Thue Systems for Pattern Recognition

John Abela

Department of Computer Science and AI,
University of Malta

Abstract. This report presents a synoptic overview of Thue Systems. Thue Systems were introduced in the early 1900s by the Norwegian mathematician and logician Axel Thue. In this report the author suggests ways in which such systems can be used in pattern recognition.

1 Introduction

Many problems in pattern recognition and machine learning can, albeit not always in a straightforward manner, reduced to the simple procedure of testing for formal language membership or to grammar induction. If patterns can be represented as strings, or for that matter tree or graphs, one can test for membership of a class simply by testing whether a given pattern, encoded as a string, is a member of the language consisting of all string encodings of the patterns in a class. Applications in pattern recognition that fall under this category include speech recognition, ECG signals, sonar signal, OCR, ECG patterns, picture languages, and many others [6, 7]. Learning in this case becomes simply a case of grammar induction. Learning grammars, for reasons we shall not discuss here, is by no means an easy task [5]. Some formal languages can be described by a set of re-writing rules that play the role of a grammar, i.e they can be used to specify, generate, or recognize the strings in the language. Before proceeding any further we present some simple definitions.

2 Reduction Systems

The Norwegian mathematician and logician Axel Thue [12] considered the following problem: Suppose one is given a set of objects and a set of rules (or transformations) that when applied to a given object yield another object. Now suppose one is given two objects x and y. Can x be transformed into y? Is there perhaps another object z such that both x and y can be transformed into z?

This problem became known as the *word problem*. Thue published some preliminary results about strings over a finite alphabet. Although Thue restricted his attention to strings he did suggest, however, that one might be able to generalize this approach to more structured combinatorial objects such as trees, graphs, and other structured objects. This generalization was later developed and the result was *reduction systems*. Reduction systems are so called because they describe, in an abstract way, how objects are transformed into other objects that are, by some criterion, *simpler* or *more general*. Reduction systems fall under the general name of *replacement systems* [8]. Replacement systems are now an important area of research in computer science and have applications in automated deduction, computer algebra, formal language theory, symbolic computation, theorem proving, program optimization, and machine learning.

The reader is referred to [3, Chapter 1] for an exposition.

Definition 21 (Reduction System) [3, page 10]
Let S be a set and let $\rightarrow$ be a binary relation on S. Then:

1. *The structure $R = (S, \rightarrow)$ is called a **reduction system**. The relation $\rightarrow$ is called the **reduction relation**. For any $x, y \in S$, if $x \rightarrow y$ then we say that x **reduces** to y.*
2. *If $x \in S$ and there exists no $y \in S$ such that $x \rightarrow y$, then x is called **irreducible**. The set of all elements of S that are irreducible with respect to $\rightarrow$ is denoted by **IRR(R)**.*
3. *For any $x, y \in S$, if $x \overset{*}{\longleftrightarrow} y$ and y is irreducible, then we say that y is a **normal form** of x. Recall that $\overset{*}{\longleftrightarrow}$ is the reflexive, symmetric, and transitive closure of $\rightarrow$.*
4. *For any $x \in S$, we denote by $\Downarrow_{\mathbf{R}}(\mathbf{x}) \overset{\text{def}}{=} \{y \in S \,|\, x \overset{*}{\longleftrightarrow} y, y \text{ is irreducible}\}$, the set of normal forms of x modulo R.*
5. *If $x, y \in S$ and $x \overset{*}{\rightarrow} y$, then x is an **ancestor** of y and y is a **descendant** of x. If $x \rightarrow y$ then x is a **direct ancestor** of y and y is a **direct descendant** of x.*
6. *If $x, y \in S$ and $x \overset{*}{\leftrightarrow} y$ then x and y are said to be **equivalent**.* □

Notation 22 (Reduction System) [3, page 10]
Let $R = (S, \rightarrow)$ be a reduction system. Then:

