

$r_1 = (\chi_1, U_1, U_2)$, $r_2 = (\chi_2, U_1, U_2)$, *where*

$\chi_1 = (a, -, -), \chi_2 = (b, -, -), U_1 \in \{C_1, C_3\}, U_2 \in \{C_2, C_4\}$

$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2), C_3 = (A_2, V_2, 1), C_4 = (A_2, V_2, 2)$

$V_1 \in \{c, b\}, V_2 \in \{a, b\}$

So the tree insertion system equivalent to D_{r_8} is $\Gamma_{r_8} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ where,

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A}, \mathcal{A}'$ *and R as given above.*

Result 2: *Given a finite bottom-up tree automata, we can construct an equivalent tree insertion system.*

Claim: From Result 1 and Result 2, the generative powers of tree insertion systems and regular tree grammars are equal.

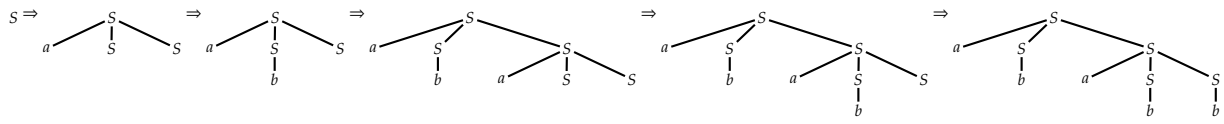
4.1.5 Extended Systems

An *extended tree insertion system* can be defined by defining two types of alphabets T and N with $\Sigma = T \cup N$ and additional restriction that the symbols in T have arity 0 and each symbol from N has a finite number of arities.

With this extended definition, the parse trees of a context-free grammar can be generated.

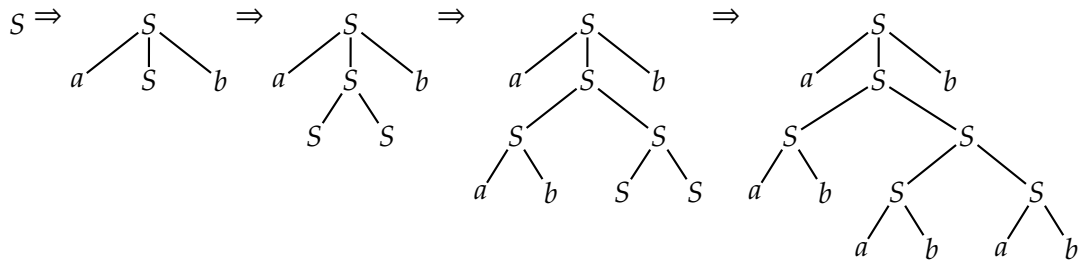
Example 32. *Consider the CFG* $G = (\{S\}, \{a, b\}, \{S \to aSS, S \to b\}, S)$. *The parse tree of* G *can be generated by the extended tree insertion system* $(\Sigma, \mathcal{A}, \mathcal{A}', R)$ *with* $\Sigma = N \cup T$, $N = \{S\}$, $T = \{a, b\}$, *arity of* S *is* $\{1, 3\}$. *Axioms are* $A_1 = \{S\}$ *and* $A_2 = \{a, b\}$ *with flags* $F_1 = [-1, -1, -1], F_2 = [-1, -1, -1]$, *where* $\mathcal{A}' = \{A_1\}$. *The rules are* $r_1 = (\chi_1, C_1, C_2, C_3)$ *and* $r_2 = (\chi_1, C_4)$, *where* $\chi_1 = (S, -, -), C_1 = (A_2, a, 1), C_2 = (A_1, S, 2), C_3 = (A_1, S, 3), C_4 = (A_2, b, 1)$.

As an example of derivation



Example 33. *Consider the CFG* $G = (\{S\}, \{a, b\}, \{S \to aSb, S \to SS\}, S)$. *The parse tree of* G *can be generated by the extended tree insertion system* $(\Sigma, \mathcal{A}, \mathcal{A}', R)$ *with* $\Sigma = N \cup T$, $N = \{S\}$, $T = \{a, b\}$, *arity of* S *is* $\{2, 3\}$. *Axioms are* $A_1 = \{S\}$ *and* $A_2 = \{a, b\}$ *with flags* $F_1 = [-1, -1, -1], F_2 = [-1, -1, -1]$, *where* $\mathcal{A}' = \{A_1\}$. *The rules are* $r_1 = (\chi_1, C_1, C_2, C_3)$, $r_2 = (\chi_1, C_1, C_2)$ *and* $r_3 = (\chi_1, C_4, C_5)$, *where* $\chi_1 = (S, -, -), C_1 = (A_2, a, 1), C_2 = (A_1, S, 2), C_3 = (A_2, b, 3), C_4 = (A_1, S, 1), C_5 = (A_1, S, 2)$.

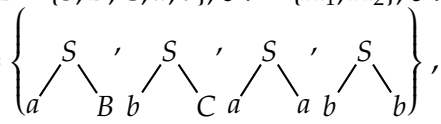
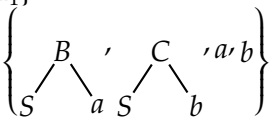
As an example of derivation



But the extended tree insertion system is not capable of generating parse trees for some context free grammars like $G = (S, B, C, a, b, S \to aB|bC|aa|bb, B \to Sa, C \to Sb, S)$.

