DESCRIBING AUTOMATA IN TERMS OF-LANGUAGES ASSOCIATED WITH THEIR PERIPHERAL DEVICES

by

Reino Kurki-Suonio

STAN-CS-75-493

MAY 1975

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Describing Automata in Terms of Languages Associated with Their Peripheral Devices

by

Reino Kurki-Suonio

Computer Science Department
Stanford University

and

University of Tampere, Finland

Abstract

A unified approach is presented to deal with automata having different kinds of peripheral devices. This approach is applied to pushdown automata and Turing machines, leading to elementary proofs of several well-known theorems concerning transductions, relationship between pushdown automata and context-free languages, as well as homomorphic characterization and undecidability questions. In general, this approach leads to homomorphic characterization of language families generated by a single language by finite transduction.

This research was supported in part by National Science Foundation grant GJ36473X and by the Academy of Finland. Reproduction in whole or in part is permitted for any purpose of the United States Government.

1. Introduction

Mathematical formulations of various classes of automata do not usually allow uniform treatment of different kinds of automata. One reason is in irrelevant differences of conventions. For instance, input may come from an input source, or it may constitute the initial contents of some storage device. Also, the formalisms which are used may be difficult to manage in a general situation. Functional notations, for instance, become quite clumsy to use when several storage devices are introduced.

The purpose of this paper is to present an approach, suggested by Floyd [1], where several kinds of automata are treated in a uniform way. In Sections 6 and 7 this approach will be applied to pushdown automata and Turing machines. The proofs of the theorems obtained are quite elementary, when compared to those usually given [2,8,9], and they have the advantage of being directly based on a simple and intuitively clear picture of the situations in question. In fact, most of the theorems are directly obvious from the basic definitions we shall make.

The general idea is to consider an automaton as a finite, non-deterministic transition system, where each transition is associated with a sequence of actions on peripheral devices, like inputs, outputs, tapes, stacks, and queues. Action sequences which are possible on an automaton are restricted by its transition system on one hand, and by the properties of the peripheral devices on the other hand. Each such restriction can be given in terms of a language over actions, and the total behavior of the automaton can then be described as the intersection of these languages.

The behavior of Turing machines has previously been described in terms of languages by Hartmanis [6]. However, he considers sequences of instantaneous descriptions instead of action sequences. The present approach is both simpler and more general. Ginsburg and Greibach have used it in [3] to exhibit an "intuitively obvious" language as a generator for one-way stack languages. Although it seems obvious that the general applicability of the approach has been known for some time, it is not known to the author that systematical use had been made of it.

Comparing the approach to that of (one-way nondeterministic) balloon automata [7] and abstract families of acceptors [4], there is the similarity that admissible behavior of memory devices is described Vocally", i.e., without reference to other aspects of the automata. However, our approach does not make explicit use of any information storage aspects of memory devices. Instead, each device is characterized purely in terms of a language of admissible action sequences. Letting these "peripheral languages" determine a class of automata, it will be shown in Section 8, that the families of languages associated with such classes are exactly those families which are generated by a single language by finite transduction. With a suitable restriction on "peripheral languages", full principal AFL [3] are obtained.

2. <u>Elementary Properties of Automata and Languages</u>

The purpose of this section is to state explicitly those concepts and properties of automata theory and formal languages on which our treatment of pushdown automata and Turing machines is based.

The basic concepts of finite automata, Turing machines, and the families of regular, context-free, context-sensitive, and recursively enumerable languages are assumed to be known. As for relationships between automata and languages, we assume that the relationships between (nondeterministic) finite automata and regular languges, and between 'Turing machines and recursively enumerable languages, are known. For questions of undecidability, one has to know the undecidability of the halting problem for Turing machines.

As for Boolean closure properties of families of languages, we shall make use of the facts that

- the family of regular languages is closed under intersection,
- the family of context-free languages is closed under intersection with regular languages,
- the family of context-sensitive languages is closed under intersection.

In addition, we shall need properties concerning <u>insertion of auxiliary letters</u> into words,' and <u>projection</u> homomorphisms deleting some letters from words.

Definition 1. Given two disjoint alphabets Σ and Σ_1, the projection of a word $x \in (\Sigma \cup \Sigma_1)^*$ into Σ^* is the word $x_\Sigma \in \Sigma^*$ obtained of x by deleting from it all occurrences of letters not belonging to Σ. For a language L over $\Sigma \cup \Sigma_1$ we define correspondingly,

$$L_\Sigma = \{x_\Sigma \mid x \in L\} .$$

Definition 2. Given a language L over Σ and an auxiliary alphabet Σ_1, $\Sigma \cap \Sigma_1 = \emptyset$, the language

$$L^{\Sigma_1} = \{x \mid x \in (\Sigma \cup \Sigma_1)^*, x_\Sigma \in L\}$$

is called insertion of Σ_1 into L.

Obviously, projection is the opposite of insertion in the sense that, for any language L over Σ and for an auxiliary alphabet Σ_1, $\Sigma \cap \Sigma_1 = \emptyset$, we have

$$(L^{\Sigma_1})_\Sigma = L .$$

For insertion we can also readily establish the following identities:

$$(L^{\Sigma_1})^{\Sigma_2} = (L^{\Sigma_2})^{\Sigma_1} = L^{\Sigma_1 \cup \Sigma_2} ,$$

$$L_1^{\Sigma_1} \cup L_2^{\Sigma_1} = (L_1 \cup L_2)^{\Sigma_1} ,$$

$$L_1^{\Sigma_1} \cap L_2^{\Sigma_1} = (L_1 \cap L_2)^{\Sigma_1} ,$$

$$(\Sigma \cup \Sigma_1)^* - L^{\Sigma_1} = (\Sigma^* - L)^{\Sigma_1} ,$$

where L, L_1, and L_2 are languages over Σ, and Σ_1, Σ_2 are two auxiliary alphabets, disjoint from Σ.

It is also easy to prove the following closure properties of language families under projection and insertion:

Theorem 1. The families of regular, context-free and context-sensitive languages are closed under insertion of an auxiliary alphabet.

Theorem 2. The families of regular and context-free languages are closed under projection.

3. Basic Definitions

By an automaton we understand a finite transition system associated with one or more devices for input, output, and storage of symbols. Each device is assumed in the following to have a finite set of primitive actions associated with it. For an input device, for instance, an input operation together with the letter obtained from the input source would constitute a primitive action. The transition system can be viewed as a finite digraph where edges are labelled with finite sequences of primitive actions. Vertices and edges of the graph are called states and state transitions, respectively. Two subsets of states are distinguished as initial and final states.

