

ABSOLUTELY PARALLEL GRAMMARS AND
TWO-WAY DETERMINISTIC FINITE-STATE TRANSDUCERS

Vaclav Rajlich
Case Western Reserve University
Cleveland, Ohio 44106

SUMMARY

Absolutely parallel grammars are defined and it is shown that the family of languages generated is equal to the family of languages generated by two-way deterministic finite-state transducers. Furthermore it is shown that this family forms a full AFL [4], is properly contained in the family of languages generated by two-way nondeterministic finite-state transducers (which is equal to the family of checking automata languages [8]) and properly contains the family of nonexpansive context-free languages [7].

INTRODUCTION

Recently there has been an extensive investigation of different types of context-sensitive grammars with the context scattered through the whole word (see, [13], [10], etc.). The paper defines a new grammar belonging to this category, called an absolutely parallel grammar. A particular feature of this grammar making it worthwhile is that it describes in simple terms all languages generated by two-way finite-state deterministic transducers, which in turn play an important role in general understanding of two-way deterministic machines [1].

The paper is divided into four sections. In the first two sections, we present the definition of the two-way deterministic finite-state transducers and the absolutely parallel grammars. The equivalence of the two corresponding families of languages is established in the third section.

In the fourth section, the family of absolutely parallel languages is shown to be a full AFL [4] and some other closure properties are mentioned. It is shown that the family of absolutely parallel languages is properly contained in the family of checking automata languages [8] (and thus has the emptiness problem solvable) and that it properly contains all nonexpansive context-free languages [7]. Furthermore it is shown that two-way deterministic and nondeterministic transducers differ in their generative power.

1. TWO-WAY FINITE-STATE TRANSDUCERS

In this section we define two-way finite-state transducers, both deterministic and nondeterministic and provide the notation necessary to establish the results of the paper.

Intuitively, a two-way nondeterministic finite-state transducer consists of an input tape, finite control and output tape. The control can see at most one letter on each side of the input pointer, which can be moved back and forth. Furthermore, it can insert a word on the end of the output tape. These actions are specified by move-rules. Each move-rule tells exactly the situation in which it applies, i.e., the state of finite control and letters on both sides of input pointer, and action to be taken, i.e. the move of the pointer, the new state of the finite control and the word added to the end of the output tape.

This completes the informal description. We now proceed formally.

Definition 1.1.

Let K, I, O be three finite sets. A configuration over (K, I, O) (abbreviated "configuration" whenever K, I, O are obvious from the context) is any triple (p, xix', w) , where $p \in K$, $xx' \in I^*$, $w \in O^*$, and $\uparrow \notin K \cup I \cup O$ (call this symbol "pointer"). Then K, I, O are called set of state, input and output symbols, respectively.

Definition 1.2.

A move-rule over (K, I, O) (abbreviated "move-rule") is a 5-tuple $\mu = (p, a\uparrow b, q, a_1\uparrow b_1, y)$, where $p, q \in K$, $y \in O^*$ and $a, b \in I \cup \{\lambda\}$, $ab = a_1b_1$, (λ denotes here the empty word).

For each move-rule μ and configurations $U, V, U \xrightarrow{\mu} V$ iff there exist $x, x' \in I^*$ and $z \in O^*$ such that $U = (p, xa\uparrow b'x', z)$ and $V = (q, xa_1\uparrow b_1x', zy)$.

Definition 1.3.

A two-way nondeterministic finite-state transducer (abbreviated 2nft) is a triplet

$F = (M, s, K')$, where K_F, I_F, O_F are finite sets, $s \in K_F$ is start state, $K' \subset K_F$ is a set of terminal states and M is a finite set of move-rules over (K_F, I_F, O_F) .

Then $U \stackrel{M}{\vdash} V$ iff there exists $\mu \in M$ such that $U \stackrel{\mu}{\vdash} V$. Let $\stackrel{K}{\vdash}_M$ and $\stackrel{*}{\vdash}_M$ be relations on configurations defined in the usual way. If $(s, tx, \lambda) \stackrel{*}{\vdash}_M (p, xt, y)$ with $p \in K'$, we say x is accepted by F and y is generated by F .