The extended tree insertion system $\Gamma = (\Sigma, \mathcal{A}, \mathcal{A}', R)$, where

$\Sigma = \{S, B, C, a, b\}$, $\mathcal{A} = \{A_1, A_2\}$, $\mathcal{A}' = \{A_1\}$

$A_1 =$  , $A_2 =$ 

and $R = \{r_1, r_2, r_3\}$ where,

$r_1 = (\chi_1, C_1, C_2), r_2 = (\chi_2, C_3, C_4), r_3 = (\chi_3, C_3, C_5)$

$\chi_1 = (S, -, -), \chi_2 = (B, -, -), \chi_3 = (C, -, -)$

$C_1 = (A_2, U_1, 1), C_2 = (A_2, U_2, 2), C_3 = (A_1, S, 1), C_4 = (A_2, a, 2), C_5 = (A_2, b, 2)$

$U_1 \in \{(a, b\}, U_2 \in \{B, C, a, b\}$

can generate tree in figure **??** which is not generated by the given grammar G. By

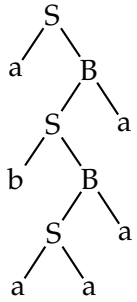


Figure 4.1: Parse Tree Generated

using a tree insertion system with replacement, we can generate the parse tree of the above grammar.

4.1.6 Tree Insertion Systems with Replacement

A *tree insertion system with replacement* can be defined by defining two types of alphabets T and N with $\Sigma = T \cup N$ and additional restriction that the symbols in T have arity 0 and each symbol from N has a finite number of arities. Here a subtree is inserted in to a tree by cutting the tree at an internal node and inserting the subtree with the cut subtree attaching to a leaf of the inserted tree.

Example 34. *The tree insertion system with replacement* $\Gamma = (\Sigma, \mathcal{A}, \mathcal{A}', R)$*, where*

$\Sigma = \{S, B, C, a, b\}$, $\mathcal{A} = \{A_1, A_2\}$, $\mathcal{A}' = \{A_1\}$

$A_1 = \left\{ \begin{array}{cccccccc} S & , & S & , & S & , & S \\ a\ \ B & & b\ \ C & & a\ \ a & & b\ \ b \end{array} \right\}$, $A_2 = \left\{ \begin{array}{ccccc} B & , & C & , & a, b \\ S\ \ a & & S\ \ b & & \end{array} \right\}$

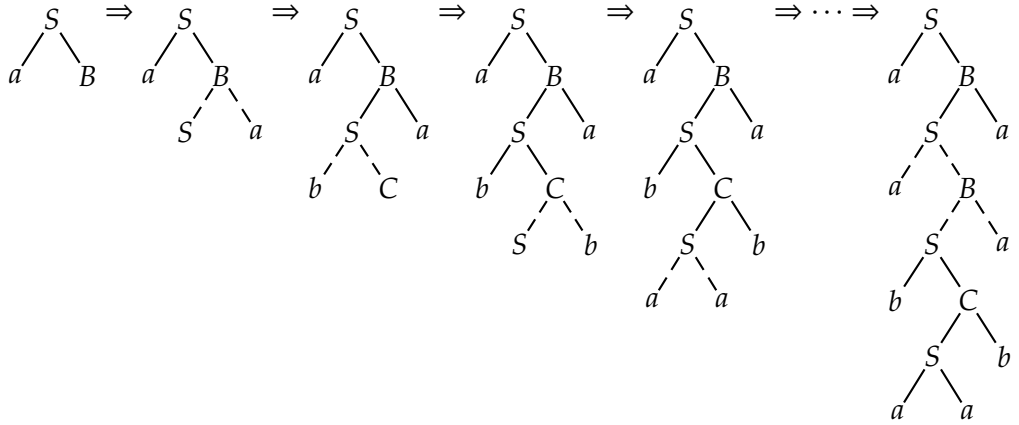
and $R = \{r_1, r_2, r_3\}$ *where,*

$r_1 = (\chi_1, C_1), r_2 = (\chi_2, C_2), r_3 = (\chi_3, C_3), r_4 = (\chi_4, C_3)$

$\chi_1 = (S, -, C), \chi_2 = (S, -, B), \chi_3 = (B, S, -), \chi_4 = (C, S, -)$

$C_1 = (A_2, C, 2), C_2 = (A_2, B, 2), C_3 = (A_1, S, 1)$

As an example of derivation



REFERENCES

[1] Akihiro Takaharai,. Takashi Yokomori, On the Computational power of insertion-deletion systems, Natural Computing 2. pp. 321-336. Kluwer Academic Publishers,(2003)

[2] Gheorghe Paun., Grzegorz Rozenberg., Arto Saloma,. DNA Computing, New Computing Paradigms. Springer, (1998)

[3] Lila Kari., Georghe Paun., Thierrin,G., Yu,S., At the crooroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. Proc. of 3rd DIMACS Workshop on DNA based Computing, Philadelphia, pp. 318-333, (1997)

[4] Hubert Comon., Max Dauchet Remi Gilleron.,Florent Jacquemard Denis Lugiez., Christof Loding., Sophie Tison Marc Tommasi., Tree Automata Techniques and Applications. Draft book; available electronically on http://www.grappa.univ-lille3.fr/tata, 2008.

