CURRENT · VERSION

VERY SPECIAL LANGUAGES AND REPRESENTATIONS OF RECURSIVELY ENUMERABLE LANGUAGES VIA COMPUTATION HISTORIES

by

David Haussler

and

H. Paul Zeiger
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado

CU-CS-177-80 April, 1980

This work supported in part by NSF grant MCS79-08338

1. Running head is "Very Special Languages."

2. Article length is 22 pages, no tables, no figures.

3. Special symbols used are:

$$\Phi,\ \Sigma,\ \Gamma,\ \Delta,\ \delta,\ \lambda,\ \varepsilon,\ =,\ \subset,\ \subseteq,\ \supset,\ \supseteq,\ \cap,\ \cup,\ \rightarrow,\ *,\ ||,\ L,\ P.$$

Also used as subscript or superscript are: Σ, Δ, Γ, $*$. Letters and subscripts may appear overbarred as in $\overline{a}$ or underbarred as in $\underline{a}$ or both as in $\underline{\overline{a}}$.

ABSTRACT

A method of encoding the computation histories of a wide class of machines is introduced and used to derive several representation theorems for the class of recursively enumerable languages. In particular it is demonstrated that any recursively enumerable language $K \subset \Sigma^*$ can be represented as $K = \Phi_\Sigma(R \cap D_1 \| D_2)$ where D_1 and D_2 are fixed semi-Dyck languages, $\|$ is the shuffle operation, R is a regular language depending on K and Φ_Σ is a weak identity homomorphism. This result is the natural analog for the recursively enumerable languages of the Chomsky-Shutzenberger representation of the context free languages.

I INTRODUCTION

Recently, [ENG,ROZ 79], an elegant representation of the recursively enumerable languages over a finite alphabet has been discovered which is analogous to the Chomsky-Shutzenberger representation of the context-free languages. One fixed language of remarkable simplicity, the complete twin shuffle, is presented, which through intersection with regular sets followed by a weak identity homomorphism, generates all the recursively enumerable languages over a fixed alphabet. A. Ehrenfeucht has proposed that a language capable of representing the recursively enumerable languages in the above sense be called a very special language or VSL. Our major result is the discovery of another very special language which is the shuffle of two semi-Dyck languages. This result yields the natural analog for the recursively enumerable languages of the Chomsky-Shutzenberger theorem.

The technique used in proving this and other results in this paper involves a method of representing histories of read and write actions of a finite automaton on a set of potentially infinite data structures (i.e., queues, stacks, etc.) as words in a regular language. Then either the application of a fixed mapping or an intersection with a fixed "checking" language serves to sort out the words representing valid computation histories from those in which the data structures were accessed incorrectly. This is in essence the technique used in [GIN 70] and [GIN 73]. One of the authors learned it from Robert Floyd in 1975. The power of this technique is demonstrated in the derivations of several representation theorems, some of which are new, while others have been previously derived by different methods.

II DEFINITIONS

Here we give formal definitions of the terminology used throughout this paper.

DEFINITION: Given a machine A, $L(A)$ denotes the language accepted by A.

DEFINITION: Given two finite alphabets Δ and Σ such that $\Delta \supset \Sigma$, the weak identity $\Phi_\Sigma : \Delta^* \to \Sigma^*$ is defined by

$$\Phi_\Sigma(a) = a \quad \text{if } a \in \Sigma$$

$$\Phi_\Sigma(a) = \lambda \quad \text{if } a \notin \Sigma.$$

DEFINITION: Given a finite alphabet Σ, we denote the semi-Dyck language generated by

$\{S \to a S \bar{a} S \mid \lambda : a \in \Sigma\}$ as D_Σ.