As an example, let us consider a simple automaton with one input and one counter. Assuming a two-letter input alphabet {a,b} , the primitive input actions could be denoted as (input a} , {input b) , and (input A) , where the first two correspond to successful input operations, while the third indicates that no letter was obtained since the source was found empty. For the counter we have a one-letter alphabet {1} together with primitive actions (push 1) , (pop 1) , and ⟨pop Λ⟩ denoting incrementation, decrementation, and the situation that the counter is found empty.

The transition system of the automaton is given in Figure 1. It can be immediately verified that a state transition sequence leading from the initial state to the final state is associated with exactly those input sequences where the number of a's equals the number of b's. Although this example presents deterministic behavior, we shall make no general restrictions on the action sequences associated with state transitions. For instance, we might have two transitions from the same state, one labelled with ⟨input a⟩⟨push 1⟩, the other with ⟨input a⟩⟨pop 1⟩, which would indicate nondeterministic behavior. Some transitions might even be labelled with impossible action sequences, like ⟨input Λ⟩⟨input a⟩.

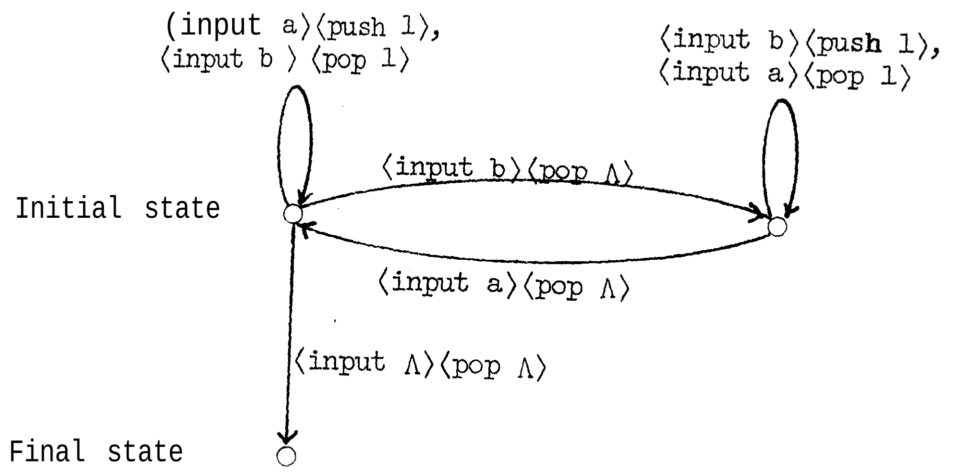


Figure1

In general, the behavior of an automaton is determined as follows:

1. Initially the automaton is required to be in one of the initial states. There may also be further requirements concerning the initial status of the various peripheral devices.
2. Operation of the automaton can terminate successfully whenever the current state is a final state, if all requirements (if any) concerning the final status of peripheral devices are satisfied.
3. If the operation is not terminated, one transition from the current state is selected, an attempt is made to perform the primitive actions associated with it (in the order indicated by the sequence), and, if this succeeds, the state is changed accordingly.

Nondeterminism is involved in the selection of the initial state, the selection of transitions, and in the decision on termination. Operation of the automaton terminates unsuccessfully, if the actions associated with the selected state transition cannot be performed. We are only interested in operations which do not terminate unsuccessfully.

An input device always has some input alphabet Σ_{input} . A word $x \in \Sigma^*_{\text{input}}$ is <u>accepted</u> by an automaton (<u>acceptor</u>), if the automaton can be operated (with successful termination) so that the word read in by the input actions is x . Similarly, an output device has an output alphabet Σ_{output} , and a word $y \in \Sigma^*_{\text{output}}$ is <u>generated</u> by the automaton (<u>generator</u>), if it can be operated (with successful termination) so that the output word given by output actions is y . The language accepted (generated) by an automaton is the set of words accepted (generated) by it.

A word $x \in \Sigma^*_{\text{input}}$ is <u>transduced</u> by an automaton (<u>transducer</u>) into $y \in \Sigma^*_{\text{output}}$, if the automaton can simultaneously accept x and generate y .

Notice that transduction is here defined only for words accepted by the automaton. A language L_1 is transduced into L_2, if L_2 is the set of words into which words of L_1 are transduced.

Two automata are <u>equivalent</u> as acceptors (generators) if they accept (generate) the same language. Two automata are <u>equivalent</u> as transducers if the transductions performed by them are the same. Notice that equivalence as transducers implies equivalence as acceptors and generators, but not conversely. By equivalence without further specification we shall understand equivalence in those respects applicable to the automata in question.

For each device there are certain restrictions determining which sequences of primitive actions on that device are <u>admissible</u>. The main purpose of these restrictions is to guarantee that the symbols fetched from a storage device correspond to those stored in it, and that no input is obtained from an input source already found empty. In addition, there may be restrictions on the initial and final status of the devices. For instance, storage devices may initially be assumed empty. Notice that input and output are considered "one-way" devices, not storage units which would allow re-examination or replacement of letters already treated once.

More formally, let A_i be the set of primitive actions on device i. It-is assumed that primitive actions can be renamed, if necessary, for avoiding conflicts. Therefore, it is always assumed that A_i for different devices are disjoint. For each i we have a language

$$B_i \subset A_i^*$$

consisting of all admissible action sequences on device i. Let A denote the union of A_i for an automaton. The finite transition system of the automaton then determines a <u>regular language</u>

$$B_{fts} \subset A^*$$

such that $\alpha \in B_{fts}$ iff α is associated with some sequence of state transitions leading from an initial state to a final state (irrespective of whether or not α makes sense for the peripheral devices). This leads to the following definition of an action sequence $\alpha \in A^*$ being admissible:

<u>Definition 3</u>. A word $\alpha \in A^*$ is called an <u>admissible action sequence</u> for an automaton, iff

(i) $\alpha \in B_{fts}$, and

(ii) $\alpha_{A_i} \in B_i$ for all devices i .

Introducing the notation

$$B_i^{\oplus} = (B_i)^{A-A_i} \quad ,$$

i.e., letting $B_i^{\oplus}$ denote the language obtained of B_i by inserting actions on all other devices to it, we can express admissible action sequences as the language

(1) $$B = B_{fts} \cap \bigcap_i B_i^{\oplus} .$$

It is pointed out that this definition implies each peripheral device being defined solely by the ways in which it can be locally manipulated. A device is completely characterized by the language B_i associated with it. Additional restrictions are required, if one wishes to introduce interdependencies between devices, like those in linear bounded automata or time/tape complexity classes of Turing machines.