1. *For each $x \in S$:*
   *Let $\boldsymbol{\Delta}(\mathbf{x})$ denote the set of **direct descendants** of x with respect to $\rightarrow$. Thus, $\Delta(x) \overset{\text{def}}{=} \{y \,|\, x \rightarrow y\}$. Also, let $\Delta^+(x) \overset{\text{def}}{=} \{y \,|\, x \overset{+}{\rightarrow} y\}$ and $\Delta^*(x) \overset{\text{def}}{=} \{y \,|\, x \overset{*}{\rightarrow} y\}$. Thus, $\Delta^*(x)$ is the set of descendants of x modulo $\rightarrow$.*
2. *For each $A \subseteq S$:*
   *Let $\boldsymbol{\Delta}(\mathbf{A})$ denote the set of **direct descendants** of A with respect to $\rightarrow$. Thus, $\Delta(A) = \cup_{x \in A} \Delta(x)$. Also, let $\Delta^+(A) \overset{\text{def}}{=} \cup_{x \in A} \Delta^+(x)$ and $\Delta^*(A) \overset{\text{def}}{=} \cup_{x \in A} \Delta^*(x)$ Thus, $\Delta^*(A)$ is the set of descendants of the subset A modulo $\rightarrow$.*
3. *For each $x \in S$:*
   *Let $\nabla(\mathbf{x})$ denote the set of **direct ancestors** of x with respect to $\rightarrow$. Thus, $\nabla(x) \overset{\text{def}}{=} \{y \,|\, x \rightarrow y\}$. Also, let $\nabla^+(x) \overset{\text{def}}{=} \{y \,|\, x \overset{+}{\rightarrow} y\}$ and $\nabla^*(x) \overset{\text{def}}{=} \{y \,|\, x \overset{*}{\rightarrow} y\}$. Thus, $\nabla^*(x)$ is the set of ancestors of x modulo $\rightarrow$.*
4. *For each $A \subseteq S$:*
   *Let $\nabla(\mathbf{A})$ denote the set of **direct ancestors** of A with respect to $\rightarrow$. Thus, $\nabla(A) \overset{\text{def}}{=} \cup_{x \in A} \nabla(x)$. Also, let $\nabla^+(A) \overset{\text{def}}{=} \cup_{x \in A} \nabla^+(x)$ and $\nabla^*(A) \overset{\text{def}}{=} \cup_{x \in A} \nabla^*(x)$ Thus, $\nabla^*(A)$ is the set of ancestors of the subset A modulo $\rightarrow$.*
5. *Note that $\overset{*}{\longleftrightarrow}$ is an equivalence relation on S. For each $s \in S$ we denote by $[s]_R$ the equivalence class of s mod(R). Formally, $[s]_R \overset{\text{def}}{=} \{\, y \,|\, y \overset{*}{\longleftrightarrow} x \,\}$. Also, for any $A \subseteq S$, $[A] \overset{\text{def}}{=} \cup_{x \in A} [x]_R$.* □

Definition 23 [3, page 10]
Let R be a reduction system.

1. *The **common ancestor problem** is defined as follows:*
   ***Instance:** $x, y \in S$.*
   ***Problem:** Is there a $w \in S$ such that $w \overset{*}{\rightarrow} x$ and $w \overset{*}{\rightarrow} y$? In other words, do x and y have a common ancestor?*
2. *The **common descendant problem** is defined as follows:*
   ***Instance:** $x, y \in S$.*
   ***Problem:** Is there a $w \in S$ such that $x \overset{*}{\rightarrow} w$ and $y \overset{*}{\rightarrow} w$? In other words, do x and y have a common descendant?*

3. *The* ***word problem*** *is defined as follows:*
   Instance: $x, y \in S$.
   Problem: *Are* x *and* y *equivalent under* $\overset{*}{\longleftrightarrow}$*?* □

In general these problems are undecidable [3]. However, there are certain conditions that can be imposed on reduction systems in order for these questions to become decidable.