The accepted language of F is the set $A(F)$ of all words accepted by F .

The generated language of F is the set $\mathcal{G}(F)$ of all words generated by F .

It is a well-known fact [12] that the accepted languages for 2nft are regular. Generated languages will be described in greater detail in the third and fourth sections of this paper.

Definition 1.4.

A two-way (deterministic) finite-state transducer (abbreviated 2ft) is a 2nft $F = (M, s, K')$ with the following restrictions:

- (i) For each $u, v \in I_F^*$ and each $p \in K_F$ there exists at most one move-rule $\mu \in M$ such that $(p, u \uparrow v, y) \stackrel{\mu}{\vdash} (p', u' \uparrow v', y')$.
- (ii) There doesn't exist any move-rule of the form $(p, a \uparrow b, q, a_1 \uparrow b_1, y)$ where $p \in K'$.

It is easy to show that family of languages generated by 2ft and 2nft are equivalent to analogous families of [1] and [2].

For the proof of some theorems of this paper, it is convenient to use a new device, closely related to the previous one.

Definition 1.5.

A left 2ft $E = (M, s, K')$ is defined analogously as 2ft with the following difference:

- x is accepted and y is generated by E iff $(s, tx, \lambda) \stackrel{*}{\vdash}_M (p, tx, y)$ where $p \in K'$.

Lemma 1.1.

The families of languages generated by 2ft and left 2ft are equal.

Definition 1.6.

A normalized left 2ft is a left 2ft F such that

- (i) each move-rule is of the form either

$(p, \uparrow a, q, a \uparrow, y)$, denoted also $R(p, a, q, y)$ or $(p, a \uparrow, q, \uparrow a, y)$, denoted $L(p, a, q, y)$.

- (ii) $K_F \cap I_F = I_F \cap O_F = O_F \cap K_F = \emptyset$
- (iii) $K' = \{q\}$ contains one state symbol only, $q \neq s$.

Lemma 1.2.

There exists an effective procedure to find to each left 2ft F a normalized left 2ft F' such that $\mathcal{G}(F) = \mathcal{G}(F')$.

Now we can introduce the following concept:

Definition 1.7.

Let $F = (M, p_0, K')$ be a left 2ft and let us have a finite sequence of configurations $\{U_i\}_{i=0}^n$ such that $U_0 = (p_0, tx, \lambda)$,

$U_n = (p_n, \uparrow xy, w_n)$ with $p_n \in K'$, $U_i \stackrel{\mu_i}{\vdash} U_{i+1}$ and μ_i is of the form $(p_i, a_i \uparrow b_i, p_{i+1}, c_i \uparrow d_i, z_i)$. Then divide all such configurations U_i into two

classes according to the position of the pointer relatively to the boundary between x and y :

$U_i \in \mathcal{L}$ iff $U_i = (p_i, x_i \uparrow x_i' y, w_i)$ and $U_i \in \mathcal{R}$ iff $U_i = (p_i, x y_i \uparrow y_i', w_i)$ where $y_i \neq \lambda$. For the purpose of simpler notation, we will always assume that U_{-1} and $U_{n+1} \in \mathcal{L}$. Then define

$P_i(xty)$ in the following way:

- (i) $P_i(xty) = p_i z_i$ iff $U_{i-1} \in \mathcal{R}$, $U_i, U_{i+1} \in \mathcal{L}$.
- (ii) $P_i(xty) = p_i$ iff $U_{i-1}, U_i \in \mathcal{L}$, $U_{i+1} \in \mathcal{R}$.
- (iii) $P_i(xty) = z_i$ iff $U_{i-1}, U_i, U_{i+1} \in \mathcal{L}$.
- (iv) $P_i(xty) = \lambda$ in all other cases.

Then $L(xty) = P_0(xty)P_1(xty) \dots P_n(xty)$ is

called the left part of computation on xy .

$L(xty)$ has a particular intuitive meaning: it consists of those pieces of an output word which are generated while pointer moves within the word x . Those pieces are separated by the symbols of the states which are assumed by the finite control while entering or leaving the word x for "longer time". Note that the states in which the pointer hits the boundary of x from left and immediately departs back to right are not included.