In accordance with common terminology, automata having (in addition to a single input and/or output) no peripheral devices, one pushdown stack, one queue, or one tape, will be called finite automata, pushdown automata, Post machines, and Turing machines, respectively. Notice that input is considered an independent device in this definition of (nondeterministic) Post machines and Turing machines. Strictly speaking, these peripherals are not devices but classes of devices, as the language B_i of a pushdown stack, queue, or tape, depends on the alphabet used on the device.

4. Languages Associated with Peripheral Devices

Languages B_i associated with some common peripheral devices will be investigated in this section. As it is known that the same family of languages -- that of recursively enumerable languages -- is accepted (generated) by Post machines, by Turing machines, and by automata with two pushdown stacks, there is some redundancy in discussing pushdown stacks, queues and tapes separately. However, it is interesting to see how these different kinds of devices lend themselves to this treatment.

4.1 Input Actions

Given an input alphabet Σ_{input}, there is a primitive input action (input a) for each $a \in \Sigma_{input}$. The meaning of such a primitive action is to take the next letter from the input source and to find it to be letter a . In addition, there may be an action (input A) , which means that the input source is found empty by an attempted input operation. Obviously, no further input operations can find input letters, if the source has already been found empty.

The set of all primitive input actions is therefore either

$$A_{input} = \{\langle \text{input } a\rangle \mid a \in \Sigma_{input}\} ,$$

or

$$A'_{input} = A_{input} \cup \{\langle \text{input } \Lambda\rangle\} .$$

Correspondingly, the set of admissible sequences of input actions is either

$$B_{input} = A^*_{input} ,$$

or

$$B'_{input} = A^*_{input}\langle \text{input } \Lambda\rangle^* .$$

In each case this is a regular language.

For notational simplicity, we shall in the following make no distinction between a language over Σ_{input} and the corresponding language over A_{input}. This means that the language accepted by an automaton can be denoted as

(2) $\qquad B_{A_{input}}$

where B is the language of all admissible action sequences from (1). More precisely, this notation involves that projection and change of alphabet can be combined in a single notation for projection. Two automata with the same A_{input}, and with admissible action sequences B and B', are then equivalent as acceptors iff $B_{A_{input}} = B'_{A_{input}}$. It is easy to see that (input A) is, in fact, a superfluous action in the following sense:

Theorem 3. For any automaton with input actions A'_{input} there is an equivalent automaton with the' same peripherals and with input actions restricted to A_{input}.

Proof. Let X be an automaton with state set S and with actions A containing an input action (input A) . Another automaton Y , equivalent to X and with actions A-((input Λ)} , can now be constructed as follows. As states of Y take $S \times \{0,1\}$, and let (s,k) be an initial (final) state of Y iff s is an initial (final) state of X and k = 0 (k = 0,1) . The state transitions of Y will be determined so that for any transition $s \xrightarrow{\alpha} t$ of X , $s,t \in S$, $\alpha \in A^*$, Y will have the following transitions:

if $a_{A|_{input}} = \varepsilon$, then $(s,k) \xrightarrow{\alpha} (t,k)$, $k = 0,1$;

else, if $\alpha_{\{\langle input\ \Lambda\rangle\}} = \varepsilon$, then $(s,0) \xrightarrow{\alpha} (t,0)$;

'else, if $\alpha_{A|_{input}} \in B^!_{input}$, then $(s,k) \xrightarrow{\alpha_{A-\{\langle input\ \Lambda\rangle\}}} (t,1)$, $k = 0,1$.

On the basis of this construction, it is straightforward to verify, that for any admissible action sequence y of X there is an admissible action sequence γ' of Y' satisfying

$$\gamma' = \gamma_{A-\{\langle input\ \Lambda\rangle\}} ,$$

and conversely. Therefore X and Y are equivalent. □

On account of Theorem 3, it is no essential restriction that we shall always restrict input actions to A_{input} in the following.

4.2 Output Actions

Given an output alphabet Σ_{output} , the set of primitive output actions is

$$A_{output} = \{\langle output\ a\rangle \mid a \in \Sigma_{output}\} ,$$

where $\langle$output a$\rangle$ denotes the action of outputting an individual letter $a \in \Sigma_{output}$. The set of admissible output action sequences is simply

$$B_{output} = A^{*}_{output} .$$

As in connection with input, we shall make no notational distinction between a language over Σ_{output} and the corresponding language over $\{\langle output\ a\rangle \mid a \in \Sigma_{output}\}$. This means that the language generated by an automaton can be denoted as

(3) $$B_{A_{output}} .$$

Two automata with the same A_{output} , and with admissible action sequences B and B' , are then equivalent as generators, iff $B_{A_{output}} = B'_{A_{output}}$.

When considering transduction of a language L , the output language is correspondingly

(4) $$(L^{\oplus} \cap B)_{A_{output}} .$$

Two automata with the same A_{input} , A_{output} , and with admissible action sequences B and B' , are equivalent as transducers, iff for each $\alpha \in B$ ($\alpha' \in B'$) there exists $\alpha' \in B'$ ($a \in B$) such that $\alpha_{A_{input}} = \alpha'_{A_{input}}$ and $\alpha_{A_{output}} = \alpha'_{A_{output}}$.

Because of Theorem 3, there is complete symmetry between input and output actions. Given an automaton accepting (generating) a language L , this same language L is generated (accepted) by the automaton obtained of the original by interchanging input and output. Therefore, for a class of automata which is closed under interchanging input and output, the families of languages accepted and generated by these automata are the same. Such a family will be referred to as the family of languages associated with that class of automata.

