Journal of mathematics and computer Science 15 (2015) 88 - 96

Contents list available at JMCS

Journal of Mathematics and Computer Science

Journal Homepage: www.tjmcs.com

A Note on Automata

Dasharath Singh[1], **Ahmed Ibrahim Isah**[2]

[1] (Former Professor, Indian Institute of Technology, Bombay) Professor, Mathematics Department, Ahmadu Bello University, Zaria, Nigeria
mathdss@yahoo.com
[2] Mathematics Department, Ahmadu Bello University, Zaria, Nigeria
aisah204@gmail.com

Article history:
Received November 2014
Accepted February 2015
Available online February 2015

Abstract

In this paper, a brief description of finite automata and the class of problems that can be solved by such devices is presented. The main objective is to introduce the concept of *length product*, illustrate its application to finite automata, and prove some related results.

Keywords: Formal language, Automata, Length product

1. Introduction

Formal languages (or simply languages) were developed in the 1950s by Noam Chomsky in an endeavor to systematically describe natural languages with an objective to make passages easier from one language to another using computers. Although, endeavors in this direction have only been partially successful, it has found a number of applications in computer science such as compiler construction for high-level programming languages, etc.

A language is a set of strings over an alphabet. A language can be finite or infinite. A finite language can be described by listing its strings which is not the case for infinite languages; even for some finite languages the listing could be quite cumbersome. Moreover, in general, for a language to be interesting mathematically, it needs to have an infinite set of strings.

Formal language theory develops mathematical and computational techniques for specifying languages. How simple or complex a language could be depends on the level of its description. There are two fundamental machineries for defining or specifying languages: *grammars* (which generate the strings) and *automata* (which recognize the strings) of a language. Regular expressions could also be used to define regular languages.

A grammar is a mathematical system that provides structures on the strings or sentences of a language. Essentially, a grammar of a language L is a finite set of *production* or *grammatical rules* that describe how to form valid strings from the alphabet of L. In other words, a grammatical structure of a language provides means to determine whether or not a string belongs to the set of valid strings. A computer programmer is mainly concerned with applying the production rules (or rules of syntax) to produce syntactically correct programs.

Chomsky [2, 4] classified grammars into the following four classes by imposing restrictions on the production rules, known as the *Chomsky hierarchy*: unrestricted grammars (or Type 0), context-sensitive grammars (or Type 1), context-free grammars (or Type 2), and regular grammars (or Type 3). The languages generated by Type 0, Type 1, Type 2 and Type 3 grammars are, respectively, called *type 0* (or *computably enumerable*), *Type 1* (or *context-sensitive*), *Type 2* (or *context-free*) and *Type 3* (or *regular*) languages (see [9] for details).

For simplicity, let the class symbols T_0, T_1, T_2 and T_3 denote, respectively, the Type 0, Type 1, Type 2, and Type 3 languages. Chomsky also showed that $T_3 \subset T_2 \subset T_1 \subset T_0$ and hence the *Chomsky hierarchy*. Nevertheless, it is known [5] that both T_2 and T_3 perform equally as far as only the issues related to *expressibility* and *ease of parsing* are taken into consideration, however, they are designed to deal with problems of different nature. In course of time, both extensions and modifications on Chomsky's original formulation of hierarchy of formal grammars, such as the notions of *mildly context-sensitive languages* and *sub-regular hierarchies*, etc., (see [8, 11, 14] for details) have appeared.

Chomsky [3] pointed out that as English has *centre embedding constructions*: involving two dependent elements a and b that are not adjacent, and that may contain another instance of the same construction between the two parts, such as *neither-nor* constructions, it is not regular.

