SERIAL COMPOSITION OF 2-WAY FINITE-STATE TRANSDUCERS AND SIMPLE PROGRAMS ON STRINGS

Michal P. Chytil
Vojtěch Jákl

Charles University
Malostranské nám. 25
118 00 Praha 1 - M. Strana
Czechoslovakia

Introduction

Hopcroft and Ullman [1] described an algorithm for reverse run of deterministic generalized sequential machines. This algorithm does not go beyond abilities of 2-way finite automata. The algorithm was then used [2] in the proof of the fact that serial composition of a deterministic gsm A_1 and a 2- way deterministic finite-state transducer A_2 can be replaced by a single 2-way deterministic finite-state transducer, say A_3. Let us recall the main idea of the proof (c.f. fig. 1).

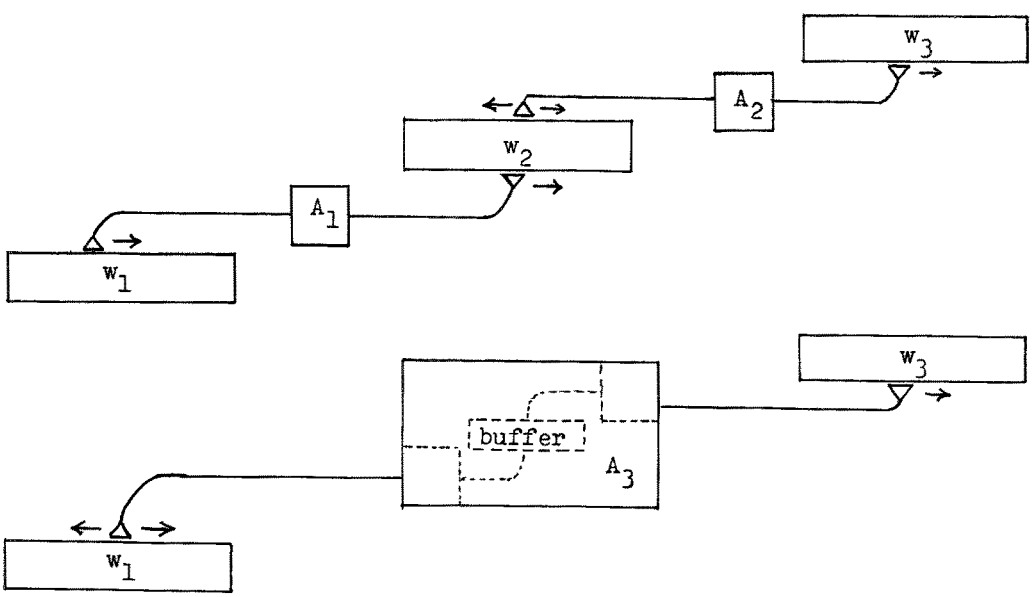

Fig. 1

A_3 in fact simulates step by step the computation of A_2 on the tape w_2. A_3 has not, however, all the tape w_2 at his disposal. It has only a segment of w_2 stored in a finite buffer. It is the segment which is in the given moment scanned by A_2. The input head of A_3 is placed at the square of w_1 which was translated by A_1 into the segment of w_2 currently stored in the buffer. If A_2 moves its head to the right, A_3 updates the content of the buffer moving its head rightwards and simulating one or more following steps of A_1. If A_2 moves its head leftwards, then A_3 updates the content of the buffer employing the algorithm for reverse run of A_1 and reconstructs one or more preceding steps of A_1.

In this paper we introduce an algorithm for reverse run of more general kind of machines, so called quasideterministic gsms. This algorithm is then employed in the proofs of some results about 2-way finite-state transducers and the functions computed by them.

The fact that the functions computable by 2-way finite transducers an closed under composition is then used in investigation of simple programs on strings.

Quasideterministic devices

A nondeterministic gsm A is quasideterministic if for every input w there is at most one computation over w beginning in an initial state and reaching a final state.