4.3 Pushdown Actions

Given a pushdown stack with stack alphabet Σ_{stack}, there are primitive actions (push a) and (pop a) for each a $\epsilon \Sigma_{stack}$. The meanings of these actions are pushing a single letter a on top of the stack, and popping one letter from the stack and finding it a letter a. In addition, there may be an action (pop Λ) , which means that the stack is found empty by an attempted pop operation. The set of all primitive pushdown actions, A_{stack}, is therefore either

$$\{\langle \text{push } a\rangle \mid a \in \Sigma_{stack}\} \cup \{\langle \text{pop } a\rangle \mid a \in \Sigma_{stack}\}$$

or

$$\{\langle \text{push } a\rangle \mid a \in \Sigma_{stack}\} \cup \{\langle \text{pop } a\rangle \mid a \in \Sigma_{stack}\} \cup \{\langle \text{pop } \Lambda\rangle\} \quad .$$

The characteristic property of a stack is that all letters pushed into it can be popped out of it in the reverse order. A grammar for B_{stack}, the set of all admissible pushdown action sequences, can be directly based on this property. In order to get B_{stack} as simple as possible, we shall require further that a stack is empty both initially and finally.[1] If A_{stack} does not contain (pop Λ) , B_{stack} is then the Dyck language generated by the grammar

[1] Relaxing this requirement is discussed in Section 6.

$$
\begin{aligned}
S &\to \varepsilon \\
&\to \langle\text{push } a\rangle\, S\, \langle\text{pop } a\rangle \qquad \text{for all } a \in \Sigma_{stack} \\
&\to SS \quad .
\end{aligned}
$$

If $\langle\text{pop } \Lambda\rangle$ is allowed, the grammar has to take care that $\langle\text{pop } \Lambda\rangle$ can appear only when the stack is empty:

$$
\begin{aligned}
S &\to \langle\text{pop } \Lambda\rangle \\
&\to T \\
&\to SS \\
T &\to \varepsilon \\
&\to \langle\text{push } a\rangle\, T\, \langle\text{pop } a\rangle \qquad \text{for all } a \in \Sigma_{stack} \\
&\to TT \quad .
\end{aligned}
$$

In each case B_{stack} is a context-free language over A_{stack} .

4.4 Queue Actions

Given a queue with an alphabet Σ_{queue} , there are primitive actions $\langle\text{write } a\rangle$ and $\langle\text{read } a\rangle$ for each $a \in \Sigma_{queue}$. The meanings of these actions are writing a letter a to one end of the queue, and reading (and erasing) a letter from the other end and finding it to be letter a . In addition, there may be an action $\langle\text{read } \Lambda\rangle$ which means that the queue is found empty by an attempted read operation. The set A_{queue} of all primitive queue actions is therefore either

$$\{\langle\text{write } a\rangle \mid a \in \Sigma_{queue}\} \cup \{\langle\text{read } a\rangle \mid a \in \Sigma_{queue}\}$$

or

$$\{\langle\text{write } a\rangle \mid a \in \Sigma_{queue}\} \cup \{\langle\text{read } a\rangle \mid a \in \Sigma_{queue}\} \cup \{\langle\text{read } \Lambda\rangle\}$$

The characteristic property of a queue is that all letters written into it can be read from it in the same order. A grammar for B_{queue}, the set of all admissible queue action sequences, can be directly based on this property. We shall make the further requirement that a queue is empty both initially and finally. If A_{queue} does not contain $\langle$read $\Lambda\rangle$, we get the following grammar:

$$\begin{aligned}
S &\rightarrow \varepsilon \\
&\rightarrow T \\
T &\rightarrow \langle\text{write } a\rangle\langle\text{read } a\rangle \quad \text{for all } a \in \Sigma_{queue} \\
&\rightarrow TT \\
\langle\text{read } a\rangle\langle\text{write } b\rangle &\rightarrow \langle\text{write } b\rangle\langle\text{read } a\rangle \quad \text{for all } a,b \in \Sigma_{queue} .
\end{aligned}$$

Notice that the context-free productions generate all action sequences where each write action is immediately followed by the corresponding read action, and the context-sensitive production takes care of arbitrary "delaying" of read actions.

If $\langle$read $\Lambda\rangle$ is allowed, we only have to add the production

$$T \rightarrow \langle\text{read } \Lambda\rangle$$

to the grammar. Hence, we find that B_{queue} is in each case a context-sensitive language over A_{queue}.

4.5 Tape Actions

A tape allows actions for reading and writing, and for moving the tape in either direction. It is customary to include one read, one write, and one move operation in one primitive action, in this order. In the following we shall adopt the convention of writing first, then moving the tape and reading. Given tape alphabet Σ_{tape}, the set of primitive tape actions is then

$$A = \{(\text{write } a, \text{ left}, \text{ read } b) \mid a,b \in \Sigma_{tape}\} \cup$$
$$\{(\text{write } a, \text{ right}, \text{ read } b) \mid a,b \in \Sigma_{tape}\}$$

The meaning of these actions is that letter a is written on tape, tape head is moved by one square to the left or to the right, and the letter in this square is read and found to be letter b .

The characteristic property of a tape can be stated as follows: when some letter has been written in a square, the same letter will be read when-this square is reached for the next time. Let us consider only one half of this property by requiring that a letter written by an action moving to the right will be read when the same square is reached next time (by an action moving to the left). Requiring further that all letters written (moving to the right) are later read (moving to the left) and denoting the set of action sequences so obtained by L_1, we get the following grammar for L_1 :

$S_1 \to X_1$ (stands for a sequence which either is empty or starts with an action moving to the right)

$\to Y_1$ (stands for an action moving to the left)

$\to S_1 S_1$

$X_1 \to \varepsilon$

$\to$ (write a, right, read b) X_1 (write c, left, read a)

for all $a,b,c \in \Sigma_{tape}$

$\to X_1 X_1$

$Y_1 \to$ (write a, left, read b) for all $a,b \in \Sigma_{tape}$.

If only the second half of the characteristic property of a tape is taken, we get a similar language L_2 with grammar:

$$\begin{aligned} S_2 &\to X_2 \\ &\to Y_2 \\ &\to S_2S_2 \\ X_2 &\to \varepsilon \\ &\to \{\text{write } a\text{, left, read } b) \, X_2 \, (\text{write } c\text{, right, read } a\rangle \\ &\qquad\qquad \text{for all } a,b,c \in C_{tape} \\ &\to X_2X_2 \\ Y_2 &\to \langle\text{write } a\text{, right, read } b) \qquad \text{for all } a,b \in \Sigma_{tape} \quad . \end{aligned}$$

Imposing no restrictions on the initial and final contents of a tape, we can then express B_{tape} , the set of all admissible tape action sequences, as

$$B_{tape} = \text{init}(L_1) \cap \text{init}(L_2) \quad ,$$

where init(L) denotes all initial parts of words in L :

$$\text{init}(L) = \{x \mid xy \in L \text{ for some } y \}$$

For a context-free L , init(L) is also context-free, as will be seen in Theorem 5. Therefore, B_{tape} is a context-sensitive language which can be expressed as an intersection of two context-free languages over A_{tape} .