Huybregts [7] and Shieber [16] argued that Swiss-German is not context-free as the dependencies between *verbs* and their *objects* were unbounded in length. In fact, it has crossing dependencies between objects and verbs, and the number of these interlocked dependencies is potentially unbounded. Since context-free grammars can handle an unbounded number of interlocked dependencies only if they are nested, Swiss-German cannot be context-free. On the same lines, Postal and Langendoen [13] observed that English was not context-free. In view of the fact that the aforesaid constructions were seen to extend to almost all other natural languages, it became imperative to design grammars formalisms that could be suitable for doing linguistics and also be in the vicinity of context-free grammars along with preserving its computational tractability.

Joshi [10] described a list of properties that an extension of the context-free languages should have if it is to be of practical use for linguistics: It should contain all context-free languages, it can describe a limited number of types of cross-serial dependencies, its membership problem has polynomial complexity, and all languages in it have constant growth property. He called classes of languages having these properties *mildly context-sensitive* as they extend the context-free languages and slightly absorbing some context-sensitive languages. In the literature, the classes of mutually equivalent formalisms satisfying the said properties are the TAG-languages (TAG abbreviating Tree Adjoining Grammar) and the MG-languages (MG abbreviating Minimalist Grammar).

TAG-languages consist of Tree Adjoining Grammars (TAGs), Combinatory Categorical Grammars (CCGs), Linear Indexed Grammars (LIGs), and Head Grammars (HGs). Joshi et al. [11] observed that they were equivalent as they described the same class of languages. MG-languages include Linear Context-free Rewrite Systems and Set-Local Multi-Component TAGs, and the formalisation of Noam Chomsky's Minimalist Grammars. The membership problem for TAG-languages is $O(n^6)$. Non-context free languages that belong to the TAG-languages are $a^n b^m c^n d^m$, the copy language, $a^n b^n c^n$, $a^n b^n c^n d^n$, etc. Examples of MG-languages include $a_1^n \dots a_k^n$ for arbitrary k, the k-copy language for any k (w^k for arbitrary k).

It may be observed from the examples above that TAG-languages may contain only up to four different types of interlocked unlimited (crossing or nesting) dependencies, but there is no such upper bound for MG-languages. This leads to a higher computational complexity of the membership problem for MG-languages. Becker et al. [1] argued that this added complexity was actually needed to capture all aspects of string order variation in German which could be applicable to other natural languages. Currently, though investigations are still on, it is assumed that all natural languages were MG-languages.

2. Finite automata

Automata theory studies mathematical models of computing devices (or machines) and the class of problems that can be solved by such devices. An automaton is an abstract computing device that recognizes strings of a language. It has a mechanism to read input, which is a string over a given alphabet, *process* it and *decide* whether the string is in the language or not (see [6, 12, 15] for details). An automaton is finite if it has a finite memory.

The languages defined by finite automata are exactly regular (Type 3) languages. Thus, in order to prove a language L regular, one needs to construct Type 3 grammar, regular expression or a finite automation that specifies it. Dually, in order to show that a language is not regular, it is to show that there does not exists a Type 3 grammar, regular expression or a finite automation that specifies it.

In the following, besides a brief description of finite automata, the notion of *Length product* is introduced, its application to finite automata is illustrated, and some related results are proved.

A finite automation is like a finite-state machine except that it has a set of acceptable states whereas the latter has an output alphabet. The class of finite automata is broadly divided into deterministic and non-deterministic types. In a deterministic finite automaton, if the *internal state*, *input* and *contents* of the storage are known, it is possible to predict the future behavior of the automaton, otherwise it is non-deterministic.

Deterministic finite automaton (DFA)

A DFA is a quintuple (or $5-$ tuple) $M = (Q, \Sigma, q_0, \delta, A)$ where

$*Q$ is a finite set of states (at all times, the internal memory is in some state $q \in Q$);

$*\Sigma$ is the set of input symbols (the machine only operates on strings over the alphabet Σ);

$*q_0$ is the initial state ($q_0 \in Q$);

$*\delta$ is the (state) transition function which maps each pair (q_i, a), where q_i is a state and a is an input symbol, to a unique next state q_j: $\delta(q_i, a) = q_j$; and

$*A$ is a *set of terminal (final or accepting) states* $(A \subseteq Q)$.