Lemma 21 *Let* $(S, \rightarrow)$ *be a reduction system such that for every* $x \in S$*,* x *has a unique normal form. Then* $\forall x, y \in S$*,* $x \overset{*}{\longleftrightarrow} y$ *if and only if the normal form of* x *is identical to the normal form of* y*.*

Proof of Lemma Let $x, y \in S$ and let x' and y' denote the normal forms of x and y respectively.
$\Rightarrow$ Suppose that $x \overset{*}{\longleftrightarrow} y$ and $x' \neq y'$. Then $x \overset{*}{\longleftrightarrow} y'$ since $x \overset{*}{\longleftrightarrow} y$ (by assumption) and $y \overset{*}{\longleftrightarrow} y'$ (by definition). Now y' is irreducible (by definition) and therefore x has two distinct normal forms: x' and y'. This is a contradiction.
$\Leftarrow$ Suppose that x and y have a common normal form z. Then, by definition, $x \overset{*}{\longleftrightarrow} z$ and $y \overset{*}{\longleftrightarrow} z$. The results follows from the symmetry and transitivity of $\overset{*}{\longleftrightarrow}$ ■

The proof of this lemma was omitted in [3]. The above result means that if for all $x, y \in S$ we have an algorithm to check if $x = y$ (very easy for strings), and also an algorithm to compute the unique normal forms of x and y, then the word problem becomes *always* decidable.

Definition 24 [3, page 11]
Let R *be a reduction system.*

1. R *is* ***confluent*** *if* $\forall w, x, y \in S$*,* $w \overset{*}{\rightarrow} x$ *and* $w \overset{*}{\rightarrow} y$ *implies that* $\exists z \in S$ *such that* $x \overset{*}{\rightarrow} z$ *and* $y \overset{*}{\rightarrow} z$*.*
2. R *is* ***locally confluent*** *if* $\forall w, x, y \in S$*,* $w \rightarrow x$ *and* $w \rightarrow y$ *implies that* $\exists z \in S$ *such that* $x \overset{*}{\rightarrow} z$ *and* $y \overset{*}{\rightarrow} z$*.*
3. R *is* ***Church-Rosser*** *if* $\forall x, y \in S$*,* $x \overset{*}{\longleftrightarrow} y$ *implies that* $\exists z \in S$ *such that* $x \overset{*}{\rightarrow} z$ *and* $y \overset{*}{\rightarrow} z$*.* □

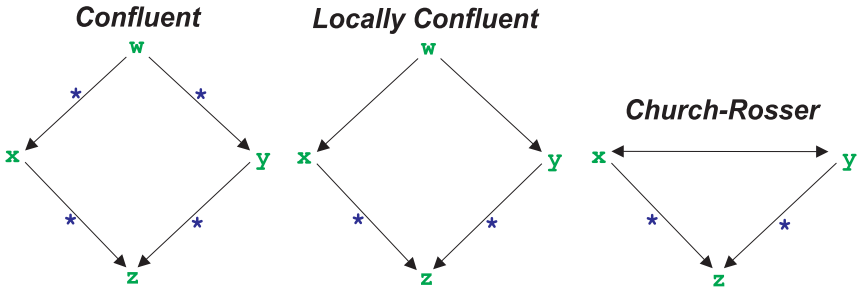

Fig. 1. Properties of Reduction Systems.

Definition 25 [3, page 12]
Let R *be a reduction system. The relation* $\rightarrow$ *is* ***noetherian*** *if there is no infinite sequence* $x_0, x_1, x_2, \cdots \in S$ *such that* $x_i \rightarrow x_{i+1}$ *for* $i \geq i$*. If* R *is confluent and* $\rightarrow$ *is noetherian then* R *is* ***convergent****.* □

If R is a reduction system and $\rightarrow$ is noetherian then we are assured that at least one normal form exists. This means that the word problem and the common descendant problem are decidable. Furthermore, if R is convergent then, for every $s \in S$, $[s]_R$ has one unique normal form. In addition, if R is convergent then R is confluent if and only if R is locally confluent (see proof of Theorem 1.1.13 in [3]).

