Triangle: Language, Literature, Computation, n. 8, 2012
Publicacions Universitat Rovira i Virgili · ISSN: 2013-939X
https://revistes.urv.cat/index.php/triangle

A Class of 2-Head Finite Automata for Linear Languages

Benedek Nagy

Research Group on Mathematical Linguistics
Universitat Rovira i Virgili
Tarragona, Spain

Faculty of Informatics
University of Debrecen
Debrecen, Hungary
E-mail: nbenedek@inf.unideb.hu

Summary. Both deterministic and non-deterministic finite state machines (automata) recognize regular languages exactly. Now we extend these machines using two heads to characterize even-linear and linear languages. The heads move in opposite directions in these automata. For even-linear languages, deterministic automata have the same efficiency as non-deterministic ones, but for the general case (linear languages) only the non-deterministic version is sufficient. We compare our automata to other two-head automata as well.

1 Introduction

The theory of automata is well developed and applicable in many theoretical and practical fields. The class of finite automata (both deterministic and non-deterministic) characterizes regular languages. They have many interesting extensions, for instance, probabilistic, weighted automata etc.

Chomsky type grammars and generated language families are some of the most basic and important fields in theoretical computer science [1, 10].

In some senses, linear languages are more related to regular languages than context free ones. For instance, in [11] regular-like expressions are used to describe linear languages. There is a language class between linear and regular ones (namely, the even-linear languages) which play an important role in learning theory as well [12].

In this paper we present a class of 2-head finite automata which characterize linear context-free languages exactly. We analyse the deterministic versions of this class of automata as well. The normal form of these automata and a special class characterizing the even-linear languages are also presented. Some relations to other two-head automata, for instance to the Watson-Crick automata [9, 8], will also be discussed.

2 Preliminaries

In this section we recall some well-known concepts of formal language and automata theory.

Let V be a finite non-empty set of symbols (usually called letters). The strings built up by letters are called words. The sets of words are the languages over the alphabet V. In this paper the sign ε refers to the empty word.

First both the deterministic and non-deterministic finite state machines are recalled

Definition 1. *A 5-tuple $A = (Q, s, V, \delta, F)$ is a finite state machine or finite automata, with the finite (non-empty) set of states Q; $s \in Q$ is the initial state; V is the (input) alphabet and $F \subset Q$ is the set of final (or accepting) states. The function δ is the transition function. There are two extremal possibilities of this functions are used. If $\delta : Q \times (V \cup \{\varepsilon\}) \to 2^Q$, then the device is the non-deterministic finite automaton. If $\delta : Q \times V \to Q$ then the machine is called a deterministic finite automaton.*

A word w is accepted by a finite automaton if there is a run starting with s, ending in a state in F and the symbols of the transitions of the path results w.

Now we recall some language families related to the Chomsky hierarchy.

Definition 2. *A grammar is a construct $G = \langle N, V, S, H\rangle$, where N, V are the non-terminal and terminal alphabets, with $N \cap V = \emptyset$; they are finite sets. $S \in N$ is a special symbol, called initial letter. H is a finite set of pairs, where a pair uses to be written in the form $v \to w$ with $v \in (N \cup V)^* N (N \cup V)^*$ and $w \in (N \cup V)^*$. (We used the well-known Kleene-star notation.) H is the set of derivation rules.*

Let G be a grammar and $v, w \in (N \cup V)^*$. Then $v \Rightarrow w$ is a direct derivation if and only if there exist $v_1, v_2, v', w' \in (N \cup V)^*$ such that $v = v_1 v' v_2$, $w = v_1 w' v_2$ and $v' \to w' \in H$. A derivation $v \Rightarrow^* u$ holds if and only if either $v = u$ or there is a finite sequence of sequential forms connecting them as $v = v_0, v_1, ...v_m = u$ in which $v_i \Rightarrow v_{i+1}$ is a direct derivation for each $0 \leq i < m$.

A sequence of letters v for which $S \Rightarrow^* w$ and $v \in (N \cup V)^*$ holds, is called a sentential form. The language generated by a grammar G is the set of terminal words that can be derived from the initial letter: $L(G) = \{w | S \Rightarrow^* w \wedge w \in V^*\}$.