The string is accepted if the internal state of the machine, after reading the whole input, is some state of A, rejected otherwise.

As its input, the automaton M receives a string $u = a_1 \cdots a_n$ which it starts to read from the left. In the beginning, M is in its initial state q_0 reading the first symbol a_1 of u. The next state q_k is determined by the transition function: $\delta(q_0, a_1) = q_k$. Note that if M is in state q_k reading the symbol a_m, its next state is $\delta(q_k, a_m)$. If the final state of M, after the last input symbol a_n is read, is a state of A, then M accepts u, otherwise u is rejected.

M accepts the empty input ε if the initial state q_0 is also a terminal state.

The language recognized by an automaton M, denoted $L(M)$, is the set of all strings accepted by M.

Non-deterministic finite automata (NFA)

Non-determinism refers to availability of choices in state transitions. An NFA, unlike a DFA, allows several outgoing transitions at the same time, however, some of them may lead to a non-final state. Nevertheless, a string is accepted if there is at least one choice of transitions that takes the machine to a final state.

A Nondeterministic Finite Automaton (NFA) is a quintuple (or $5-$ tuple) $M = (Q, \Sigma, q_0, \delta, A)$ where

Q, Σ, q_0 and A are as for DFA; and the transition function δ is defined as

$\delta: Q \times \Sigma \longrightarrow 2^Q$.

2^Q, as usual, is the power set of Q. That is, for each state q_i and an input letter a, the transition function δ maps each pair (q_i, a) to exactly one subset (possible next states) of Q.

NFA with ε – moves

ε – *moves* are spontaneous transitions that result in a change of states without reaching any input letter and there is no restriction on its number i.e., any number of ε – moves are allowed.

An NFA M with $\varepsilon -$ moves ($\varepsilon -$ NFA, for short) is defined as $M = (Q, \Sigma, \delta, q_0, A)$ where Q, Σ, q_0, and A are the same as for an arbitrary NFA, but the function δ is defined as

$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \longrightarrow 2^Q$,

satisfying for each q, the transitions with all input letters a and ε.

i. $\delta(q, a)$ is the set of states p such that there is a move from q to p with input a, and
ii. $\delta(q, \varepsilon)$ is the set of states p such that there is a *spontaneous move* from q to p.

3. Operations on regular languages

It is known [12, 15] that if L_1, L_2 are regular languages, then $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$, $\overline{L_1}$, $L_1 L_2$, etc., are regular languages. In other words, if there exist finite automata M_1, M_2 for L_1, L_2 respectively, then there exist finite automata for set-theoretic operations over them. In the case of context-free languages, intersection and complement do not hold.

We introduce below a new operation termed *length product*, illustrate its application to finite automata and prove some related results.

3.1 Definition   Length Product

Let $L_1 = \{u_1, u_2, \dots\}$ and $L_2 = \{v_1, v_2, \dots\}$ be languages. Then, the length product of L_1 and L_2, denoted $L_1 \otimes L_2$, is defined as $L_1 \otimes L_2 = \{(u_i, v_j) \,|\, |u_i| = |v_j|,\ u_i \in L_1,\ v_j \in L_2\}$ where $|u_i|$ denotes the length of the string u_i and (u_i, v_j) represents an ordered pair. It may be noted that unlike Cartesian product, in order that the operation $\otimes$ to be defined, the strings need to be of equal length.

Theorem 3.1

If the languages L_1 and L_2 are recognized by finite automata M_1 and M_2 respectively, then their length product $L_1 \otimes L_2$ is recognized by a finite automaton. In other words, if L_1 and L_2 are regular languages, then their length product is also a regular language.