3 String-Rewriting Systems

A string-rewriting system, T, is simply a set of *rewriting rules* of the form (l, r) where $l, r \in \Sigma^*$ for some finite alphabet Σ. The reduction system associated with T is $R = (\Sigma^*, \rightarrow_T)$ where $\rightarrow_T$ is the *reduction relation* induced by T. If $(l, r) \in T$ implies that $(r, l) \in T$ then T is called a *Thue System* otherwise it is called a *semi-Thue System*. In recent years there has been a resurgence of interest in Thue systems [3, 4]. This interest is perhaps due to the advances made in computer algebra, automated deduction and symbolic computation in general [4]. There have also been a number of new results in the theory of replacement systems and this has spurned on more research. In machine learning we are concerned primarily with string-rewriting systems that induce reduction relations that are noetherian and, in particular, those that have only length-reducing rules, i.e where $|l| > |r| \, \forall (l, r) \in T$. This property is desirable since it ensures that for any string $x \in \Sigma^*$, the normal forms of x exist and are computable. One of the objectives of my research is to show how such string-rewriting systems can be used to (partially) specify certain interesting subclasses of formal languages.

3.1 Definitions and Notation

Definition 31 (String-Rewriting Systems) *Let Σ be a finite alphabet.*

1. *A **string-rewriting system** T on Σ is a subset of $\Sigma^* \times \Sigma^*$ where every pair $(l, r) \in T$ is called a **rewrite rule**.*
2. *The **domain** of T is the set $\{l \in \Sigma^* \mid \exists r \in \Sigma^* \, and \, (l, r) \in T\}$ and denoted by **dom(T)**. The **range** of T is the set $\{r \in \Sigma^* \mid \exists l \in \Sigma^* \, and \, (l, r) \in T\}$ and denoted by **range(T)**.*
3. *When T is finite the **size** of T, which we denote by $\|T\|$, is defined to be the sum of the lengths of the strings in each pair in T. Formally, $\|T\| \stackrel{\text{def}}{=} \sum_{(l,r) \in T} (|l| + |r|)$.*
4. *The **single-step reduction relation** on Σ^*, $\rightarrow_T$, that is induced by T is defined as follows: for any $x, y \in \Sigma^*$, $x \rightarrow_T y$ if and only if $\exists u, v \in \Sigma^*$ such that $x = ulv$ and $y = urv$. In other words, $x \rightarrow_T y$ if and only if the string y can be obtained from the string x by replacing the factor l in x by r to obtain y.*
   *The **reduction relation** on Σ^* induced by T, which we denote by $\stackrel{*}{\rightarrow}_T$, is the reflexive, transitive closure of $\rightarrow_T$.*
5. *$R_T = \{\Sigma^*, \rightarrow_T\}$ is the reduction system induced by T.*
6. *The **Thue Congruence** generated by T is the relation $\stackrel{*}{\longleftrightarrow}_T$ - i.e. the symmetric, reflexive, and transitive closure of $\rightarrow_T$. Any two strings $x, y \in \Sigma^*$ are **congruent mod(T)** if $x \stackrel{*}{\longleftrightarrow}_T y$. For any string $w \in \Sigma^*$, the (possibly infinite) set $[w]_T$, i.e. the equivalence class of the string w mod(T), is called the **congruence class** of w (mod(T)).*
7. *Let S and T be two string-rewriting systems. S and T are called **equivalent** if they generate the same Thue congruence, i.e. if $\stackrel{*}{\longleftrightarrow}_S = \stackrel{*}{\longleftrightarrow}_T$.* □

Notes to Definition 31 For any string-rewriting system T on Σ, the pair $(\Sigma, \rightarrow_T)$ is a reduction system. T is a finite set of string pairs (rules) of the form (l, r). Each rule can be interpreted to

mean 'replace l by r'. The reduction relation induced by T, $\rightarrow_T$, is usually much larger than T itself since it contains not just the rules of T but also all those strings pair (x, y) such that, for some $a, b \in \Sigma^*$, $y = arb$ is obtained from $x = alb$ by a single application of the rule (l, r). In practice, for obvious reasons, $\rightarrow_T$ is infinite.