[5] M B Siddardha and K. Krithivasan, Ambiguity in Insertion-Deletion Kolam Array Grammers , Journal of Combinatorics, Information and System Sciences, Vol. 33,Nos. 3-4, pages 323-337, 2008.

[6] M. Murata, D. Lee, M. Mani and K. Kawaguchi. Taxonomy of XML Schema Languages using Formal Language Theory, *ACM Trans.*

[7] S. Marcus, Contextual grammars, Rev. Roum. Math. Pures Appl., 14 (1969),15251534.

[8] M. Margenstern, Gh. Paun, Yu. Rogozhin, S. Verlan, Context-free insertion-deletion systems. Theoretical Computer Science, 330 (2005), 339348. Inter. Tech., 5(4):660-704, 2005.

[9] Sergey Verlan, On Minimal Context-Free Insertion-Deletion Systems,Journal of Automata, Languages and Combinatorics,2007, 12,317-328,1-2.

[10] Madhu Mutyam and Kamala Krithivasan and A. Siddhartha Reddy,On Characterizing Recursively Enumerable Languages by Insertion Grammars,Fundam. Inform.,64,1-4,2005,317-324,

4.2 Implementation and Application of Tree Transducers

4.2.1 Extended Top-down Tree Transducers

Algorithm

To check whether a top-down sequence in a tree is matching with left hand side of a production rule.

The function 'lhs' take a node sequence (in top-down manner) from the tree and check

all the transition function to find a matching.

for($k = 0; k < no\_items; k++$)

$root\_array[k] -> visited = 1;$

Take a node and check all transitions in the file.

If a match with a node, reset the $root\_array\_bl$ to start the operation and call function match

Copies the RHS of the matching transtion function with and without space

For each string in RHS call the function 'rhs' to process in the tree

If no match read the whole line so that we can search from next line again

Example 35. *The Extended top-down tree transducer for number convertion*

$q(crore2(x1, x2)) \rightarrow 10000000(q(x1), q(x2))$	$q(crore1(x1)) \rightarrow 10000000(q(x1), 100000(0, 0))$	$q(lakh2(x1, x2)) \rightarrow 100000(q(x1), q(x2))$
$q(lakh1(x1)) \rightarrow 100000(q(x1), 1000(0, 0))$	$q(thousand2(x1, x2)) \rightarrow 1000(q(x1), q(x2))$	$q(thousand1(x1)) \rightarrow 1000(q(x1), 100(0, 0))$
$q(hundred2(x1, x2)) \rightarrow 100(q(x1), q(x2))$	$q(hundred1(x1)) \rightarrow 100(q(x1), 10(0, 0))$	$q(-ty2(x1, x2)) \rightarrow 10(2 + (x1), 1 + (x2))$
$q(-ty1(x1)) \rightarrow 10(2 + (x1), 0)$	$q(-teen1(x1)) \rightarrow 10(1, 3 + (x1))$	$q(zero) \rightarrow 10(0, 0)$
$q(one) \rightarrow 10(0, 1)$	$q(two) \rightarrow 10(0, 2)$	$q(three) \rightarrow 10(0, 3)$
$q(four) \rightarrow 10(0, 4)$	$q(five) \rightarrow 10(0, 5)$	$q(six) \rightarrow 10(0, 6)$
$q(seven) \rightarrow 10(0, 7)$	$q(eight) \rightarrow 10(0, 8)$	$q(nine) \rightarrow 10(0, 9)$
$q(ten) \rightarrow 10(1, 0)$	$q(eleven) \rightarrow 10(1, 1)$	$q(twelve) \rightarrow 10(1, 2)$
$1 + (zero) \rightarrow 0$	$1 + (one) \rightarrow 1$	$1 + (two) \rightarrow 2$
$1 + (three) \rightarrow 3$	$1 + (four) \rightarrow 4$	$1 + (five) \rightarrow 5$
$1 + (six) \rightarrow 6$	$1 + (seven) \rightarrow 7$	$1 + (eight) \rightarrow 8$
$1 + (nine) \rightarrow 9$	$2 + (twen-) \rightarrow 2$	$2 + (thir-) \rightarrow 3$
$2 + (four) \rightarrow 4$	$2 + (fif-) \rightarrow 5$	$2 + (six) \rightarrow 6$
$2 + (seven) \rightarrow 7$	$2 + (eigh-) \rightarrow 8$	$2 + (nine) \rightarrow 9$
$3 + (thir-) \rightarrow 3$	$3 + (four) \rightarrow 4$	$3 + (fif-) \rightarrow 5$
$3 + (six) \rightarrow 6$	$3 + (seven) \rightarrow 7$	$3 + (eigh-) \rightarrow 8$
$3 + (nine) \rightarrow 9$		

Table 4.1: Transition table for number convertion.