B_{tape} can also be characterized as the complement of a context-free language L over A_{tape} . A context-free grammar for L is obtained easily from the observation that at least one letter has to be read differently from what was written in thesquare. This leads to the following grammar (together with the above productions for X_1, Y_1, X_2, and Y_2):

$S \to$ (write a, right, read b) X_1 (write c, left, read d)

$\to$ (write a, left, read b) X_2 (write c, right, read d)

for all $a, b, c, d \in \Sigma_{tape}$, $a \neq d$

$\to TST$

$T \to \varepsilon$

$\to Y_1$

$\to Y_2$.

5. Serial Combination of Automata

It is often useful to consider an automaton as a serial combination of several automata, so that the output of one automaton is used as input to the next one. More formally, let T and U be two automata with disjoint sets of primitive actions $A_T = A_1 \cup A_{output}$, $A_U = A_2 \cup A_{input}$, where the same alphabet is associated with A_{input} and A_{output}, and A_1 and A_2 stand for the primitive actions of the other devices in T and U . An action sequence

$$\gamma \in (A_1 \cup A_2)^*$$

is defined to be admissible for the serial combination of T and U iff there are action sequences α and β, admissible for T and U, satisfying

$$(5) \quad \begin{cases} \alpha_{A_1} = \gamma_{A_1} \\ \beta_{A_2} = \gamma_{A_2} \\ \alpha_{A_{output}} = \beta_{A_{input}} \end{cases}$$

Concerning such serial combination of automata, we have:

Theorem 4. For any serial combination of (a finite number of) automata there is a single automaton, equivalent to this serial combination, having the same peripheral devices as the original automata, omitting the intermediate output/input devices.

[illegible]. Let T and U be two automata as described above. Without affecting generality, we can assume that each state transition of T (U) is associated with at most one output (input) action. A third automaton V, with the same peripheral devices as T and U (except the intermediate output and input), and equivalent to the serial combination of T and U can now be constructed as follows.

Let the state sets of T and U be S_T and S_U. The set $S_T \times S_U$ will then be taken as the state set of V, and (t,u) will be an initial (final) state of V iff t and u are initial (final) states of T and U. The state transitions of V will be determined as follows:

- for each transition $t \xrightarrow{a} t'$ of T, where $\alpha \in A_1^*$,

  take $(t,u) \xrightarrow{\alpha} (t',u)$ for all $u \in S_u$;

- for each transition $u \xrightarrow{\beta} u'$ of U, where $\beta \in A_2^*$,

  take $(t,u) \xrightarrow{\beta} (t,u')$ for all $t \in S_T$;

- for each pair of transitions

  $t \xrightarrow{a} t'$ of T,

  $u \xrightarrow{\beta} u'$ of u,

where $\alpha_{A_{\text{output}}} = \beta_{A_{\text{input}}} \neq \varepsilon$ (when actions (input a) and (output a) are equated), take

$$(t,u) \xrightarrow{\alpha_{\Delta_1}\beta_{\Delta_2}} (t',u') \quad .$$

The claimed equivalence can be easily verified as follows. Firstly, consider an arbitrary action sequence γ admissible for the serial combination of T and U . Then there exist α and β , admissible for T and U , satisfying (5), and associated with some state transition sequences

$$(6) \qquad \begin{array}{l} t_0^{(o)} \rightarrow \ . \ . \ . \rightarrow t_0^{(i_o)} \rightarrow \\ \quad . \ . \ . \ . \ . \ . \ . \ . \\ \rightarrow t_p^{(o)} \rightarrow \ldots \rightarrow t_p^{(i_p)} \quad , \end{array}$$

$$(7) \qquad \begin{array}{l} u_0^{(o)} \rightarrow \ . \ . \ . \rightarrow u_0^{(j_o)} \\ \quad . \ . \ . \ . \ . \ . \ . \ . \ . \\ \rightarrow u_p^{(o)} \rightarrow \ldots \rightarrow u_p^{(j_p)} \quad , \end{array}$$

where $t_0^{(o)}$ and $u_0^{(o)}$ are initial states, $t_p^{(i_p)}$ and $u_p^{(j_p)}$ are final states, p , $i_0, \ldots, i_p, j_0, \ldots, j_p \geq 0$, and output (input) actions are associated with exactly those transitions where the subscript of the state is changed. According to the construction of V , (6) and (7) can be "merged" into a transition sequence from an initial state to a final state of V :

$$(t_0^{(o)}, u_0^{(o)}) \to \ldots \to (t_0^{(i_o)}, u_0^{(j_o)}) \to$$

$$\text{(8)} \qquad \cdots\cdots\cdots\cdots$$

$$\to (t_p^{(o)}, u_p^{(o)}) \to \ldots \to (t_p^{(i_p)}, u_p^{(j_p)}) .$$

The action sequence γ' associated with (8) is a merge of α_{A_1} and β_{A_2}. Therefore,

$$\gamma'_{A_1} = \alpha_{A_1} = \gamma_{A_1} ,$$

$$\gamma'_{A_2} = \beta_{A_2} = \gamma_{A_2} ,$$

which means that γ' is admissible and contains the same (external) input and/or output as γ .

Conversely, any action sequence γ , admissible on V , corresponds to a state transition sequence of the form (8). According to the construction of V , this determines sequences (6) and (7) for T and U, with action sequences α and β satisfying (5). Therefore, y is also admissible for the serial combination of T and U . □

We shall need this theorem only for the special case that one of the automata is a finite automaton. For this special case we have:

<u>Corollary 1</u>. For any class of automata determined by their peripheral devices (in addition to a single input and/or output), the family of languages associated with the class is closed under finite transduction.

Finite transduction is in itself a very powerful operation. As its special cases we have, for instance, projection, insertion, intersection with regular languages, and quotients by regular languages defined as

left quotient of L by R $= R\backslash L = \{x \mid yx \in L$ for some $y \in R\}$,

right quotient of L by R $= L/R = \{x \mid xy \in L$ for some $y \in R\}$.

Notice also that $\text{init}(L) = L/Z^*$.

In particular, we have for finite automata:

Corollary 2. Finite transduction of a regular language is regular.

Knowing that the family of regular languages is closed under insertion, intersection, and projection, one could also see this of (4), which for finite transducers assumes the form

(9) $$(L^{\oplus} \cap B_{fts})_A \quad .$$

6. Context-free Languages and Pushdown Automata

Knowing that the family of context-free languages is closed under insertion, intersection with regular languages, and projection, (9) gives :

Theorem 5. Finite transduction of a context-free language is context-free.