Many of the properties of reduction systems we discussed in Section 2 apply also to R_T. In particular, if T is a string-rewriting system on Σ and $R_T = \{\Sigma^*, \rightarrow_T\}$ is the reduction system induced by T, then, for any two strings $x, y \in \Sigma^*$;

- $\rightarrow_T$ is **confluent** if $w \overset{*}{\rightarrow}_T x$ and $w \overset{*}{\rightarrow}_T y$ for some $w \in \Sigma^*$, then $\exists z \in \Sigma^*$ such that $z \overset{*}{\rightarrow}_T z$ and $y \overset{*}{\rightarrow}_T z$. T is therefore confluent if whenever any 2 strings have a common ancestor they also have a common descendant.
- $\rightarrow_T$ is **Church-Rosser** if $x \overset{*}{\longleftrightarrow}_T x$ then $\exists z \in \Sigma^*$ such that $z \overset{*}{\rightarrow}_T z$ and $y \overset{*}{\rightarrow}_T z$. Informally, $\rightarrow_T$ is Church-Rosser if any pair of equivalent strings has a common descendant.
- $\rightarrow_T$ is **locally confluent** if $w \rightarrow_T x$ and $w \rightarrow_T y$ for some $w \in \Sigma^*$, then $\exists z \in \Sigma^*$ such that $z \overset{*}{\rightarrow}_T z$ and $y \overset{*}{\rightarrow}_T z$. In other words, $\rightarrow_T$ is locally confluent whenever any two strings have a common direct ancestor they also have a common descendant.

It is important to note that the above are not if-and-only-if conditions. For any two strings x and y, x and y having a common descendant does not necessarily imply that x is equivalent to y or that they have a common ancestor. Consider, as an example, the string-rewriting system $T = \{(ax, z), (ay, z)\}$ where $\Sigma = \{a, b, x, y, z\}$. The strings axb and ayb have a common descendant since $axb \rightarrow_T zb$ and $ayb \rightarrow_T zb$ but clearly cannot have a common ancestor.

As from this point onwards, purely in the interests of brevity and clarity, we shall omit the subscript T and simply use $\rightarrow$, $\overset{*}{\rightarrow}$, and $\overset{*}{\longleftrightarrow}$ instead of $\rightarrow_T$, $\overset{*}{\rightarrow}_T$, and $\overset{*}{\longleftrightarrow}_T$.

Definition 32 (Orderings on Σ^*) *Let $\rhd$ be a binary relation on Σ.*

1. *If T is a string-rewriting system on Σ, $\rhd$ is said to be **compatible with T** if $l \rhd r$ for each rule $(l, r) \in T$.*
2. *$\rhd$ is a **strict partial ordering** if it is irreflexive, anti-symmetric, and transitive.*
3. *If $\rhd$ is a strict partial ordering and if, $\forall x, y \in \Sigma^*$, either $x \rhd y$, or $y \rhd x$, or $x = y$, then $\rhd$ is a **linear ordering**.*
4. *$\rhd$ is **admissible** if, $\forall x, y, a, b \in \Sigma^*$, $x \rhd y$ implies that $axb \rhd ayb$. In other words, left and right concatenation preserves the ordering.*
5. *$\rhd$ is called **well-founded** if it is a strict partial ordering and if there is no infinite chain $x_0 \rhd x_1 \rhd x_2 \cdots$. If $\rhd$ is well-founded but also linear then it is a **well-ordering**.* □

Notes to Definition 32 It turns out that if T is a string-rewriting system on Σ then $\rightarrow_T$ is noetherian if and only if there exists an admissible well-founded partial ordering $\rhd$ on Σ^* that is compatible with T. (Lemma 2.2.4 in [3]). This is useful because, for reasons we outlined previously, we want to consider only string-rewriting systems that are noetherian. For any string-rewriting system T, in order to establish whether $\rightarrow_T$ is noetherian we need only find (or construct) an admissible well-founded partial ordering that is compatible with T. In our case we usually opt for the **length-lexicographical ordering**, i.e. where strings are ordered according to length first and then lexicographically.