Observe the following properties of $L(xty)$:

Observation 1.1.

$L(xty) = y_0 p_1 p_2 y_2 p_3 p_4 \dots p_{2n-1} p_{2n} y_{2n}$, where $y_i \in O_F^*$, $p_i \in K_F$, $n \geq 0$. Moreover $i \neq j$ implies $p_i \neq p_j$ (otherwise F "loops" and xy is not an accepted word).

2. ABSOLUTELY PARALLEL GRAMMARS

In this section, we define absolutely parallel grammars and provide the necessary notation.

Definition 2.1.

If N, T are finite sets, $N \cap T = \emptyset$, then an absolutely parallel grammar (abbreviated apg) is a 4-tuple $G = (N, T, S, P)$, where N, T, S are nonterminals, terminals and start symbol, respectively and P is a finite set of productions π of the form $(A_1, \dots, A_n) \rightarrow (y_1, \dots, y_n)$, where $A_i \in N$ and $y_i \in (N \cup T)^*$.

For $w, w' \in (N \cup T)^*$, $w \Rightarrow_{\pi} w'$ iff
 $w = u_1 A_1 u_2 A_2 \dots u_n A_n u_{n+1}$ and
 $w' = u_1 y_1 u_2 y_2 \dots u_n y_n u_{n+1}$, where $u_i \in T^*$.

For any set of productions P , $w \Rightarrow_P w'$ iff there exists $\pi \in P$ such that $w \Rightarrow_{\pi} w'$. Again, let \Rightarrow_P^k and \Rightarrow_P^* be relations on words defined in the usual way.

Definition 2.2.

The language generated by an apg G is $\mathcal{L}(G) = \{w | w \in T^* \text{ and } S \Rightarrow_P^* w\}$. Moreover define $\mathcal{L}^N(G) = \{w | S \Rightarrow_P^* w\}$. Family of all languages generated by an apg will be called family of absolutely parallel languages (apl).

Definition 2.3.

Let P' be a nonempty finite set, called code alphabet and let us have a bijection $C: P \rightarrow P'$ called coding function. Then for each production $\pi \in P$, $C(\pi)$ is called a code letter.

$A_1 \dots A_n$ will be called left side of π (abbreviated $ls(\pi)$) and if $y_1 \dots y_n = z_1 B_1 \dots z_m B_m z_{m+1}$, where $z_i \in T^*$, $B_i \in N$, then $B_1 \dots B_m$ will be called content of the right side of π (abbreviated $crs(\pi)$).

Definition 2.4.

An apg $G = (N, T, S, P)$ is in an indexed form iff

- (i) For each $ls(\pi)$, $i \neq j$ implies $A_i \neq A_j$.
- (ii) For each two productions $(A_1, \dots, A_n) \rightarrow (y_1, \dots, y_n)$ and $(B_1, \dots, B_m) \rightarrow (z_1, \dots, z_m)$ either $A_1 \dots A_n = B_1 \dots B_m$, or for each i, j , $A_i \neq B_j$.

- (iii) For every production π , $crs(\pi) \neq S$.

Lemma 2.1.

There exists an effective procedure to find to each apg G an apg G' in indexed form such that $\mathcal{L}(G) = \mathcal{L}(G')$.

3. MAIN RESULT

In this section, we present the main theorem of the paper and its proof.

Theorem 3.1.

The family of languages generated by 2ft is equivalent to the family of apl.

Proof of the theorem follows with the aid of the constructions and lemmas of this section.

Construction 3.1.

Let $G = (N, T, S, P)$ be an apg in the indexed form. We want to construct a left 2ft F such that each $w \in \mathcal{L}^N(G)$ will be "simulated" by $L(xty)$ for some $x, y \in I_F^*$. For this, we need to distinguish three state symbols for each non-terminal in N : states for which pointer moves to right, left and changes from right to left, respectively.