Example 36. *The Extended top-down tree transducer for Malayalam to English Convertion*

$q(S(SP1, N1, SP2)) \rightarrow S(p(N1), r(SP2), s(SP1))$	$p(N1(aval)) \rightarrow NP(DT1(the), N3(girl))$	$r(SP2(V1, AV1)) \rightarrow SP2(t(AV1), u(V1))$
$t(AV1(aayirunnu)) \rightarrow AV1(was)$	$u(V1(paaduka)) \rightarrow V(singing)$	$s(SP1(N2, V2, AV2)) \rightarrow SP1(v(AV2), w(N2), y(V2))$
$v(AV2(appol)) \rightarrow AV2(when)$	$w(N2(avan)) \rightarrow NP1(DT2(the), N2(boy))$	$y(V2(vannu)) \rightarrow V2(came)$

Table 4.2: Transition table for Malayalam to English Convertion.

Figure 4.2: Extended Top-down Transducer for Number Conversion

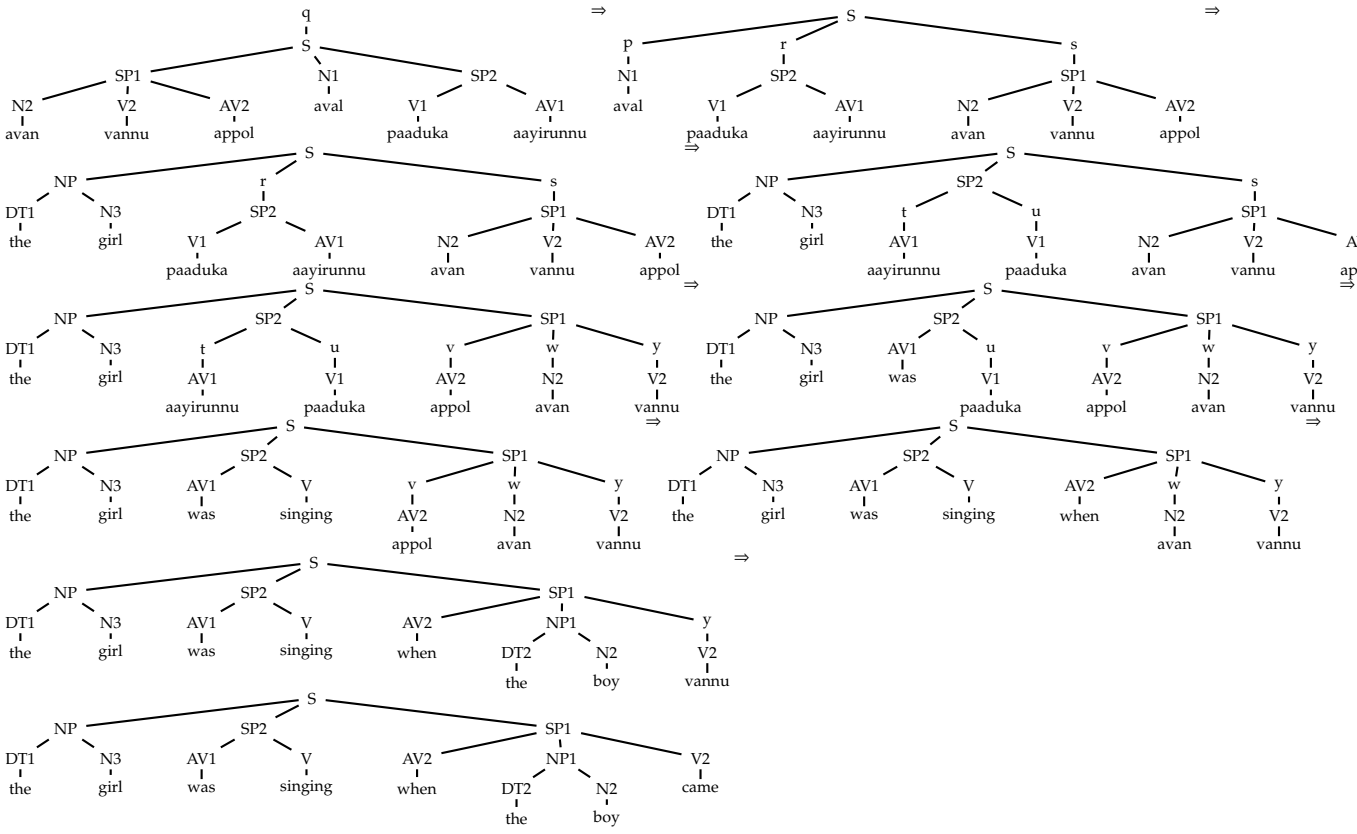


Figure 4.3: Extended Top-down Transducer for Malayalam to English Translation

4.2.2 Extended Multi Bottom-up Tree Transducers

Algorithm

$lhs_leaf(*tfp, *gfp);$

The 'lhs_leaf' function processes the leaves of the tree.