Then for every mapping (as defined e.g. in [3]) $A:X^* \to \mathcal{P}(Y^*)$ performed by a quasideterministic gsm A is $\mathrm{card}A(w) \leq 1$ for every $w \in X^*$. Such a mapping can be in an obvious way identified with a (partial) mapping $A:X^* \to Y^*$. Every mapping of this kind will be called quasideterministic gsm-mapping in this paper.

Example 1. Let g be a 1-1 mapping computable by a sequential machine. Then the inverse mapping g^{-1} is a quasideterministic gsm mapping.

Outline of the proof. In the state diagram of the Mealy sequential machine M performing g "reverse arrows" and interchange the role of input and output symbols. The results is a quasideterministic gsm for g^{-1}.

Example 2.

This example is connected with deterministic 2-way finite-state transducers.

By a deterministic 2-way finite-state transducer (2-DFT) we shall mean a deterministic device with finite-state control unit, 2-way read - only input head and a 1-way output head. The output head can print a word in an output alphabet in every step of computation. We suppose that the input words are bounded by a pair of endmarkers $\not\in,\not\ni$. Every computation starts in the initial state on the left endmarker. It stops by leaving the input tape. Every 2-DFT determines a (partial) function computable by 2-DFT.

By the history for 2-DFT M and an input word $\not\in x_1 \dots x_n \not\ni$ we mean a word

$$\mathrm{Hist}(M,w) = (\not\in, q_{01}, q_{02}, \dots, q_{0r_0})(x_1, q_{11}, \dots, q_{1r_1}) \dots (\not\ni, q_{n+1,1}, \dots, q_{n+1,r_{n+1}}),$$

where $q_{ij} \in Q \times \{L,R\}$. $q_{ij} = (q,L)$ (or $q_{ij} = (q,R)$) means that visiting the i-th square for the j-th time, M is in state q and enters the square from the left (or from the right).

If the computation of M on w is nonterminating, then $\mathrm{Hist}(M,w)$ is undefined.

Lemma 1. The mapping

$$w \longrightarrow \mathrm{Hist}(M,w)$$

is a quasideterministic mapping for every 2-DFT M.

Outline of the proof. By records we shall mean the elements composing histories, i.e. records are sequences of the form $(x,q_1,\dots,q_2)$, where $x \in X \cup \{\not\in,\not\ni\}$ and $q \in Q \times \{L,R\}$. For arbitrary M there is only a finite number of records which can occur in some terminating computation, because no square can be visited two times in the same state. The set R of these admissible records is therefore finite.

Let us construct a gsm G whose state-space and output alphabet are both equal to R. The control function δ of G is defined as follows:

$$[(x',q_1',\dots,q_s'),(x,q_1,\dots,q_r)] \in \delta((x,q_1,\dots,q_r),z) \Leftrightarrow_{df}$$

$$\Leftrightarrow_{df} \quad x = z \;\&\; \{(x',q_1',\dots,q_s') \text{ can occur as the right neighbour of } (x,q_1,\dots,q_r) \text{ in some history of a computation}\}$$

In every pair $(r_1,r_2) \in \delta(\dots.)$, r_1 represents the next state and r_2 the symbol to be printed.

The set $S \subseteq R$ of the initial states of G is formed by all records with the first component equal to $\not\in$ and the final set

$F \subseteq R$ of G is formed by all records with the first component equal to $\not\in$.

Every successful computation of G for input w outputs a history of a successful computation of M over w. Since M is deterministic, there is at most one such a computation. Different computations of G yield different outputs and therefore there is at most one successful computation for any input w. G is quasideterministic. □

A nondeterministic gsm G_1 is <u>dual</u> to a nondeterministic gsm G_2 iff the state diagram of G_1 can be obtained from the state diagram of G_2 by reversing arrows and interchanging the role of initial and final states.

The following lemma is obvious.