More formally, let $F = (M, S, \{S^L\})$ be a left 2ft where

- (i) $K_F = \{A, A^L, A^E | A \in N\}$, $O_F = T$, I_F is a code alphabet for P and $C: P \rightarrow I_F$ is coding function.
- (ii) In the following, let every $\pi \in P$ be of the form $(A_1, \dots, A_n) \rightarrow (y_1, \dots, y_n)$ where $y_i = u_{i,1} B_{i,1} \dots u_{i,m_i} B_{i,m_i} u_{i,m_i+1}$ and $B_{i,j} \in N$, $u_{i,j} \in T^*$, $m_i \geq 0$.
Let $C(\pi) = b$.

Then define

$$M_1 = \{R(A_i, b, B_{i,1}, u_{i,1}),$$

$$L(B_{i,m_i}^L, b, A_i^L, u_{i,m_i}) | \pi \in P, y_i \notin T^*\}.$$

$$M_2 = \{(B_{i,j}^L, b, B_{i,j+1}, b, u_{i,j+1}) |$$

$$\pi \in P, y_i \notin T^*(N \cup \{\lambda\})T^*\}$$

$$M_3 = \{R(A_i, b, A_i^E, y_i), L(A_i^E, b, A_i^L, \lambda) |$$

$$\pi \in P, y_i \in T^*\}.$$

$$\text{Let } M = \bigcup_1^3 M_i.$$

It is obvious that this construction is effective and could be applied to any app G in indexed form and therefore (by Lemma 2.1) to any app. It is easy to check that the resulting automaton is deterministic (conditions (i) and (ii) of Definition 1.4). Lemmas 3.1 and 3.2 relate an app G with the constructed left 2ft F.

Lemma 3.1.

$$\mathcal{L}(G) \subseteq \mathcal{L}(F).$$

Proof.

The Lemma is proved by the means of the statement:

- (1) If $w_1 A_1 \dots w_n A_n w_{n+1} \Rightarrow_{\pi_k} \dots \Rightarrow_{\pi_1} z \in T^*$,
 where $\pi_j \in P$, $k \geq 0$, $w_i \in T^*$, $A_i \in N$,
 then $z = w_1 x_1 \dots w_n x_n w_{n+1}$ and
 $(A_i, \uparrow C(\pi_z) \dots C(\pi_1), \lambda) \stackrel{*}{\vdash}_M$
 $(A_i^L, \uparrow C(\pi_z) \dots C(\pi_1), x_i).$

This statement can be proved by induction on k.

Lemma 3.2.

$$\mathcal{L}(G) \supseteq \mathcal{L}(F).$$

Proof.

The lemma is proved by the means of the statements:

- (2) $L(x_1 \uparrow x_2) = w_1 A_1^L \dots w_k A_k^L w_{k+1}$ where
 $k \geq 0$, $w_i \in T^*$ and $A_i \in N$.

This statement can be proved by induction on $|x_2|$.

- (3) Let $L(x_1 \uparrow x_2) = w_1 A_1^L \dots w_k A_k^L w_{k+1}$; then
 $w_1 A_1 \dots w_k A_k w_{k+1} \in \mathcal{L}^N(G).$

This statement is proved by induction on $|x_1|$ with the aid of statement (2).

Construction 3.2.

Let $E = (M, s, \{r\})$ be a normalized left 2ft. Then for each $p, q \in K_E$, $b \in I_E$, construct the following sets:

$$Y(p, q, b) = \{y_1(p_1, p_2) y_2 y_3(p_3, p_4) \dots (p_{2n-1}, p_{2n}) y_{2n} \mid$$

$$(p, \uparrow b, \lambda) \stackrel{\cdot}{\vdash}_M (p_1, \uparrow b, y_1), (p_{2i}, \uparrow b, \lambda) \stackrel{\cdot}{\vdash}_M$$

$$(p_{2i}^1, \uparrow b, y_{2i}) \stackrel{\cdot}{\vdash}_M (p_{2i+1}, \uparrow b, y_{2i} y_{2i+1}),$$

$$(p_{2n}, \uparrow b, \lambda) \stackrel{\cdot}{\vdash}_M (q, \uparrow b, y_{2n})$$