Two grammars are (weakly) equivalent if they generate the same language (modulo ε).

Depending on the possible structures of the derivation rules we are interested in the following classes.

• type 2, or context-free (CF) grammars: for every rule the next scheme holds: $A \to v$ with $A \in N$ and $v \in (N \cup V)^*$.

• linear (lin) grammars: each rule is one of the following forms: $A \to v$, $A \to vBw$; where $A, B \in N$ and $v, w \in V^*$.

• even-linear (elin) grammars: each rule is one of the following forms: $A \to v$, $A \to w_1 B w_2$; where $A, B \in N$ and $v, w_1, w_2 \in V^*$ and the length of w_1 equals the length of w_2 for each rule.

• type 3, or regular (reg) grammars: each derivation rule is one of the following forms: $A \to w$, $A \to wB$; where $A, B \in N$ and $w \in V^*$. (Note, that this form is the so-called right-linear form of these grammars; we will use later alternative forms as well.)

The generating powers of these grammars are in the following hierarchy: $L_{reg} \subsetneq L_{elin} \subsetneq L_{lin} \subsetneq L_{CF}$.

Now we present normal forms for the rules of linear / even-linear / regular grammars. (For instance this is well-known and widely used for regular grammars.)

Lemma 1. *Every linear grammar has an equivalent grammar in which all rules are in forms* $A \to aB, A \to Ba, A \to a$.
Every even-linear grammar has an equivalent grammar in which all rules are in forms $A \to aBb, A \to a, A \to \varepsilon$.
Every regular languages can be generated by grammar having only rules of types $A \to aB, A \to a$ $(A, B \in N, a, b \in V)$.

Proof. Introducing new non-terminals each rule can be replaced by a sequence of rules in the desired forms. □

For context-free languages the concept of push-down automata fits. In the literature, push-down automata with a restriction are used for linear languages (as they are special context-free languages). This restriction is the following: whenever the content of the stack is decreasing in a transition, it cannot push anything again into the stack. These special push-down automata are called 1-turn push-down automata.

From a derivation-tree point of view linear languages are more related to regular ones (Fig. 1, [6]). On the basis of this observation we modify the well-known concept of finite automata to get an accepting device for linear languages.

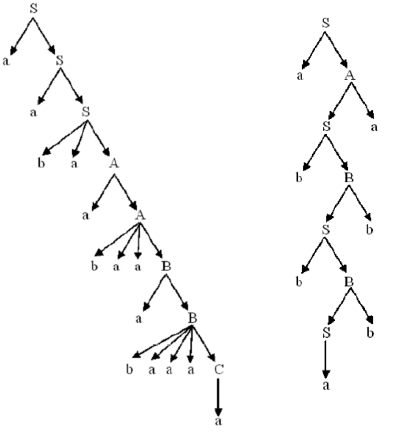

Fig. 1. Derivations in regular grammar and in linear grammar (in normal form)

There is at most 1 non-terminal in each sentential form, soit can be modelled by a finite-state machine as we will show in the next section.

3 Definition of 2-Head Finite Automata Accepting Linear Languages

We construct finite automata with two heads. They read the word from the beginning and the end, in parallel.

Definition 3. *The 5-tuple* $\langle Q, s, V, d, F\rangle$ *with the transition function* $d : Q \times (V \cup \{\varepsilon\}) \times (V \cup \{\varepsilon\}) \to 2^Q$*, where* Q *is the finite set of states,* $s \in Q$ *is the starting state,* $F \subset Q$ *contains the final states, and* V *is (as usual the set of terminals:) the alphabet.*

This automaton finishes reading the input word when the heads meet, so the whole word is processed: every letter is read by one of the heads.

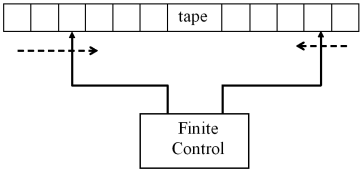

Fig. 2. A draw of a 2-head finite automaton

In transitions we assign a pair of symbols (a, b) to the arrows meaning that the first head reads symbol a, the second reads b and both step. We allow both a and b to be the sign ε.