<u>Lemma 2</u>. The nondeterministic gsm dual to a quasideterministic gsm is quasideterministic.

Let us describe the algorithm for reverse run of quasideterministic gsm.

Let G_1 be a quasideterministic gsm, Q, X, Y, δ , S and F its state-space, input alphabet, output alphabet, control function, initial and final set, respectively. Let G_2 be the gsm dual to G_1. G_2 is quasideterministic by Lemma 2. Let D_1 be the deterministic finite acceptor derived from G_1 by the standard subset construction, D_2 the deterministic acceptor derived by the subset construction from G_2.

Then the state of D_1 after reading $\not\in x_1 \dots x_{i-1}$ is equal to the set

$$K_1^i = \delta(S, \not\in x_1 \dots x_{i-1}) = \{q \in Q;\ q \text{ is accessible by } Q \text{ from an initial state by reading } \not\in x_1 \dots x_{i-1}\}$$

The state of D_2 after scanning $\not\in x_n x_{n-1} \dots x_i$ is equal to the set

$$K_2^i = \{q \in Q;\ G_2 \text{ can access } q \text{ from a state of } F \text{ by reading } \not\in x_n \dots x_i\}$$

Since G_1 is quasideterministic, card $(K_1^i \cap K_2^i) \leq 1$. The set $K_1^i \cap K_2^i$ contains one element iff there exists a successful computation of G_1 for input $\not\in x_1 \dots x_n \not\in$ (cf. Fig. 2).

Now it is easy to see how the algorithm for the reverse run of G_1 works. It keeps K_1^i and K_2^i in its finite memory, when the head is visiting the i-th square. $K_1^i \cap K_2^i$ determines uniquely the state of G_1 during the successful computation, if there is any.

To reconstruct the situation of G_1 at the i-1 -th square, it is

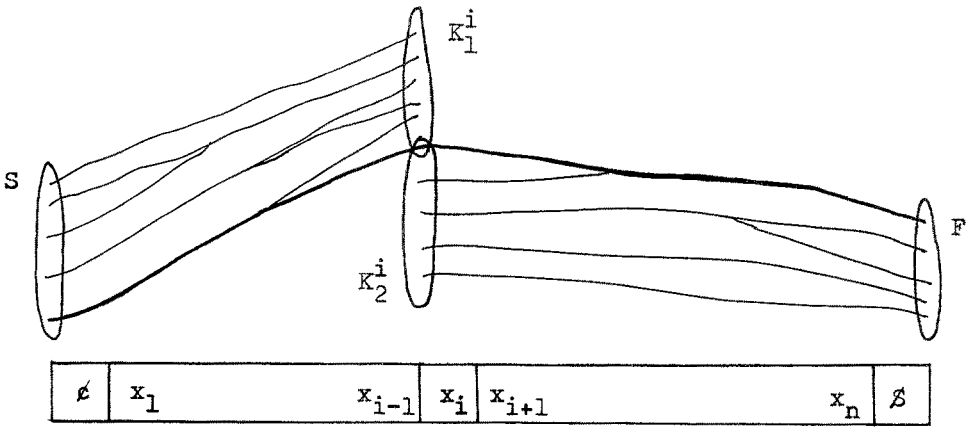

Fig. 2

sufficient to simulate one step of the deterministic automaton D_2 - - K_2^{i-1} is constructed - and to reconstruct K_1^{i-1}. Since D_1 is deterministic, K_1^{i-1} can be reconstructed employing the algorithm of Hopcroft and Ullman mentioned in the introduction to this paper.

All this apparently can be done by a 2-DFT. Evaluation of the next state of G_1 is the same problem as computing one step in the reverse run of G_2 and is solved by the dual procedure.

The algorithm starts by computing K_1^1 and K_2^1. It makes no problem, as $K_1^1 = \delta(S, ¢)$ is a fixed set and K_2^1 is computed by scanning the tape backwards and simulating D_2.