$$\text{and } i \neq j \text{ implies } p_i \neq p_j\}.$$

$$Z(p, q, b) = \{y_1 y_2 \mid (p, \uparrow b, \lambda) \stackrel{\cdot}{\vdash}_M (p', \uparrow b, y_1) \stackrel{\cdot}{\vdash}_M$$

$$(q, \uparrow b, y_1 y_2) \text{ for some } p' \in K_E\}.$$

Let $G_E = ((K_E \times K_E), 0_E, (s, r), P)$ where

$$P = \{(p_1, p_2), (p_3, p_4), \dots, (p_{2n-1}, p_{2n}) \rightarrow (z_1, z_2, \dots, z_n) \mid$$

$$n \geq 1, k \neq j \text{ implies } p_k \neq p_j \text{ and}$$

$$z_i \in Z(p_{2i-1}, p_{2i}, b) \cup Y(p_{2i-1}, p_{2i}, b)\}.$$

This completes the construction. Remaining lemmas of this section relate left 2ft E with the constructed app G_E .

Lemma 3.3.

$$\mathcal{L}(G_E) \subseteq \mathcal{L}(E).$$

Proof.

The Lemma is proved with the help of statement:

- (4) If $z_0(p_1, p_2) z_2(p_3, p_4) \dots (p_{2h-1}, p_{2h}) z_{2h}$
 $\xrightarrow{p}^k z_0 z_1 z_3 \dots z_{2h-1} z_{2h} \in O_E^*$,
 then there exists $u \in I_E^*$ such that

$$(p_{2i-1}, \uparrow u, \lambda) \stackrel{*}{\vdash}_M (p_{2i}, \uparrow u, z_{2i-1}).$$

Proof is by induction on k.

Lemma 3.4.

$$\mathcal{L}(G_E) \supseteq \mathcal{L}(E).$$

Proof.

The Lemma is proved with the help of statement:

- (5) If $L(x_1 \uparrow x_2) = z_0 p_1 p_2 z_2 p_3 p_4 \dots p_{2h-1} p_{2h} z_{2h}$,
 then $z_0(p_1, p_2) z_2(p_3, p_4) \dots (p_{2h-1}, p_{2h}) z_{2h} \in \mathcal{L}^N(G_E).$

Proof is by induction on $|x_1|$.

Proof of the Theorem 3.1.

We have established in Constructions and Lemmas of this chapter, that the family of languages generated by left 2ft is equivalent to the family of apl. Using Lemma 1.1 we get the Theorem 3.1.

4. SOME PROPERTIES OF AFL

Theorem 4.1.

The family of apl is a full AFL [3] closed under substitution.

Proof.

It was shown in [2] that apl are closed under substitution and intersection with regular sets. It is obvious that a^* is apl for every a . Hence by a theorem of [9], family of apl is a full AFL.

Observation 4.1.

Besides reversal [2], a particularly characteristic closure property for the family of apl is repetition: If L is an apl, then $\{w^n | w \in L\}$ is apl for any fixed integer n .

Lemma 4.1.

If $L \subset a^*$ and L is an apl, then L is regular.

Proof.

Let $G = (N, \{a\}, S, P)$ be the apg which generates L . Let $\ell = \max\{|w| \mid w = \ell_s(\pi) \text{ or } w = \text{crs}(\pi), \pi \in P\}$. Then let $G' = (N', \{a\}, S, Q)$ be a left-linear context-free grammar [2], where $N' = (N \cup \{\lambda\})^\ell$ and

$$Q = \{(A_1 \dots A_m) \rightarrow (B_1 \dots B_n)z_1 \dots z_n z_{n+1} \mid ((A_1, \dots, A_m) \rightarrow (y_1, \dots, y_m)) \in P \text{ and } y_1 \dots y_m = z_1 B_1 \dots z_n B_n z_{n+1} \text{ where } B_i \in N, z_i \in \{a\}^*\}.$$

Then obviously $L = \mathcal{L}(G) = \mathcal{L}(G')$.

It is known that languages generated by left-linear grammars are regular (for proof see [3]) and therefore $L = \mathcal{L}(G)$ is regular.