In Figure 2 the sketch of this type of automata can be seen with the 2 heads and the directions of their motions.

One of our aims is to develop and analyse this automaton.

4 Properties of the 2-Head Finite Automata

First in this section we prove that linear languages are exactly those languages which are accepted by the 2-head automata.

The proof consists of two parts.

Theorem 1. *Every linear language is accepted by a 2-head finite automaton.*

Proof. The proof is constructive. Let us start by a grammar of the linear languages of normal form presented in Lemma 1. Let the states of the automaton are the non-terminal symbols of the grammar with initial state $s = S$. Put a new state to the automaton as the final state. The alphabet V is the same as in the grammar. Now we give the transition function: for each rule of the form $A \to aB$ let a transition be $B \in d(A, a, \varepsilon)$. For the rules of type $A \to Ba$ let a transition be $B \in d(A, \varepsilon, a)$. Finally for the rules $A \to a$ let the final state is in $d(A, a, \varepsilon)$. It is easy to show by the construction that each derivation has a one-to-one correspondence with a run of the automaton. Therefore, the automaton accepts the linear language generated by the grammar exactly. □

Theorem 2. *Every language accepted by a 2-head finite automaton is a linear language.*

Proof. Now we construct a linear grammar based on the given automaton. Let the set V be the same for both the automaton and the grammar. Let the non-terminals be the representations of the states of the automaton, let S represent the initial state. The rules of the grammar will be generated from the transition function. For each transition $B \in d(A, a, b)$ let the rule $A \to aBb$ in H $(a, b \in (V \cup \{\varepsilon\}))$. Finally, for all final-state F the rule $F \to \varepsilon$ is given. It is easy to check that every run of the automaton has a unique derivation in the grammar and vice-versa. So, the grammar generates the same language as the accepted language of the automaton. □

As a special consequence of the previous theorems and constructions we can define a 'normal form' for this type of automata.

Consequence 1 *For each 2-head finite automaton there is an equivalent one (accepting the same language as the original one) with only transitions of the forms* $B \in d(A, a, \varepsilon)$ *and* $B \in d(A, \varepsilon, a)$.

This fact is based on the normal form of linear grammars presented in Lemma 1.

In figures the transitions of a normal form automaton can be (a, ε) and (ε, a). We can use the alternative notions $\to a$ and $\leftarrow a$ to indicate the direction of the moving head.

Now let us see a famous example for a linear language: namely, the palindrome language. This language contains all the words which are read in exactly the same way both forwards and backwards.

In Figure 3 the automaton of this language can be seen over the binary alphabet. This automaton is in 'normal form': at each transition the arrow shows which head is moving by reading the terminal letter.

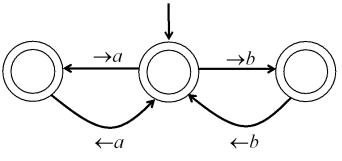

Fig. 3. 2-head finite automaton accepting the palindrome language

Now we show that the deterministic version of these automata is weaker: i.e. they do not accept all linear languages.

Consider the linear language $\{a^n b^n\} \cup \{a^{3n} b^n\}$ $(n > 0)$. It is clear that it can be accepted by a 2-head non-deterministic finite automata trying both possibilities to check in a non-deterministic way. For a deterministic automaton it should be decided which head moves in which step. With a finite control it is impossible to know at first how many steps of the first head should be followed by a step of the second head.

4.1 Characterization of even linear languages

In this section we will use a special subclass of linear languages: namely, the even-linear ones. Note that the even-linear language class strictly contains the class of regular languages.

Theorem 3. *The 2-head automata using transitions type $B \in d(A, a, b)$ and $C \in d(A, a, \varepsilon)$, where C is a final state and $a, b \in V$ accepts exactly the even-linear languages, if there is not any transition from the final states which can be reached by a transition type $C \in d(A, a, \varepsilon)$.*

Proof. Using the same constructions as in Theorems 1 and 2 the result will be a special linear grammar: namely, even-linear one. The automaton has transitions without finishing the read of the input word only type $B \in d(A, a, b)$. It means that both heads must move one step at the same time (using the normal form of Lemma 1 the relation is obvious). The only exception is when the input has only 1 unread letter. In these cases only the first head steps finishing the word and accept it. Since there is no transition from the final state can be reached with only a transition using 1 head, the automata must stop even if the input has unread letters. □

In the special automata above the process cannot be continued from a final state which is reached by a step not type $B \in d(A, a, b)$, but from Consequence 1 we know that these automata can be translated to automata in 'normal form'.