Theorem 1. If g is a quasideterministic gsm mapping and if f is a mapping computable by a 2-DFT, then $f \circ g$ is computable by a 2-DFT.

The proof of the theorem is quite analogous to the proof of the theorem by Aho, Hopcroft and Ullman [2] outlined in the introduction. It suffices to replace the algorithm for reverse run of deterministic gsm by the above described algorithm.

Recall a lemma from [4] .

Lemma 3 (Aho, Ullman). Let M_1 be a 2-DFT computing a function f. Let g be another function computable by 2-DFT. Then there is a 2-DFT M_2 computing a function h such that

$h(\mathrm{Hist}(M_1, w)) = g \circ f(w)$, for all w.

Lemma 1, Theorem 1 and Lemma 3 immediately imply the following theorem.

Theorem 2. (Aho, Ullman [4]) The class of functions computable by 2-DFT is closed under composition of functions.

We close the paragraph by two results which can be proved analogously to Theorem 2 using appropriate modifications of Lemma 1.

Corollary. Let f be a function computable by a deterministic bounded-crossing transducer (cf. [5]) and let g be a function computable by 2-DFT. Then $g \circ f$ is computable by 2-DFT.

By quasideterministic 2-way finite transducer (2-QFT) we shall mean a nondeterministic 2-way finite transducer with the property that for every input word $\not\vdash w \dashv$ there is at most one terminating computation starting on $\not\vdash$ in an initial state.

Corollary. Let f be computable by 2-QFT and let g be computable by 2-DFT. Then $g \circ f$ is computable by 2-DFT.

Consequently, a function is computable by 2-QFT iff it is computable by 2-DFT.

Simple programs on strings

Various kinds of automata have been investigated as devices for recognizing and generating languages. On the other hand, D. Scott [6] expressed the idea that "functions are better than sets", i.e. that investigation of functions computable by automata could prove to be in some aspects more fruitful approach. The results of this paragraph support the idea.

We shall define a simple programming language LOOP 2 which is a little more powerful than the language LOOP and the language LOOP with weak sum introduced and studied by Ausiello [7] and Ausiello Moscarini [8]. We shall prove that the functions programmable in LOOP 2 are exactly the functions computable by 2-DFT. The existence of such a language bears upon the fact that the class of functions is closed under composition. For, if $P_1;P_2$ should be a program whenever P_1 and P_2 are programs, then the closure under composition of functions is usually unavoidable.

To express the semantics of LOOP 2 - programs, we have to consider each identifier in a program to be the name of a register containing a word in a given alphabet. We assume, besides, the existence of a special control register, which is used during execution of loops. Single symbols of strings in the control register are accessible for inspecting (but not rewriting) through a "window" which can be moved backwards and forwards.

Then the LOOP 2 programs operating on strings in an alphabet $X = \{x_1,\ldots,x_n\}$ can be defined inductively as follows. We explain the semantics paralelly with the syntax of the programs.

1) elementary statements

a) $\langle id\rangle := \varepsilon$: clear register $\langle id\rangle$

b) $\langle id_1\rangle := \langle id_2\rangle$: store the content of $\langle id_2\rangle$ in $\langle id_1\rangle$

c) $\langle id\rangle := \langle id\rangle \circ$ "$\langle const\rangle$" : compute the right $\langle const\rangle$-successor of the word contained in $\langle id\rangle$ (i.e. the word consisting of the word from $\langle id\rangle$ followed by $\langle const\rangle$, where $\langle const\rangle$ is a word in X^*) and store the result in $\langle id\rangle$.