If a match with a node, reset the $root_array_gbl$to start the operation and call function match

$match(root_array_gbl[i], *fp, 2);$

Copies the first string from the LHS of the transition function

Search the $root_array_temp$ for the matching LHS (first string) and replace it with corresponding RHS

To count how many words are there in the RHS of a production rule.

For each string in RHS call the function 'rhs' to process in the tree

rhs(c);

If no match read the whole line so that we can search from next line again

$lhs_non_leaf(*tfp, *gfp);$

If a match with a node, reset the $root_array_gbl$ to start the operation and call function match

Copies the RHS of the matching transtion function with and without space

Storing the data of previous match

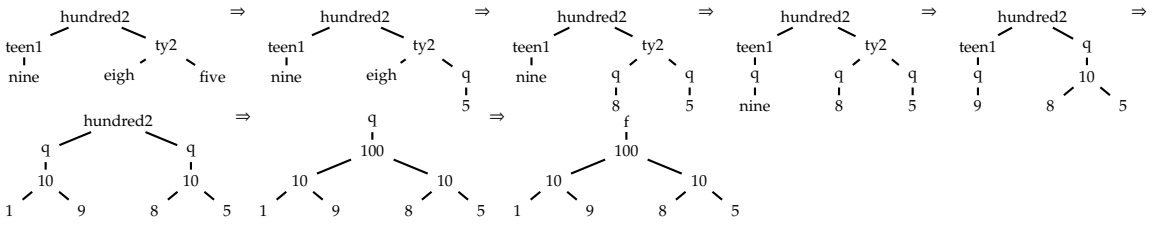


Figure 4.4: Extended Multi Bottom-up Transducer for Number Convertion

Example 37. *The Extended Multi Botttom-up tree transducer for Number Conversion.*

$nine \to q(9)$ $\quad eigh \to q(8)$ $\quad five \to q(5)$ $\quad -teen1(q(x1)) \to q(10(1, x1))$
$-ty2(q(x1), q(x2)) \to q(10(x1, x2))$ $\quad hundred2(q(x1), q(x2)) \to q(100(x1, x2))$ $\quad q(x1) \to f(x1)$

Table 4.3: Transition table for Number Conversion.

4.3 Learning of Distributed Tree Automata

4.3.1 Learning from Positive Samples

CHAPTER 5

OBSERVATIONS AND EXTENSIONS

5.1 Tree Insertion Systems

5.1.1 In Parse Trees

Applications in Parikh's Theorem

5.2 Blocked Tree Insertion System

We know that *tree insertion system* can be used to generate any regular tree languages. Since it has control only at the part of tree where current insertion taking place, it cannot be used for generating non-regular tree languages. But we can increase the generating power of the system by grouping the rules together to form a block and the resulting system is known as *blocked tree insertion system*.Here a block contains one or more rules of insertion. If a block starts working, all the rules in that block must execute before giving control to some other block.

Definition 7. *The* tree insertion system *is a tuple*

$\Gamma_B = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

- Σ *is a finite set of* ranked alphabets.
- $\mathcal{A} = \{A_1 \cup A_2 \cup \cdots \cup A_m\}$, *where each* $A_i, 1 \leq i \leq m$ *is a finite set of axioms. With each set* A_i *is associated a* **flag** F_i *which is a triple* $[x_i, y_i, z_i]$, *where* $x_i, y_i, z_i \in \{-1, 0, 1, \cdots, k\}$, *for some fixed* k, *are integers, which plays some role in language generation unless* $x_i = y_i = z_i = -1$. *(For each insertion from* A_i, *the* x_i *value gets incremented if* $x_i \leq y_i$ *and the* x_i *value gets decremented if* $x_i > y_i$. *The* x_i *will be set to* z_i *if one insertion from* A_i *happens when* $x_i = y_i$. *The tree insertion system is said to be* **stable**, *if* $x_i = y_i$ *for all flags with* $x_i \leq y_i$ *initially and* $x_i \neq y_i$ *for all flags with* $x_i > y_i$ *initially.)*
- $\mathcal{A}' \subseteq \mathcal{A}$ *is a finite set of* initial axioms.
- $R = \{B_1, B_2, \cdots, B_m\}$ *is a finite set of block of insertion rules*

  B_i*'s represents block of rules in the form* $\{r_1, r_2, \cdots, r_n\}$
  Each r_i, *for* $1 \leq i \leq n$ *is of the form* $(\chi, C_1, C_2, \cdots, C_p)$ *where,*
  - $\chi = (root, left, right)$ *which represents a context.*
    - *root is any node in the tree*
    - $left$ *is* i^{th} *child of root.*
    - *right is* $(i+1)^{th}$ *child of root.* $0 \leq i \leq arity(root)$ *and* $p \leq arity(root)$ *(*$-$ *checks for the absence of a child).*
  - $C_i = (X, rt', k), 1 \leq i \leq p, X \in \mathcal{A}$
    - rt' *is the root of the tree to be attached.*
    - k *is the position at which* rt' *is to get attached.* $1 \leq k \leq arity(root)$ *and it is between the nodes* $left$ *and right.*