Now let us examine the deterministic version of these restricted automata.

Theorem 4. *The 2-head deterministic finite automata using transitions type $B \in d(A, a, b)$ and $C \in d(A, a, \varepsilon)$, where C is a final state and $a, b \in V$ accepts exactly the even-linear languages, if there is not any transition from the final states which can be reached by a transition type $C \in d(A, a, \varepsilon)$.*

Proof. First, it is trivial (and it is a consequence of the previous theorem) that the languages accepted by the deterministic version must be even-linear languages. Now, we will prove that all even-linear language can be accepted by deterministic 2-head automata having the above properties. Let us use a method similar to the

one for regular languages starting with a non-deterministic automata to receive a deterministic one. This construction is the so-called set-construction. Let us start with the automaton $\langle Q, s, V, d, F\rangle$ described in Theorem 3. Now let our new states (Q') be the possible subsets of the original set Q. Let $s' = \{s\}$. The same alphabet V is used for the deterministic automaton as well. Let the new transition function d' be determined in the following way. For every pair of $a, b \in V$ the state $q'_2 = d'(q'_1, a, b)$ such that $q_i \in q'_2$ if and only if there is a state $q_j \in q'_1$ for which $q_i \in d(q_j, a, b)$. For the possible transitions of another type $(q_i \in d(q_j, a, \varepsilon))$, let the deterministic transitions be $q'_F = d'(q'_1, a, \varepsilon)$ with a new state q'_F if q'_1 contains q_j. We allow these transitions for the automaton only in cases when both heads can read the same place (i.e. the middle) of the tape and, therefore, it is not possible to step with both heads simultaneously. Let the set of the final states be all the states q'_i which contains any of F and q'_F. Finally, the states which are not accessible from the initial one can be deleted. It is easy to see that this automaton is deterministic and recognizes the same even-linear language as the original one. □

4.2 Comparison with other 2-head automata

In this section we compare our 2-head automata with other 2-head automata.

Usually in the literature the heads of the 2-head automata can move in the same direction [2].

Note that in [8] the so-called Watson-Crick automata are described. They are highly similar to other 2-head automata in the literature from our view-point. The main difference between our automata and these is the following. In Watson-Crick automata both heads go in the same direction. (We do not want to describe here that the Watson-Crick automata usually work with double strings, such as double stranded DNA molecules.)

Now we want to show some examples of formal languages which can be accepted by the traditional 2-head automata and/or our new 2-head automata.

For instance the language contain all words in shape $a^n b^n$ $(n > 1)$ can be recognized by both the traditional and the new 2-head machine. (In Figure 4 the new automaton type can be seen for this language.)

A marked version of the so-called 'copy'-language $(\{wcw | w \in \{a, b\}^*\})$ can be recognized by a traditional non-deterministic 2-head machine. The languages $\{a^n b^n c^n | n \in \mathbb{N}\}$ and $\{a^n b^m c^n d^m | n, m \in \mathbb{N}\}$ can be accepted by traditional deterministic 2-head machines. Since these languages are not even context-free ones, they are not accepted by any new type 2-head automata.

The new automata accepts the language $\{ww^R | w \in V^*$ and w^R is the reverse of the word $w\}$. This language cannot be accepted by any 2-head finite automata with heads moving to the same direction.

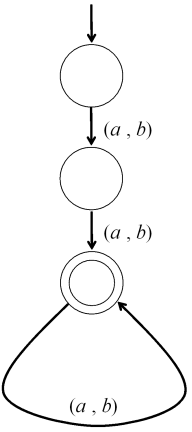

Fig. 4. A 2-head finite automaton accepting the language $a^n b^n$ $(n > 1)$

Finally we present a variation of the new 2-head automata. In the new variation the heads do not know the position of the other one. Both heads should read the whole word, but in different directions.