(We shall refer to statements of the types a) - c) as to basic statements.)

d) left : move the window of the control register one position leftwards

e) right : move the window one position rightwards

f) (conditional statements)

if $\langle id\rangle \neq \varepsilon$ then $S_1;\ldots;S_n$ else $S_{n+1};\ldots;S_m$ fi ,

where $S_1,\ldots S_m$ are elementary statements

: if the register $\langle id\rangle$ is nonempty then the sequence of statements $S_1,\ldots,S_n$ is executed, $S_{n+1};\ldots;S_m$ are executed otherwise.

if $\langle id\rangle \neq \varepsilon$ then $S_1;\ldots;S_m$ fi

2) elementary programs

An elementary program is every program of the form $S_1;\ldots S_n$, where $S_1,\ldots,S_n$ are elementary statements.

3) LOOP 2 programs

LOOP 2 programs willbe also called simple programs in the sequel. A simple program is each program satisfying one of the following conditions.

i) P is an elementary program,

ii) P is of the form

```
LOOP ⟨id⟩
 x_1: P_1
  ...
 x_n: P_n
END ,
```

where $P_1,\ldots,P_n$ are elementary programs,

iii) P is of the form $P_1;P_2$, where P_1, P_2 are simple programs.

The program

```
LOOP ⟨id⟩
 x_1: P_1
  ...
 x_n: P_n
END
```

is interpreted in the following way:

- the content of ⟨id⟩ (a string w) is stored in the control register and the control window is moved to the first symbol of the string w;
- while the window is attached to some symbol of w execute all the program P_i, if the window displays the symbol x_i (i.e. the statements right and left within P_i do not influence the running execution of P_i).

Theorem 3. All functions computable by 2-DFT can be programmed in LOOP 2.

Proof. Let M be a 2-DFT and $Q = \{q_0,\ldots,q_m\}$, X_1, X_2, $\delta: Q \times X_1 \longrightarrow Q \times X_2^* \times \{\text{left}, \text{right}\}$ and q_0 its state space, input alphabet, output alphabet, control function and the initial state, respectively. Denote $X = X_1 \cup X_2 = \{x_1,..,x_n\}$. Then the following simple program evidently computes the same function as M.

```
IN U; OUT V;  {declaration of input and output register - U contains
               the input string at the start of the computations, the
               other registers are empty. V contains the output word at
               the end of the computation}
Q_o := x_1;
LOOP U        {simulation of M}
x_1 : P_1
 ...
x_n: P_n
END           {end of the program}        , where
```

for $x_i \notin X_1$ is P_i = left; right {infinite cycle}

for $x_i \in X_1$ is $P_i = S_{io};\ldots;S_{im}$ and

S_{ij} = if $Q_j \neq \varepsilon$ then $Q_j := \varepsilon$; $Q_k := x_1$; V:=V."v"; S fi , where S $\in$ {right, left} and k, v and S are such that $\delta(q_j,x_i) = (q_k,v,S)$.

Theorem 4. All functions programmable in LOOP 2 are computable by 2-DFT.

Proof. Let P be a simple program and $X_1,\ldots,X_m$ all identifiers occurring in P. Suppose that all the identifiers are declared as input and at the same time as output ones. Then P defines a partial transformation of $(X^*)^m$.

We shall construct a 2-DFT A_P performing the same transformation as P in the following sense: input and output tapes of A_P have m tracks. Input m-tuple of strings $(w_1,\ldots,w_m)$ is encoded on the input tape as indicated in Fig. 3.

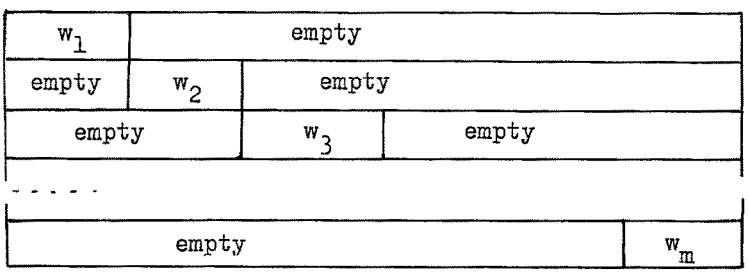

Fig. 3

The output m-tuple is encoded in the same way.

Let us describe the construction of A_P inductively, following the inductive definition of simple programs.