Theorem 4.2.

The family of languages generated by 2ft is properly contained in the family of languages generated by 2nft (see also [2]).

Proof.

Consider the language $L = \{a^n \mid n \text{ is not a prime}\}$. Then $L = \mathcal{L}(F)$ for some 2nft F . Moreover L is not regular, and therefore by the Lemma 4.3, it is not an apl.

Lemma 4.2.

The family of nonexpansive context-free languages [7] is properly contained in the family of apl.

Proof.

We will use [7], Theorem 4.2:

Any nonexpansive context-free language is equivalent to a derivation bounded set X for some context-free grammar $G = (N, T, S, P)$, i.e. $w \in X$ iff $S \Rightarrow_P w_1 \Rightarrow \dots \Rightarrow_P w_k \Rightarrow_P w$ and the number of nonterminal symbols in w_1, \dots, w_k is less than some fixed integer k .

Then we can easily construct an apg $G' = (N, T, S, Q)$, where $Q = \{(A_1, \dots, A_n) \rightarrow (y_1, \dots, y_n) \mid 1 \leq n < k \text{ and if } (A_1 \rightarrow w_1) \in P, \text{ then } y_1 = A_1 \text{ or } y_1 = w_1\}$. Then $\mathcal{L}(G') = \mathcal{L}(G)$.

Nonexpansive cfl are properly contained in apl, because they are not closed under properties from the Observation 4.2.

Finally we want to present a result, which is well-known, but which doesn't appear in the literature:

Theorem 4.3.

The family of languages generated by 2nft is equivalent to the family of checking automata languages.

Corollary 4.1.

The emptiness problem is solvable for apl.

ACKNOWLEDGEMENT.

The author wishes to thank Professors Gene F. Rose and William Rounds for valuable suggestions and encouragement.

REFERENCES

1. Aho, A.V., Ullman, J.D., A Characterization of Two-Way Deterministic Classes of Languages. In "IEEE Conference Record of 1969 Tenth Annual Symp. on Switching and Automata Theory", pp. 231-239. Institute of Electrical and Electronics Engineers, New York, 1969.
2. Ehrlich, R. and Yau, S.S., Two-Way Sequential Transducers and Stack Automata.
3. Ginsburg, S., "The Mathematical Theory of Context-Free Languages", McGraw-Hill Co., New York, 1966.
4. Ginsburg, S. and Greibach, S.A., Abstract Families of Languages, Mem. Amer. Math. Soc. 87 (1969), 1-32.
5. Ginsburg, S., Greibach, S.A. and Harrison, M.A., One-Way Stack Automata, J. Assoc. Comput. Mach. 14 (1967), 389-418.

6. Ginsburg, S. and Rose, G.F. Preservation of Languages by Transducers, Inform. Control 2 (1966), 153-176.
7. Ginsburg, S. and Spanier, E.H., Derivation-Bounded Languages, In "IEEE Conference Record of 1968 Ninth Ann. Symp. on Switching and Automata Theory", pp. 306-314. Institute of Electrical and Electronics Engineers, New York, 1968.
8. Greibach, S.A., Checking Automata and One-Way Stack Languages, J. Comput. System Sci. 3 (1969), 196-217.
9. Greibach, S.A. and Hopcroft, J., Independence of AFL Operations, Mem. Amer. Math. Soc., 87 (1969), 33-40.
10. Greibach, S.A. and Hopcroft, J., Scattered Context Grammars, J. Comput. System Sci. 3 (1969), 233-247.
11. Hartmanis, J., Shank, H., On the recognition of Primes by Automata, J. Assoc. Comput. Mach. 15 (1968), 382-389.
12. Hopcroft, J. and Ullman, J., "Formal Languages and their Relation to Automata", Addison-Wesley Publishing Co., Reading, Mass., 1969.
13. Nash, B.O. and Cohen, R.S., Parallel Leveled Grammars, In "IEEE Conference Record of 1969 Tenth Annual Symposium on Switching and Automata Theory", pp. 263-276. Institute of Electrical and Electronics Engineers, New York, 1969.
14. Rose, G.F. Personal communication.