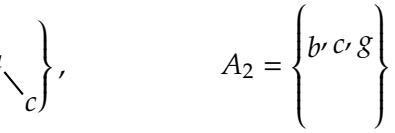
The **language generated** by a blocked tree insertion system Γ_B, represented by $L(\Gamma_B)$, is the set of trees, with each node having children exactly equal to its arity, derivable in Γ_B, when it is in *stable stae*, from an initial axiom, using rules of Γ_B.

$$L(\Gamma_B) = \left\{ \begin{array}{c} t | S \stackrel{*}{\Rightarrow} t, S \text{ in some } A_i \in \mathcal{A}'. \text{ Each node of } t \text{ has children exactly as its arity} \\ \text{and } \Gamma_B \text{ is in stable state} \end{array} \right\}$$

Example 38. $L_{nr_1} = \{a(b^i(g), c^i(g)), i \geq 1\}$

$\Gamma_{B_{nr_1}} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, g_0\}$, $\mathcal{A} = \{A_1, A_2\}$,$\mathcal{A}' = \{A_1\}$

$A_1 =$  , $A_2 =$ 

$R = \{B_1, B_2\}$, *where*

$B_1 = \{r_1, r_2\}$ *and* $B_2 = \{r_3, r_4\}$

$r_1 = (\chi_1, C_1)$, $r_2 = (\chi_2, C_2)$, $r_3 = (\chi_1, C_3)$, $r_4 = (\chi_2, C_3)$, *where*

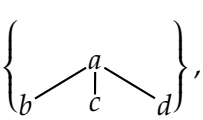
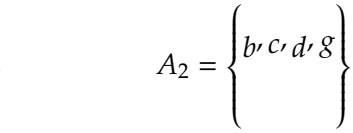
$\chi_1 = (b, -, -)$, $\chi_2 = (c, -, -)$

$C_1 = (A_2, b, 1)$, $C_2 = (A_2, c, 1)$, $C_3 = (A_2, g, 1)$

Example 39. $L_{nr_2} = \{a(b^i(g), c^i(g), d^i(g))\}, i \geq 1$

$\Gamma_{B_{nr_2}} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, d_1, g_0\}$, $\mathcal{A} = \{A_1, A_2\}$,$\mathcal{A}' = \{A_1\}$

$A_1 =$  , $A_2 =$ 

$R = \{B_1, B_2\}$, *where*

$B_1 = \{r_1, r_2, r_3\}$ *and* $B_2 = \{r_4, r_5, r_6\}$

$r_1 = (\chi_1, C_1)$, $r_2 = (\chi_2, C_2)$, $r_3 = (\chi_3, C_3)$, $r_4 = (\chi_1, C_4)$ $r_5 = (\chi_1, C_4)$, $r_6 = (\chi_1, C_4)$,*where*

$\chi_1 = (b, -, -)$, $\chi_2 = (c, -, -)$, $\chi_3 = (d, -, -)$

$C_1 = (A_2, b, 1)$, $C_2 = (A_2, c, 1)$, $C_3 = (A_2, d, 1)$, $C_4 = (A_2, g, 1)$

5.3 Insertion Tree Automata

Tree automata are accepting devices for trees. Finite tree automata are generalizations of word automata. While a word automaton accepts a word, a tree automaton accepts a tree. According to the manner in which the automaton runs on the input tree, finite tree automata can be either bottom-up or top-down. A *top-down tree automaton* starts its computation at the root of the tree and then simultaneously works down the paths of the tree level by level. The tree automaton accepts the tree if such a run can be defined. A *bottom-up tree automaton* starts its computation in the leaves of the input tree and works its way up towards the root. A finite tree automaton can be either deterministic or non-deterministic. This is an important issue since deterministic top-down automata are strictly less expressive than non-deterministic top-down automata. For the bottom-up case, deterministic bottom-up tree automata are just as powerful, from the point of view of language equivalence, as non-deterministic bottom-up tree automata. Non-deterministic top-down tree automata are equivalent to non-deterministic bottom-up tree automata.None of the above model is capable of accepting a *non-regular* tree language unless it consists more than one component.

In *insertion tree automata*, we are trying to combine the features of *fnite tree automata* and *tree insertion system* so that the resulting machine is capable of accepting both regular and non-regular tree languages.

5.3.1 As Accepting Device

We can design *tree automata* for accepting trees generated by a *tree insertion system*. More interesting thing is that, a tree automata with a single component can accept

a non-regular tree language if we apply insertion operation in the input tree before processing.

Example 40. $L_{nr_3} = \{a(b^i(g), c^j(g)), i, j \geq 1, |i - j| \leq 2\}$.

A tree automata with a single component cannot accept this language. But the distributed automata, given below, with 3 *components and working in* $= 1 -$ *mode is capable of accepting* L_{nr_3}.

$D_1 = (K, \Sigma, \{q_f\}, \Delta)$, *where*

$\Sigma = a, b, c, g, a \in \Sigma_2, b, c \in \Sigma_1, g \in \Sigma_0$.