Proof

Let $M_1 = (Q, \Sigma_1, q_0, \delta, F)$ and $M_2 = (P, \Sigma_2, p_0, \beta, G)$ be two automata such that M_1 recognizes L_1 and M_2 recognizes L_2. Then an automaton M, recognizing $L_1 \otimes L_2$, is constructed as follows:

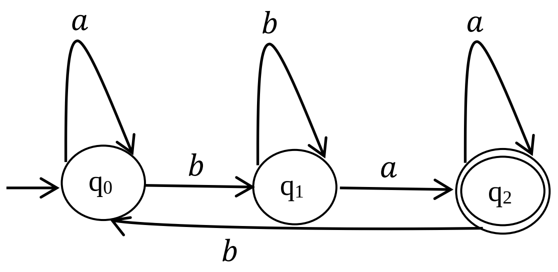
$M = (R, \Sigma, r_0, \mu, D)$ where $R = Q \times P$, $\Sigma = \Sigma_1 \times \Sigma_2$, $r_0 = (q_0, p_0)$, $D = F \times G$, and if $\delta(q_k, a) = q_r$ and $\beta(p_m, b) = p_s$, then μ is defined as $\mu((q_k, p_m), (a, b)) = (q_r, p_s)$ where $a \in \Sigma_1, b \in \Sigma_2, q_k\text{'s} \in Q, p_m\text{'s} \in P$.

Let us consider a miniature example to illustrate the aforesaid process.

Let $M_1 = (Q, \Sigma_1, q_0, \delta, F)$ with $Q = \{q_0, q_1, q_2\}$, $\Sigma_1 = \{a, b\}$, $q_0 = q_0$, $F = \{q_2\}$, and δ is given as below:

	a	b
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_0.

The transition diagram for M_1 is the following:



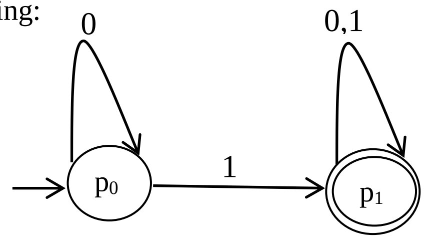
Transition diagram for M_1

Also, $L(M_1) = \{ba, aba, bba, \dots\}$.

Let $M_2 = (P, \Sigma_2, p_0, \beta, G)$ with $P = \{p_0, p_1\}$, $\Sigma_2 = \{0,1\}$, $p_0 = p_0$, $G = \{p_1\}$, and β is given as below:

	0	1
p_0	p_0	p_1
p_1	p_1	p_1.

The transition diagram for M_2 is the following:



Transition diagram for M_2

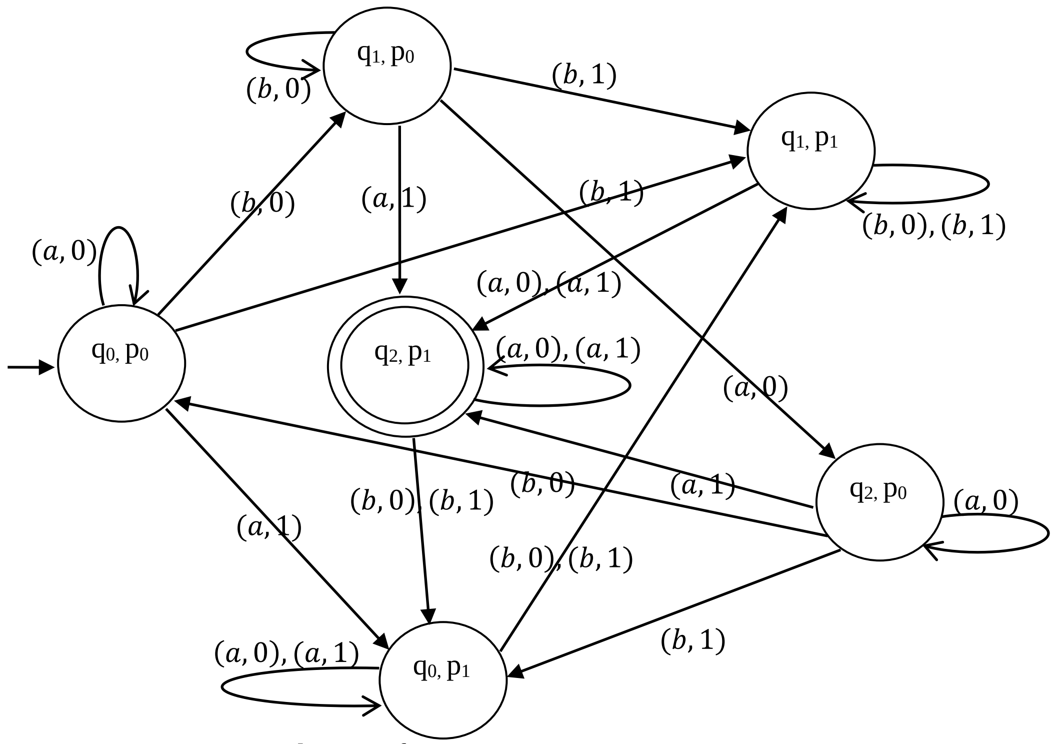
Also, $L(M_2) = \{01, 011, 001, \dots\}$.