As a special application of this result we notice that B_{stack} of Section 4.3 will remain context-free even if the initial and final stack contents are only required to belong to some regular languages over Σ_{stack} . In particular, one could allow the final contents to be any element of Σ_{stack}^* .

For a pushdown transducer, (4) takes the form

$$(L^{\oplus} \cap B^{\oplus}_{\text{stack}} \cap B_{\text{fts}})A_{\text{output}} \quad . \tag{10}$$

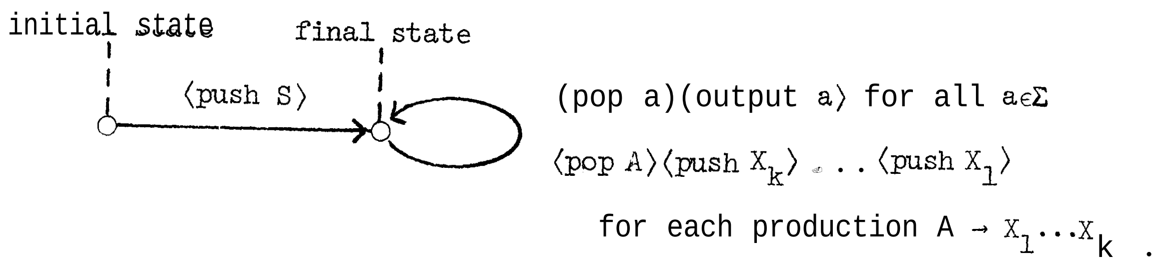
If L is regular, then all operations in (10) preserve the context-free property of B_{stack} , and we have:

<u>Theorem 6</u>. Pushdown transduction of a regular language is context-free.

Next, we shall show that the family of languages associated with pushdown automata is the family of context-free languages:

<u>Theorem 7</u>. The family of languages associated with pushdown automata is the family of context-free languages.

<u>Proof</u>. For a pushdown automaton, B of (1) is context-free. Therefore the language accepted (2) or generated (3) is also context-free. For the second part of the theorem the following simple construction is sufficient:


initial state
final state
⟨push S⟩
(pop a)(output a⟩ for all $a \in \Sigma$
⟨pop A⟩⟨push X_k⟩ . . . ⟨push X_1⟩
for each production $A \rightarrow X_1 \dots X_k$.

Here Σ_{stack} consists of all terminal (Σ) and non-terminal symbols of the context-free grammar. The operation of the automaton corresponds to following a leftmost derivation sequence of a word. Initially, S is pushed into the stack, and each time there is a nonterminal on top of the stack it is replaced 'by the right-hand side of a production for it (in reverse order). When a terminal symbol is encountered in the stack, it is removed for output. An admissible sequence of actions ends up with an empty stack, which corresponds to having completed a derivation sequence. □

Notice that the action (pop Λ) was not needed in the above construction. So this action does not add anything to the recognition or generative power of pushdown automata.

As the proof of the first part of Theorem 7 was only based on the context-free property of B_{stack} , it remains valid even if the initial and final requirements for stack contents are relaxed to arbitrary regular languages over Σ_{stack} . Corresponding changes are also easy to make in the construction for the second part of the theorem, so that any given regular languages could be used as initial and final stack contents. (In particular, the final contents could be allowed to be any member of Σ^*_{stack} .)

As Es-tack can always be encoded in terms of a fixed alphabet containing at least two letters, Theorem 7 shows that any context-free language over Σ can be represented in terms of a fixed context-free language and a regular language:

Theorem 8. Given an alphabet E , there is a fixed context-free language L_o over $\Sigma \cup \Sigma'$ (where Σ' is an auxiliary alphabet) such that any context-free language L over E can be expressed as

$$L = (L_o \cap R)_{\Sigma} ,$$

where R is some regular language over E $\cup \Sigma'$.

Proof. Consider generating arbitrary context-free languages L by pushdown automata with a fixed stack alphabet. Select $B^{\oplus}_{stack}$ as L_0, and take B_{fts} as R . $\square$

Since only two letters are required in Σ_{stack} , four letters (corresponding to push and pop operations for the two stack letters) are sufficient in Σ' .

Using the construction above to generate an arbitrary context-free language L , one could also proceed as follows. Encode only nonterminals as stack letters by using two auxiliary stack letters, and delete all Output actions. Then we have[1/]

$$L = (L_o \cap R)_{\{\langle pop\ a\rangle]} .$$

This shows that L_o in Theorem 8 can be chosen as a Dyck language over a 4-letter alphabet, and that $|\Sigma| + 4$ letters are then sufficient in Σ' .

As intersection with a regular language, decoding of alphabet, and projection (together with a possible change of alphabet) can all be performed by finite transducers, Theorem 4 gives us:

Theorem 9. Every context-free language is a finite transduction of a fixed Dyck language over a four-letter alphabet.

[1/] Instead of letters in E , L here has corresponding letters in $\{\langle pop\ a\rangle \mid a \in \Sigma\}$.

7. Turing Machines and Undecidable Questions on Languages

The same reasoning as was used in the previous section can be applied also to other classes of automata. As any recursively enumerable language can be generated by a Turing machine, we get the following theorem:

Theorem 10. Given an alphabet E , there are two fixed context-free languages L_1 and L_2 over $\Sigma \cup \Sigma'$ (where Σ' is an auxiliary alphabet) such that any recursively enumerable language L over E can be expressed as

$$L = (L_1 \cap L_2 \cap R)_\Sigma$$

where R is some regular language over $\Sigma \cup \Sigma'$.

Proof. Consider generating arbitrary recursively enumerable languages L by Turing machines with a fixed tape alphabet. Select $\text{init}(L_1)^\oplus$ and $\text{init}(L_2)^\oplus$ of Section 4.5 as L_1 and L_2 , and take B_{fts} as R . □

Since only two letters are required to encode any actions of A_{tape} , two letters are sufficient in Σ' .

As intersection with a context-free language can be implemented by pushdown transduction, and intersection with a regular language and projection (with a possible change of alphabet) can be performed by finite transducers, Theorem 4 gives us:

Theorem 11. Every recursively enumerable language is a pushdown transduction of a fixed context-free language.

Considering an automaton with two pushdown stacks, instead of a tape, one notices that the fixed context-free language in Theorem 11 can be chosen as a Dyck language.

According to our definitions, B of (1) is empty iff the operation of the automaton cannot terminate successfully. Our approach then ties several questions about formal languages directly to the halting problems of automata. For instance, as B for a Turing machine is always an intersection of two context-free languages (which can be effectively constructed from a description of the Turing machine), the emptiness problem for intersection of con-text-free languages must be undecidable. Undecidability questions will not be treated in more detail here, as the proofs by Hartmanis [6] are directly applicable to our approach as well.