- Component 1
  - $K_1 = \{q_b, q_g\}$
  - $\delta_1 = \{b(q_g) \rightarrow q_b, b(q_b) \rightarrow q_b\}$
- Component 2
  - $K_2 = \{q_g, q_c\}$
  - $\delta_2 = \{c(q_g) \rightarrow q_c, c(q_c) \rightarrow q_c\}$
- Component 3
  - $K_3 = \{q_b, q_c, q_g, q_f\}$
  - $\delta_3 = \{g \rightarrow q_g, a(q_b, q_c) \rightarrow q_f\}$

But if we do some insertion operations, using the insertion *system given below, in the input tree belongs to* L_{nr_3}, *it will get accepted by a single component tree automata.*

$\Gamma_{B_{nr_3}} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, g_0\}$, $\mathcal{A} = \{A_1, A_2\}$,$\mathcal{A}' = \{A_1\}$

$A_1 = \left\{ \begin{array}{ccc} & a & \\ b & & c \end{array} \right\}$, $A_2 = \left\{ b, c, g, x \right\}$

$R = \{B_1, B_2, B_3, B_4, B_5\}$, *where*

$B_1 = \{r_1, r_2\}$, $B_2 = \{r_1\}$, $B_3 = \{r2\}$, $B_4 = \{r_1, r_1\}$, $B_5 = \{r_2, r_2\}$

$r_1 = (\chi_1, x, arity(b+1))$, *where* $\chi_1 = (b, \bar{x}, -)$

$r_2 = (\chi_2, x, arity(c+1))$, *where* $\chi_2 = (c, \bar{x}, -)$

The resultant tree after insertion *operation will be accepted by the following tree automata with a* single component.

$A_1 = (Q, \Sigma, \{q_a\}, \Delta)$, *where*

$Q = \{q_g, q_a, q_b, q_c, q_x\}$, $\Sigma = \{a, b, c, g, x\}$ *and*

Δ *contains following rules.*

$g \to q_g$ $\quad x \to q_x$ $\quad a(q_b, q_c) \to q_a$

$b(q_g, q_x) \to q_b$ $\quad c(q_g, q_x) \to q_c$ $\quad b(q_b, q_x) \to q_b$ $\quad c(q_c, q_x) \to q_c$

Example 41. $L_{nr_4} = \{b(a(b^{2i}(g), c^{2j}(g)),\ i, j \geq 1, i = j \text{ or } i = j+1 \text{ or } j = i+1\}$.

A tree automata with a single component cannot accept this language. But the distributed automata, given below, with 3 *components and working in* $= 2 -$ *mode is capable of accepting* L_{nr_4}.

$D_2 = (K, \Sigma, \{q_f\}, \Delta)$, *where*

$\Sigma = a, b, c, g, a \in \Sigma_2, b, c \in \Sigma_1, g \in \Sigma_0.$

- Component 1
  - $K_1 = \{q_b, q_g\}$
  - $\delta_1 = \{b(q_g) \rightarrow q_b, b(q_b) \rightarrow q_b\}$
- Component 2
  - $K_2 = \{q_g, q_c\}$
  - $\delta_2 = \{c(q_g) \rightarrow q_c, c(q_c) \rightarrow q_c\}$
- Component 3
  - $K_3 = \{q_a, q_b, q_c, q_g, q_f\}$
  - $\delta_3 = \{g \rightarrow q_g, a(q_b, q_c) \rightarrow q_a, b(q_a) \rightarrow q_f\}$

But if we do some insertion operations, using the insertion *system given below, in the input tree belongs to* L_{nr_4}*, it will get accepted by a single component tree automata.*

$\Gamma_{B_{nr_4}} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ *where,*

$\Sigma = \{a_2, b_1, c_1, g_0\}$, $\mathcal{A} = \{A_1, A_2\}$, $\mathcal{A}' = \{A_1\}$

$A = \left\{ \begin{array}{c} b \\ | \\ a \\ b \diagup \diagdown c \\ | \quad | \\ b \quad c \end{array} \right\}$, $A = \left\{ b, c, g, x \right\}$

$R = \{B_1, B_2, B_3\}$, *where*

$B_1 = \{r_1, r_1, r_2, r_2\}$, $B_2 = \{r_1, r_1\}$, $B_3 = \{r_2, r_2\}$

$r_1 = (\chi_1, x, arity(b + 1))$, *where* $\chi_1 = (b, \bar{x}, -)$

$r_2 = (\chi_2, x, arity(c + 1))$, *where* $\chi_2 = (c, \bar{x}, -)$

The resultant tree after insertion *operation will be accepted by the following tree automata with a* single component.

$A_2 = (Q, \Sigma, \{q_f\}, \Delta)$, *where*

$Q = \{q_g, q_a, q_b, q_c, q_x, q_f\}$, $\Sigma = \{a_2, b_1, c_1, g_0, x_0\}$ *and*

Δ *contains following rules.*

$g \rightarrow q_g$ $\quad x \rightarrow q_x$ $\quad a(q_b, q_c) \rightarrow q_a$

$b(q_g, q_x) \rightarrow q_b$ $\quad c(q_g, q_x) \rightarrow q_c$ $\quad b(q_b, q_x) \rightarrow q_b$ $\quad c(q_c, q_x) \rightarrow q_c$

5.4 Tree Transducers

5.4.1 For Local Language Translation

5.5 Insertion System in Tree Transducers

5.5.1 Tree Transducer with Insertion System

I have implemented three different types of tree transducers *(Top-down, Extended Top-down and Extended Multi Bottom-up)* as a part of my M.Tech project and have shown some language translation like *english to malayalam, malayalam to english and arabic to english*.