Notice also that for any string-rewriting system T, the set of direct descendants of a string $x \in \Sigma^*$ modulo T, $\Delta(x)$, is finite. This is true even if $\rightarrow_T$ is not noetherian and follows from the fact that any string $x \in \Sigma^*$ has a finite number of substrings and therefore the rules in T can only be applied in a finite number of ways. On the other hand, if $\rightarrow_T$ is noetherian, then $\forall x \in \Sigma^*$, the set of all descendants of x, $\Delta^*(x)$, is finite. This follows by König's Infinity Lemma.

Definition 33 (Normalized String-Rewriting Systems) *Let T be a string-rewriting system on Σ. T is **normalized** if, for every rule $(l, r) \in T$,*

1. *$l \in IRR(T - \{(l, r)\})$, and*
2. *$r \in IRR(T)$.* □

Notes to Definition 33 Informally, T is normalized if and only if, for each rule (l, r) in T, the left-hand side l can only be reduced by the rule (l, r) itself and the right-hand side r is irreducible. If T is a string-rewriting system that is not normalized, i.e. it contains rules whose right-hand side that is reducible, there is a polynomial time algorithm that on input T will output a string-rewriting system T' such that T' is normalized and equivalent to T [3, Page 47]. Unless otherwise stated, all string-rewriting systems we will consider from now on are normalized.

3.2 Length-Reducing String-Rewriting Systems

A string-rewriting system T is called *length-reducing* if $(l, r) \in T$ implies that $|l| > |r|$. In other words the left hand side of a rule is always longer than the right hand side. The obvious implication of this property is that when a rule is applied to a string x the resulting string x' is always strictly shorter than x. Recall that if S is any string-rewriting system on Σ then $\rightarrow_S$ is noetherian if and only if there exists an admissible well-founded partial ordering $\rhd$ on Σ^* that is compatible with S. Therefore, let $\rhd$ be the length-lexicographical partial order on Σ^*. Since $l \rhd r$ clearly holds $\forall (l, r) \in T$ and also since $\rhd$ is admissible and well-founded then we can conclude $\rightarrow_T$ is noetherian. There are three particular types of length-reducing we shall use in this report. We now give the definitions.

Definition 34 (Monadic String-rewriting Systems)
*Let T be a string-rewriting system on Σ. T is called **monadic** if T is length-reducing and $|r| = 1$ or $r = \varepsilon$, $\forall (l, r) \in T$.*

Definition 35 (Special String-rewriting Systems)
*Let T be a string-rewriting system on Σ. T is called **special** if T is length-reducing and $r = \varepsilon$, $\forall (l, r) \in T$.*

Definition 36 (Trivial String-rewriting Systems)
*Let T be a string-rewriting system on Σ. T is called **trivial** if $r = \varepsilon$, and $l = a, a \in \Sigma$, $\forall (l, r) \in T$.*
□

3.3 Congruential Languages

We now informally investigate the possibility (and, in my talk, the feasibility) of using string-rewriting systems to define, and also test for membership of, formal languages.

Let T be a string-rewriting system over an alphabet Σ. How can we use T to define a proper language $L \subset \Sigma^*$? In other words, are there any interesting, non-trivial languages *induced by* T? Of course, we must define exactly what it means for a language L to be induced by a string-rewriting system T. We examine some possibilities.

- Let L_1 be the set of all irreducible strings modulo T, i.e. $L_1 = \text{IRR(T)}$.
- Let L_2 be the union of all the equivalence classes with respect to T.

We observe that L_1 can be finite or infinite depending on T. It turns out that the set $IRR(T)$, i.e. the set of all irreducible strings modulo some string-rewriting system T, is a regular language and a finite state automaton that recognizes L_1 can be constructed in polynomial time from T [3, Lemma 2.1.3, Page 37]. Whether such languages are useful or not is open to discussion but a review of the literature does not reveal any particular use. L_2, it turns out, is Σ^* itself. It appears, therefore, that T by itself is rather limited for the purpose of defining formal languages. Suppose, however, that we use T together with a finite number of equivalence (congruency) classes modulo T.