Now the automaton M, recognizing $L_1 \otimes L_2$, is defined as follows:

$M = (R, \Sigma, r_0, \mu, D)$ where $R = \{(q_0, p_0), (q_0, p_1), (q_1, p_0), (q_1, p_1), (q_2, p_0), (q_2, p_1)\}$, $\Sigma = \{(a, 0), (a, 1), (b, 0), (b, 1)\}, r_0 = (q_0, p_0), D = \{(q_2, p_1)\}$, and μ as constructed below:

	$(a,\ 0)$	$(a,\ 1)$	$(b,\ 0)$	$(b, 1)$
(q_0, p_0)	(q_0, p_0)	(q_0, p_1)	(q_1, p_0)	(q_1, p_1)
(q_0, p_1)	(q_0, p_1)	(q_0, p_1)	(q_1, p_1)	(q_1, p_1)
(q_1, p_0)	(q_2, p_0)	(q_2, p_1)	(q_1, p_0)	(q_1, p_1)
(q_1, p_1)	(q_2, p_1)	(q_2, p_1)	(q_1, p_1)	(q_1, p_1)
(q_2, p_0)	(q_2, p_0)	(q_2, p_1)	(q_0, p_0)	(q_0, p_1)
(q_2, p_1)	(q_2, p_1)	(q_2, p_1)	(q_0, p_1)	(q_0, p_1).

The transition diagram for M is as follows:



Transition diagram for M

Also, $L(M) = \{(ba, 01), (aba, 011), (aba, 001), (bba, 011), (bba, 001), \ldots\}$.

Theorem 3.2

The length product is closed under union and intersection.

Proof

Let L_1, L_2, L_3 and L_4 be regular languages. Then, from above, $\overline{L_1}, \overline{L_2}, \overline{L_3}, \overline{L_4}$ are regular, and hence $\overline{L_1} \cup \overline{L_2}$ and $\overline{L_1} \cap \overline{L_2}$ are regular.

Moreover, applying De Morgan's laws, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, and thus $L_1 \cap L_2$ is regular, similarly for $L_3 \cap L_4$. Thus, $L_1 \cap L_2 \otimes L_3 \cap L_4$ is regular from theorem 3.1.

Analogously, $L_1 \cup L_2 \otimes L_3 \cup L_4$ is also regular.