1) P is elementary

We shall describe three 2-DFT A_1,A_2,A_3 such that the serial composition of A_1,A_2 and A_3 will perform the same transformation as P.

Automaton A_1: A_1 has the program P stored in its finite control unit.

It also disposes of an auxiliary m-bit memory E. In every moment, the i-th bit of E is set to 1 iff the register X_i of P is nonempty in a corresponding moment of the execution of P.

At the beginning of the computation, A_1 inspects the input tape and learns which tracks are completely empty. This information is stored in E.

Then it prints out the input tape. After it (i.e. on the right of the string w_m in the m-th track), it prints a sequence of basic statements $S_1,\ldots,S_r$. This sequence describes step by step all actions of P changing the content of registers. A_1 constructs the sequence as follows:

it goes step by step through the program P stored in its finite memory and

- when it reaches a basic statement S in P, it prints S. Basic statements can clear a nonempty register or vice versa. Therefore A_1 updates E when printing S;
- it omits statements <u>right</u> and <u>left</u> (because they have no meaning in elementary programs);
- when a conditional statement is to be executed in P, A_1 decides (by inspecting E) whether the test condition of the statement holds or not and enters the corresponding branch of the program P.

The final form of the output tape is illustrated by Fig. 4.

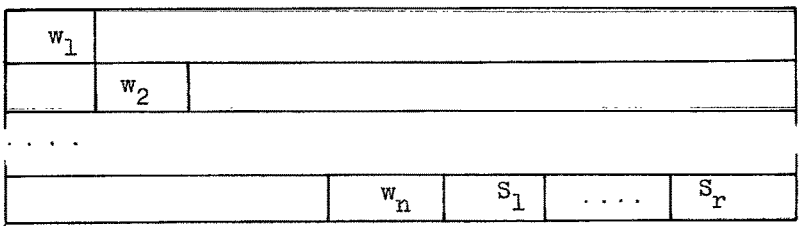

Fig. 4

<u>Automaton A_2</u>: A_2 disposes of an auxiliary finite memory R divided into m squares. Each of the squares can contain one of the symbols 1,2,...,m,<u>erase</u>. The content of R will be interpreted as references of registers to other registers. Denote the content of the i-th square by x_i. Then e.g. the reference $x_i = j$ means that X_i should be treated as X_j.

At the beginning of the computation, A_2 sets $x_i = i$ for all i. Then A_2 is scanning the input tape leftwrds and executing the statements $S_r,\ldots,S_1$ (in this order). Execution or more precisely interpreting of the statements of the type $X_i := \varepsilon$ and $X_j := X_k$ consists in changing

the content of the memory R while the remaining type of basic statements, i.e. $X_i := X_i \circ$ "⟨const⟩" causes an output printing. More precisely:

- scanning $X_i := \varepsilon$, A_2 assigns erase to each x_k such that $x_k = i$ and makes a left shift,
- scanning $X_i := X_j$, A_2 executes assignements $x_k := j$, for all x_k equal to i and makes a left shift,
- scanning $X_i := X_i \circ$"u", A_2 prints u in the k-th track, for all k such that $x_k = i$ and also proceeds leftwards.

When A_2 reaches the segment of the tape on the left of S_1, then it continues going leftwards and in each step the symbol which is in the k-th track on the input tape is printed into the i-th output track whenever $x_i = k$; blank is printed into the i-th track for x_i = erase.

Let us summarize the main idea of this procedure. If the program P assigns $X_i := X_j$ in the s-th step of computation, then all the history of X_i up to the s-th step is in fact cancelled and replaced by the history of X_j up to the s-th step. Similarly, the statement $X := \varepsilon$ means that X_i is cleared irrespective of all its history up to the s-th step.

Therefore, the automaton A_2 traverses the history of the execution of P backwards and if it has $x_i = j$ in its table of references R when scanning S_s then for its following computation in the i-th track, the computation in the j-th track is authoritative. If x_i = erase, then the remaining segment of the i-th track is to be erased.