8. Relation to Abstract Families of Languages

The approach pursued above can be generalized by thinking of an arbitrary language B_i as being associated with some abstract peripheral device. On account of (1) it is sufficient to consider automata with only one peripheral device (in addition to a single input and/or output). In fact, any two devices with languages $B_1 \subset A_1^*$, B_2 c A; , where $A_1 \cap A_2 = \emptyset$, can be replaced by a single device with the language $(B_1)^{A_2} \cap (B_2)^{A_1}$.

Let us now rewrite (2) and (3) as

$$(11) \qquad L = (L_o^{\Sigma} \cap R)_{\Sigma}$$

where L_o is the "peripheral language" $L_o \subset \Sigma_p^*$, and R is the regular language $R \subset (\Sigma \cup \Sigma_p)^*$ associated with the state transition system. The "peripheral alphabet" Σ_p is arbitrary, yet disjoint from Σ, which is the input (output) alphabet. Notice that an arbitrary alphabet Σ

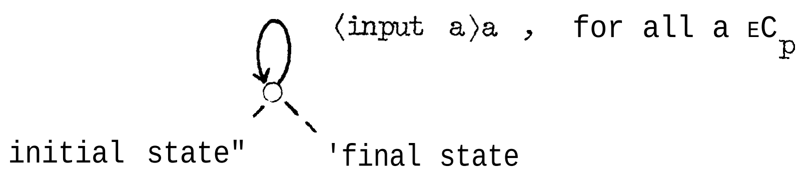
can be dealt with by renaming its elements and letting the projection notation in (11) denote the combination of projection and reversal of this renaming.

It is natural to associate a class of automata with each "peripheral language" L_o . Let us call such classes fap-classes (for finite action peripherals):

Definition 4. The set of automata with the same peripheral language L_o is called the fap-class generated by L_o .

Definition 5. A family of languages associated with a fap-class (generated by L_o) is called a fap-family (generated by L_o).

Obviously, a fap-family consists of the languages (11) for arbitrary R . In particular, it contains its generator L_o , which is accepted by the following automaton:



Corollary 1 to Theorem 4 says that fap-families are closed under finite transduction. On the other hand, all operations involved in (11) are special cases of finite transduction. Therefore, we can conclude:

Theorem 12. A family of languages is a fap-family iff it is generated by a single language by finite transduction.

Let L_1 and L_2 be two arbitrary languages. Without affecting generality we can assume that their alphabets have been made disjoint by renaming. The fap-family generated by $L_1 \cup L_2$ does then contain both L_1 and L_2 . Intuitively, two devices are then combined into one so that any of the component devices, but only one of them, can be used. On the other hand, it is straightforward to verify that each fap-family is closed under union. Therefore, this fap-family is the family generated by L_1 and L_2 by finite transduction, and Theorem 12 gets the stronger form:

Theorem 12b. -- A family of languages is a fap-family iff it is generated by a finite set of languages by finite transduction.

It can be easily verified that fap-families need not be closed under concatenation. Intuitively this is associated with the fact that it may be impossible to reset a peripheral device from a final status to an initial status. The intuitive notion of resetting peripheral devices by action sequences $R_{L_o} \subset \Sigma_p^*$ can be formulated as follows:

- For each action sequence $x \in L_o$ there is a resetting sequence $r \in R_{L_o}$ such that $xry \in L_o$ for all $y \in L_o$.
- If an admissible action sequence $z \in L_o$ contains a resetting sequence $r \in R_{L_o}$, i.e., $z = xry$ for some $x,y \in \Sigma_p^*$, then the status of the device is an admissible final (initial) status after the actions in x (xr) , i.e., $x,y \in L_o$.

If $\text{Replace}_{R,c}$ denotes a transduction which transduces any word xry , where $r \in R$, into xcy , then the result of these intuitive considerations can be stated in terms of the following definition:

Definition 6. A language L is called resettable by R_L, if

$$\text{Replace}_{R_L,c}(L) = LcL ,$$

where c is a symbol not contained in L .

As an example, the languages B_{stack} and B_{queue} of Section 4 are reset-table by $R_{stack} = \{\langle pop\ \Lambda\rangle\}$, $R_{queue} = [(read\ \Lambda\rangle\}$. If the requirements for the final contents were totally removed, no finite resetting languages would suffice, and we would get

$$R'_{stack} = \langle pop\ a \mid a \in \Sigma_{stack}\rangle^{*}\langle pop\ \Lambda\rangle ,$$

$$R'_{queue} = (read\ a \mid a \in \Sigma_{queue}\rangle^{*}\langle read\ A)$$

For fap-classes with resettable devices we can now show:

Theorem 13. A fap-family of languages is closed under concatenation and plus[1] ($^{+}$) iff it is associated with a fap-class where peripheral devices are resettable by regular languages.

Proof. If there are more than one peripheral device, then the concatenation of their resetting languages would be a resetting language for the single device obtained of them. It is therefore sufficient to consider only the case of one device.

Let us start with the 'if'-part. Let T and U be two automata of the fap-class generated by a language L_0 reset-table by a regular language R . Being regular, R is associated with a finite transition

[1] Plus is defined as $L^{+} = LL^{*}$. In fact, we could have star (*) in this theorem equally well, but the later analogue for restricted fap-families is valid only for plus ($^{+}$) .

system where edges are labelled by elements of Σ_p^* . A third automaton V of the same fap-class can now be constructed as follows. Connect the final states of T by E-transitions to the initial states of the transition system for R , and connect the final state of this system again by s-transitions to the initial states of U . Initial states of T (final states of U) are taken as initial (final) states of V . The language associated with V is now the concatenation of the languages associated with T and U .

In fact, let x and y be some admissible action sequences for T and U . In particular, $x_{\Sigma_p}, y_{\Sigma_p} \in L_o$. On account of the definition of R , there is now a resetting sequence $r \in R$ satisfying $x_{\Sigma_p} r y_{\Sigma_p} \in L_o$. As x , y , and r are associated with admissible transition sequences in T , U , and the transition system for R , there is an admissible transition sequence in V , associated with the action sequence z = xry . Since $z_{\Sigma_p} = x_{\Sigma_p} r y_{\Sigma_p} \in L_0$, and $z_\Sigma = (xy)_\Sigma$, V accepts each concatenation of words accepted by T and U .