It is open to prove that this variation also accepts all the linear languages.

The automata may guess when the heads are in the same position and after this point it uses the opposite transitions as before. This means that if it was a transition to the first head with a terminal, then it will be a transition with the same terminal for the second head in this second phase, etc.; but it is not easy to prove that there are no false acceptances.

5 Conclusions

One can imagine our new automata as automata working on words which are doubled up (see Figure 5, where the tape can move).

To get all linear languages (not only the even-linear ones, in which each rule has the same number of terminals before and after the non-terminal (if any) on the right-hand-side) we allow transition steps in which only 1 head moves, while the other does not (it reads the empty word). A normal-form of the automata is presented. Using only 1 head in each step the work of the machine looks like the inverse of a derivation tree using the normal form (Lemma 1).

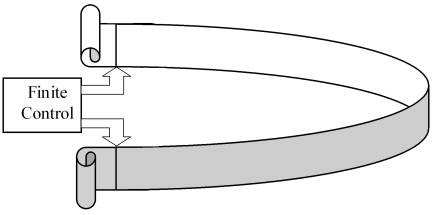

Fig. 5. 2-head finite automaton with a folded tape

One can use a special, restricted form of the automata – corresponding to the even-linear languages (for instance in normal form, the heads of the automaton can only step one after the other). Moreover it is proved that the even-linear languages can be accepted by deterministic machines as well.

So we applied the finite state machines to recognize a wider class of languages than the original ones with 1 head. These results make it possible to use linear languages as simply as regular ones. We would like to extend these automata to work on words which are folded several times to characterize other language classes as well.

It would also be interesting to analyse the differences among the languages accepted by the variations of the new and the language classes accepted by variations of the known (traditional) 2-head (for instance Watson-Crick) automata.

A note on related works

Between the time this paper is written and appeared some works are done in this and related topics, see, for instance [3, 4, 5, 7]. The class of languages that can be accepted by our deterministic machines are also characterized in [5].

References

1. Hopcroft, J. and J.D. Ullmann (1979). *Introduction to automata theory, languages, and computation.* Reading: Addison-Wesley, Reading.
2. Hromkovic, J. (1985). On one-way two-head deterministic finite state automata. *Computers and Artificial Intelligence*, 4/6: 503–526.

3. Nagy, B. (2007). On $5' \to 3'$ sensing Watson-Crick finite automata. In *Preproceedings of the DNA 13, The 13th International Meeting on DNA Computing*, pp. 327-336. Memphis.
4. Nagy, B. (2008). On $5' \to 3'$ sensing Watson-Crick finite automata. *Lecture Notes in Computer Science*, 4848: 256–262.
5. Nagy, B. (2009). On a hierarchy of $5' \to 3'$ sensing WK finite automata languages. *CiE 2009, Computability in Europe 2009: Mathematical Theory and Computational Practice, Abstract Booklet*, pp. 266-275. University of Heidelberg.
6. Nagy, B. (2012). Derivation "trees" and parallelism in Chomsky-type grammars. *Triangle*, 8: 101-120 (this volume).
7. Leupold, P. and B. Nagy (2009). $5' \to 3'$ Watson-Crick automata with several runs. In *Workshop on Non-Classical Models of Automata and Applications (NCMA)*, pp. 167–180. Wroclaw.
8. Păun, Gh., G. Rozenberg and A. Salomaa (1998). *DNA computing*. Berlin: Springer.
9. Petre, E. (2003). Watson-Crick-Automata. *Journal of Automata, Languages and Combinatorics*, 8(1): 59–70.
10. Rozenberg, G. and A. Salomaa (eds.) (1997). *Handbook of formal languages*. Berlin: Springer.
11. Sempere, J.M. (2000). On a class of regular-like expressions for linear languages. *Journal of Automata, Languages and Combinatorics*, 5(3): 343–354.
12. Sempere, J.M. and P. García (1994). A Characterization of Even Linear Languages and its Application to the Learning Problem. *LNAI*, 862: 38–44.