But my implementation works only for fixed trees. We can give a sentence in the form of a tree, say *initial tree* and some transitions as inputs to the transducer. Based on the transitions given, the transducer converts given sentence to some other language.

For example a sentence *The machine runs the program* can be given in the form of a tree as

What happens if we need some changes in the input sentence, after generating the tree?

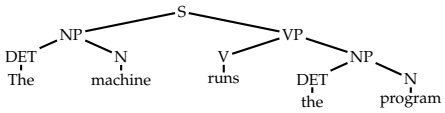


Figure 5.1: Initial Tree

One idea is to create a new tree for the new sentence. But it is not an efficient method, if there are only minor changes. Because a small change may not affect the whole tree which already generated. So reconstruction of the tree is not a good idea.

Here I would like to use the idea of *tree insdel system*. The idea is to construct a tree, say *auxilary trees* for the parts of sentence where changes occurs. This auxilaruy trees can be inserted to the initial trees using the concept of tree insdel system.

For example the tree structure of the sentence *The machine which compiles the language runs the program* is almost similar to the above tree. By making some insertions in the above tree we can generate tree for this sentence. To do that, first construct an auxilary tree for *which compiles the language* and insert it to the initial tree.

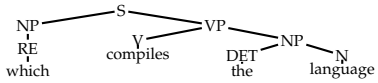


Figure 5.2: Auxilary Tree

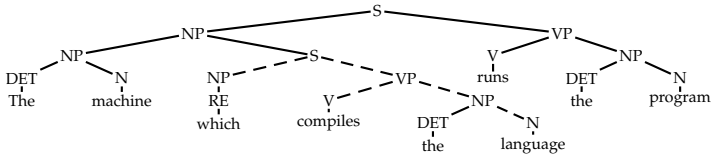


Figure 5.3: Initial Tree after Insertion of Auxilary Tree

The system mentined above has a lot of applications in language processing. So I would like to do some work on that.

CHAPTER 6

CONCLUSION

6.1 Conclusion

In this paper we defined tree insertion systems and showed that the generative power is the same as that of non deterministic bottom-up tree automata (or equivalently regular tree grammars). By defining an extended system, the power slightly increases. It should be noted that the insertion definition given here is a restricted one with the inserted tree occupying lower levels only. It may be interesting to widen the definition in such a way that a subtree is inserted in to a tree by cutting the tree at an internal node and inserting the subtree with the cut subtree attaching to a leaf of the inserted tree. Exploring the generative power of such systems with different contraints is being explored.

6.2 Future Work

It would also be interesting to consider *deletion system also*. Also there are some applications for tree transducers in XML, since it has a tree structure.

APPENDIX A

A SAMPLE APPENDIX

Just put in text as you would into any chapter with sections and whatnot. Thats the end of it.

Publications

i. S. M. Narayanamurthy and B. Ravindran (2007). Efficiently Exploiting Symmetries in Real Time Dynamic Programming. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2556–2561.

REFERENCES

[1] **Amarel, S.**, On representations of problems of reasoning about actions. *In* **D. Michie** (ed.), *Machine Intelligence 3*, volume 3. Elsevier/North-Holland, Amsterdam, London, New York, 1968, 131–171.

[2] **Barto, A. G.**, **S. J. Bradtke**, and **S. P. Singh** (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, **72**, 81–138.

[3] **Bellman, R. E.**, *Dynamic Programming*. Princeton University Press, 1957.

[4] **Crawford, J.** (1992). A theoretical analysis of reasoning by symmetry in first-order logic. URL `citeseer.ist.psu.edu/crawford92theoretical.html`.

[5] **Griffiths, D. F.** and **D. J. Higham**, *Learning LaTeX*. SIAM, 1997.

[6] **Knoblock, C. A.**, Learning abstraction hierarchies for problem solving. *In* **T. Dietterich** and **W. Swartout** (eds.), *Proceedings of the Eighth National Conference on Artificial Intelligence*. AAAI Press, Menlo Park, California, 1990. URL `citeseer.ist.psu.edu/knoblock90learning.html`.

[7] **Kopka, H.** and **P. W. Daly**, *Guide to LaTeX (4th Edition)*. Addison-Wesley Professional, 2003.

[8] **Lamport, L.**, *LaTeX: A Document Preparation System (2nd Edition)*. Addison-Wesley Professional, 1994.

[9] **Manning, J. B.** (1990). *Geometric symmetry in graphs*. Ph.D. thesis, Purdue University.

[10] **Ravindran, B.** and **A. G. Barto** (2001). Symmetries and model minimization of markov decision processes. Technical report, University of Massachusetts, Amherst.