Conversely, let z be an admissible action sequence of V . Because of the construction of V , z must have the form z = xry , $r \in R$, where x and Y are associated with some admissible transition sequences of T and U . The admissibility of z also means that $z_{\Sigma_p} \in L_o$. As $z_{\Sigma_p} = x_{\Sigma_p} r y_{\Sigma_p}$, we have $x_{\Sigma_p}, y_{\Sigma_p} \in L_o$ on account of the definition of R . This shows that x and y are, indeed, admissible for T and U . Hence, $z_\Sigma = x_\Sigma y_\Sigma$ is a concatenation of words accepted by T and U .

For plus operation, an automaton W can be constructed of a given automaton T as follows. Connect the final states of T by e-transitions to the initial states of the transition system for R , and connect the final states of this system by E-transitions to the initial states of T . As initial and final states take those of T . The correctness of this construction can be proved similarly to the above proof for concatenation. For brevity, details are omitted here.

For the 'only if'-part it suffices to notice that any device can be made resettable by adding a new primitive action for this purpose. This means that the generator L_o can be replaced by another generator $L'_o = L_o \cup L_o(cL_o)^+$, where c is a new symbol outside the original Σ_p and Σ . This language obviously has a resetting language $\{c\}$, and, if the fap-family generated by L_o is closed under concatenation and plus, then L'_o generates the same fap-family. ❑

Theorem 4 showed that characterizing peripheral devices locally leads to language families closed under arbitrary finite transductions. A simple and common way of introducing non-local restrictions is the following: let us associate an integer $k \geq 0$ with each automaton, and let us accept an action sequence $x \in L_o^{\Sigma} \cap R$ as admissible iff $x \notin \Sigma_p^+$, and $x \notin (\Sigma \cup \Sigma_p)^* \Sigma_p^{k+1} (\Sigma \cup \Sigma_p)^*$. In other words, a non-empty action sequence is required to contain at least one action for input (if accepting) or output (if generating or transducing), and it cannot contain more than k consecutive actions of other kinds.

More formally, let R_k , $k \geq 0$, be defined as the complement of

$$R_k = \Sigma_p^+ \cup (\Sigma \cup \Sigma_p)^* \Sigma_p^{k+1} (\Sigma \cup \Sigma_p)^* \quad .$$

A k-restricted automaton with peripheral language L_o and transition system language R is then associated with the language

$$L = (L_o^{\Sigma} \cap R_k \cap R)_{\Sigma} \quad .$$

As $R_k \cap R$ can always be associated with a transition system, we can also use (11) with the additional assumption that the transition system is k-restricted, i.e., $R \subset R_k$.

An automaton will be called restricted, if it is k-restricted for some $k \geq 0$. As an example, let us consider restricted automata with an arbitrary finite number of pushdown stacks, queues and tapes. Obviously, L_O of (11) for any such automaton is context-sensitive. It can be shown [8] that a projection, not erasing more than k consecutive characters for some $k \geq 0$, preserves the context-sensitive property of a language. Therefore, we notice that languages associated with these automata are context-sensitive. Similarly, we can see of (4) that these automata transduce context-sensitive languages into context-sensitive languages.

Analogously to Definitions 4 and 5 we now define:

Definition 7. The set of restricted automata with the same peripheral language is called a restricted fap-class. A family of languages associated with a restricted fap-class is called a restricted fap-family.

It can now be easily verified that the construction used in the proof of Theorem 4 produces a restricted automaton V , if both T and U are restricted. Also, for restricted automata, all operations involved in (11) are special cases of restricted finite transductions. Therefore, an analogue of Theorem 12b holds for restricted fap-families and restricted finite transduction.

Similarly, an analogue of Theorem 13 is obtained for restricted fap-families associated with automata with peripherals resettable by finite languages. (Some care must be exercised in the constructions in the proof, in order to deal properly with languages containing ε.)

Abstract families of languages, or AFL, can be characterized [4] as those families which are closed under union, concatenation, plus ($^+$), and restricted finite transduction. An AFL is a full. AFL if it is closed under an arbitrary finite transduction, and it is a principal AFL if it is generated by a single language by the operations listed above.

In terms of these notions we can now conclude the discussion with the following theorem, already obvious from what has been proved:

Theorem 14. A family of languages is a (restricted) fap-family associated with automata with peripherals resettable by regular (finite) languages, iff it is a full (not necessarily full) principal AFL.

9. Conclusions

The approach presented is based on an inherently nondeterministic conception of automata. An interesting way to introduce determinism would be to take an auxiliary input for controlling nondeterminism.

Another limitation was dealing with only one-way input. Obviously, two-way input could, in principle, be handled similarly to other devices. However, much of the elegance of this approach seems to be lost with the complications in the language involved. For instance, it is not obvious from this language that each two-way finite automaton accepts a regular language.

As a unified approach to automata, this approach has the advantage that one does not need to be concerned with information representation in the infinite storage. Also, very clear intuition is provided for principal AFL, and their homomorphic characterization theorem. Obviously, more general classes of automata than fap-classes are needed for non-principal AFL.

In conclusion, it is felt that the approach presented deserves attention in courses and textbooks on automata and formal languages, and it is hoped that this paper can serve in making it more widely known.

References

[1] Floyd, Robert W., Private communication.

[2] Ginsburg, Seymour, The Mathematical Theory of Context-free Languages. McGraw-Hill, 1966.

[3] Ginsburg, Seymour and Sheila Greibach, 'Principal AFL," J. Comput. System Sci. 4, 1970, 308-338.

[4] Ginsburg, Seymour, Sheila Greibach, and John Hopcroft, "Studies in Abstract Families of Languages," Memoirs of the Amer. Math. Soc., 87, 1969.

[5] Ginsburg, Seymour, and Gene F. Rose, 'Preservation of Languages by Transducers," Information and Control 9, 1966, 153-176.
See also: "A Note on Preservation of Languages by Transducers," Information and Control 12, 1968, 549-552.

[6] Hartmanis, J., "Context-free Languages and Turing Machine Computations," Proceedings of Symposia in Applied Mathematics 19, American Mathematical Society, 1967.

[7] Hopcroft, J. E. and J. D. Ullman, "An Approach to a Unified Theory of Automata," Bell System Tech. J. 46, 1967, 1793-1829.

[8] Hopcroft, John E., and J. D. Ullman, Formal Languages and Their Relation to Automata. Addison-Wesley, 1969.

[9] Salomaa, Arto, Formal Languages. Academic Press, 1973.