Definition 37 (Congruential Languages) [9]
*Let T be a finite string-rewriting system on Σ. A **congruential language** is any finite union, C, of congruency classes of T.* □

We specify a congruential language by the pair (T, C). Since both T and C are finite sets, we have a finite description of the language. Congruential languages have been subject to study by various researchers [4, 9]. One interesting result is that all NTS languages are congruential [**?**]. A context-free grammar is said to be NTS if the set of sentential forms it generates is unchanged when the rules are used both ways. It is quite common, and also sensible, to restrict attention to congruential languages where T has only length-reducing rules. This will ensure that $\rightarrow_T$ is noetherian and this guarantees that the normal forms for each string exist and are computable. We give below an example of congruential languages where T has only length reducing rules.

Example 31 *An Example of a Congruential Language*

- *Let $\Sigma = \{a, b\}$ and let $T = \{(ab, \varepsilon), (\varepsilon, ab)\}$. This is the Dyck Language of matching parenthesis.*

4 Thue Systems in Pattern Recognition

In my talk I will give examples of how Thue string rewriting system can be used in pattern recognition and also how learning algorithms can be developed to learn classes where the instances of the class are encoded as strings in some formal language. One of my objectives is to introduce Thue Systems to my colleagues especially those interested in formal language theory, compiling techniques, theoretical CS, and machine learning. The seminal work is this area is Ronald Book's excellent exposition [3].

References

1. Abela, John, *Topics in Evolving Transformation Systems*, Masters Thesis, Faculty of Computer Science, University of New Brunswick, Canada, **1994**.
2. Abela, John, *Learning Picture Languages*, Technical Report TR-CS-9605, Department of Computer Science and Artificial Intelligence, University of Malta, **1996**.
3. Book, Ronald V. and Otto, Friedrich, *String-Rewriting Systems*, Springer-Verlag, **1993**.
4. Book, Ronald V., *Thue Systems as Rewriting Systems*, J. Symbolic Computation, Vol. 3, pp. 39-68, **1987**.

5. Bunke, H. and Sanfeliu, A., (eds) *Syntactic and Structural Pattern Recognition - Theory and Applications*, World Scientific series in Computer Science, Vol. 7, **1990**.
6. Duda, Richard and Hart, Peter, *Pattern Classification and Scene Analysis*, Wiley Interscience, **1973**.
7. Fu, King Sun, *Syntactic Pattern Recognition and Applications*, Prentice-Hall, **1982**.
8. Jantzen, Matthias, *Confluent String Rewriting*, EATCS monographs on theoretical computer science, Springer-Verlag, **1988**.
9. MacNaughton, R., et al, *Church-Rosser Thue Systems and Formal Languages*, Journal of the ACM, vol. 35, pp. 324-344, **1998**.
10. McNaughton, R., Narendran, P., and Otto, F., *Church-Rosser Thue Systems and Formal Languages*, Journal of the ACM, Vol. 35, No. 2, pp. 324-344, April **1988**.
11. Oommen, B.J., and loke, R.K.S., *Pattern Recognition of Strings with Substitutions, Insertions, Deletions, and Generalized Transpostions*, Pettern Recognition, vol. 30, No. 5, pp. 789-800, **1997**.
12. Thue, A., *Probleme über Veränderungen von Zeichenreihen nach gegbenen Regeln*, Skr. Vid. Kristania, I Mat. Natuv. Klasse, No. 10, **1914**.
13. Tou, J.T. and Gonzalez, R.C., *Pattern Recognition Principles*, Addison-Wesley, **1974**.
14. Vidal, E., *Grammatical Inference: An Introductory Survey*, in Carrasco, R., and Oncina, J., (eds), *Grammatical Inference and Applications*, Second International Colloquium, ICGI-94, Alicante, Spain, September **1994**, published by Springer-Verlag.