When A_2 finishes its work, its output tape contains the final content of X_i in the i-th output track ($1 \leq i \leq m$). The output, however, need not be in the desired normalized form and blank segments can be scattered between symbols in each track.

Automaton A_3: A_3 rearranges the tape in the normalized fashion, i.e. it prints out the content of the first input track into the first output track omitting the blanks, then it prints the content of the second track and so on. (Recall Fig. 3.)

The serial composition of A_1, A_2, A_3 performs the same transformation of $(X^*)^m$ as P. By Theorem 2, the serial composition of A_1, A_2 and A_3 can be replaced by a single 2-DFT A_P.

2) P is of the form

LOOP X_1
x_1: P_1
...
x_n: P_n
END

The equivalent 2-DFT A_P can be again described by a serial composition of three 2-DFTs A_1', A_2, A_3. A_2 and A_3 are the same as in the preceding case. A_1' is a modification of A_1. It starts by initializing E and by reprinting the input tape in the same way as A_1. Then A_1' moves its head to the first symbol of w_1. If the first symbol is x_i, it starts by executing P_i similarly as A_1 executes all P with the exception that the statements right and left are not neglected as in the previous case. They are inerpreted as instructions controlling the moves of the head. If the way through P_i is accomplished then the symbol currently scanned by the head is used to determine which subroutine P_j is to be executed next. If the head is out of w_1 in such a moment, A_1' stops.

3) P is of the form $P_1;P_2$

where P_1 and P_2 are simple programs. Fictive variables can be introduced into P_1 and P_2 (e.g. using statements of the form X:= X) so that P_1 and P_2 contain the same identifiers as P. Then 2-DFT A_{P_1} and A_{P_2} equivalent to P_1 and P_2, respectively, exist by the induction assumption. The serial composition of A_{P_1} and A_{P_2} is equivalent to P and this composition can be replaced by a single 2-DFT by Theorem 2.

We have proved that for every program P there is an equivalent 2-DFT, provided all variables were declared as input and output ones. For the general case when variables $X_{i_1},\ldots,X_{i_r}$ are declared as the input ones and variables $X_{o_1},\ldots,X_{o_s}$ as the output ones, it suffices to construct the serial composition of 2-DFTs A_{in}, A_P, A_{out}, where A_{in} performs the input encoding $(X^*)^r \longrightarrow (X^*)^m$ for A_P by printing the input values into the appropriate tracks and leaving the remaining tracks empty, A_{out} on the other hand picks up the output values.

The serial composition of the automata performs the same mapping $(X^*)^r \longrightarrow (X^*)^s$ as P and can be replaced by a single 2-DFT.

References

1. J.E. Hopcroft and J.D. Ullman, An Approach to a Unified Theory of Automata, The Bell System Technical Journal 46 (1967), 1793-1829.

2. A.V. Aho, J.E. Hopcroft and J.D. Ullman, A General Theory of Translation, Mathematical Systems Theory 3 (1969), 193-221.

3. J.E. Hopcroft and J.D. Ullman, Formal Languages and Their Relation

to Automata, Addison-Wesley 1969

4. A.V. Aho and J.D. Ullman, A Characterization of Two-Way Deterministic Classes of Languages, JCSS 4 (1970), 523-538.

5. V. Rajlich, Bounded-Crossing Transducers, Information and Control 27 (1975), 329-335.

6. D. Scott, Some Definitional Suggestions for Automata Theory, JCSS 1 (1967), 187-212.

7. G. Ausiello, Simple Programs on Strings and Their Decision Problems, University of Rome, R.75-17, 1975.

8. G. Ausiello and M. Moscarini, On the Complexity of Decision Problems for Classes of Simple Programs on Strings, Proceedings of GI - 6. Jahrestagung (1976), Informatik - Fachberichte, Springer Verlag, 1976.