3. Concluding Remarks

The *cardinality bounded language* X^{n*} was introduced in [17], which is a language consisting of the set of all strings of length $\leq n$ over an alphabet X, that is, $X^{n*} = X^n \cup X^{n-1} \cup \dots \cup X^0$. Clearly, $\{X^{n*}\}$ is a strictly monotonic increasing nested sequence and $X^* = X^{0*} \cup X^{1*} \cup \dots \cup X^{n*} \cup \dots$ gives an alternative form of representation of the usual one viz., $X^* = X^0 \cup X^1 \cup \dots \cup X^n \cup \dots$, where X^n is the set of all strings of length n over X. Moreover, $\bigcup_{n=0}^{\infty} X^n = \bigcup_{n=0}^{\infty} X^{n*}$, but $\bigcap_{n=0}^{\infty} X^n = \emptyset$, whereas $\bigcap_{n=0}^{\infty} X^{n*} = \{\varepsilon\}$. It is straightforward to see that the length product is also applicable to the language X^{n*}, and hence it seems to have a promise, specially from application points of view to problems representing physical situations. Nevertheless, the *complexity* aspects of the operation *length product* is yet to be investigated.

References

[1] T. Becker, A. Joshi, and O. Rambow, "*Long-distance scrambling and tree adjoining grammars*" In Proceedings of the fifth conference on European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, (1991) 21–26.

[2] N. Chomsky, "*Three Models for the Description of Languages*", IRE Transactions on Information Theory, 2 (3) (1956) 113–124.

[3] N. Chomsky, "*Syntactic Structures*", Mouton, The Hague, (1957).

[4] N. Chomsky, "*On Certain Formal Properties of Grammars*", Information and Control, 2 (2) (1959) 137–167.

[5] D. Grune and C. H. Jacobs, "*Parsing Techniques – A Practical Guide*", Ellis Horwood, England (1990).

[6] J. E. Hopcroft and J. D. Ullman, "*Formal Languages and Their Relations to Automata*", Addison-Wesley Publishing (1969).

[7] R. Huybregts, "*The weak inadequacy of context-free phrase structure grammars*", In Ger Jan de Haan, Mieke Trommelen, and Wim Zonneveld, editors, Van periferie naar kern, Foris, Dordrecht, (1984) 81–99.

[8] G. Jäger and J. Rogers, "*Formal Language Theory: Refining the Chomsky Hierarchy*", Philosophical Transactions of Royal Society of London, Series B, Biological Sciences, (2012) 1–29.

[9] T. Jiang, B. Ravikumar, M. Li and K. W. Regan, "*Formal Grammars and Languages*", Lecture Notes, (2010) 1 – 40.

[10] A. Joshi, "*How much context-sensitivity is necessary for characterizing structural descriptions — tree adjoining grammars*" In David Dowty, Lauri Karttunen, and Arnold Zwicky, editors, Natural Language Processing, Theoretical, Computational and Psychological Perspectives, Cambridge University Press, Cambridge, UK, (1985).

[11] A. K. Joshi, K. V. Shanker and D. Weir, "*The Convergence of Mildly Context-Sensitive Grammar Formalisms*", Technical Reports (CIS), Paper 539, (1990) 1–65.
[12] J. Kari, "*Automata and Formal Languages*", University of Turku, Finland, (2013) 1– 150.
[13] P. M. Postal and D. T. Langendoen, "*English and the Class of Context-Free Languages*", Computational Linguistics 10 (3 – 4) (1984) 177–181.
[14] J. Rogers and G. Pullum, "*Aural pattern recognition experiments and the subregular hierarchy*" In Marcus Kracht, editor, Proceedings of 10th Mathematics of Language Conference, University of California, Los Angeles (2007) 1–7.
[15] K. Ruohonen, "*Formal Languages*", Lecture Notes, (2009) 1–94.
[16] S. Shieber, "*Evidence against the context-freeness of natural language*", Linguistics and Philosophy, 8 (1985) 333–343.
[17] D. Singh and A. I. Isah, "*Some Algebraic Structures of Languages*", the Journal of Mathematics and Computer Science 14 (2015) 250–257.