DEFINITION: Given two languages L and M, contained in Σ^*, we denote the shuffle of L and M as $L||M$ where

$$L||M = \{w_1 u_1 \ldots w_n u_n : w_i, u_i \in \Sigma^* \text{ for } 1 \le i \le n,\ w_1 \ldots w_n \in L \text{ and } u_1 \ldots u_n \in M\}.$$

DEFINITION: The complete twin shuffle over the alphabet Σ, denoted L_Σ, is defined by

$$L_\Sigma = \{w_1\bar{u}_1 \ldots w_n\bar{u}_n : w_i, u_i \in \Sigma^* \text{ for } 1 \le i \le n, \text{ and } w_1 \ldots w_n = u_1 \ldots u_n\}.$$

DEFINITION: The bar right twin shuffle over the alphabet Σ, denoted BRL_Σ is defined by

$$BRL_\Sigma = \{w_1 \bar{u}_1 \ldots w_n \bar{u}_n : w_i, u_i \in \Sigma^* \text{ for } 1 \le i \le n,\ w_1 \ldots w_n = u_1 \ldots u_n \text{ and } \forall i : 1 \le i < n, u_1 \ldots u_i \text{ is a prefix of } w_1 \ldots w_i\}.$$

<u>DEFINITION</u>: The class of recursively enumerable languages over the finite alphabet Σ will be denoted RE_Σ.

<u>DEFINITION</u>: Given a word $w \in \Sigma^*$, w^R will denote the reversal of w.

<u>DEFINITION</u>: Given a word $w \in (\Sigma \cup \overline{\Sigma})^*$, w^c will denote the complement of w, i.e., w^c is obtained from w by the mapping $c(a) = \overline{a}$, $c(\overline{a}) = a$ for $a \in \Sigma$.

Finally our fundamental definition:

<u>DEFINITION</u>: Given two finite alphabets Δ and Σ such that $\Delta \supset \Sigma$, a language $L \subset \Delta^*$ is a very special language for Σ, abbreviated $L \in VSL_\Sigma$, iff for any recursively enumerable language $K \subset \Sigma^*$ there exists a regular language $R \subset \Delta^*$ such that

$$K = \Phi_\Sigma(R \cap L).$$

III MAIN RESULTS

Our first theorem will be our natural analog for the recursively enumerable languages of the Chomsky-Shutzenberger theorem.

THEOREM I: For any finite alphabet Σ, we can define two semi-Dyck languages D_1 and D_2 on disjoint alphabets Δ_1 and Δ_2 such that $\Sigma \subset \Delta_1$ and for any recursively enumerable lanquage $K \subset \Sigma^*$ there exists a regular language $R \subset (\Delta_1 \cup \Delta_2)^*$ such that

$$K = \Phi_\Sigma(R \cap D_1 || D_2)$$

i.e., $D_1 || D_2 \in VSL_\Sigma$.

PROOF: Let us represent the class RE_Σ as the class of languages accepted by finite automata with one way read only input and two push down stores P_1 and P_2, each with read-write alphabet Γ distinct from Σ (see [HOP,ULL 79]). Using this representation (with some additional stipulations on the machines to be mentioned later), we will show that $D_{\Sigma \cup \Gamma} || D_{\underline{\Gamma}} \in VSL_\Sigma$.

Our first step is to define a scheme for encoding the computation histories of these machines as words over the alphabet $\Delta \cup \overline{\Delta}$ where $\Delta = \Sigma \cup \Gamma \cup \underline{\Gamma}$. For reasons which will become apparent later, we will choose the following encoding:

for $a \in \Gamma$	a	will encode "push a on P_1"
	$\overline{a}$	will encode "pop a from P_1"
	$\underline{a}$	will encode "push a on P_2
	$\overline{\underline{a}}$	will encode "pop a from P_2"

and for $a \in \Sigma$ a will encode "read a from the read only input"

$\bar{a}$ will have no significance in terms of computation. It will only be used to maintain the semi-Dyck character of the computation histories.

Instead of having special encodings of "test for empty stack," we will assume that one of the letters of Γ is used exclusively as a bottom of stack marker.

Using this encoding, we then assume that an automaton A of the type we are considering is given as a sextuple $<Q, \Sigma, \Gamma, \delta, q_0, F>$ where

Q is the set of states,

Σ and Γ are as given above,

q_0 is the initial state,

F is the set of final states and

δ, the transition function, is a mapping

$\delta : (\Sigma \cup \bar{\Gamma} \cup \underline{\Gamma}) \times Q \to \mathcal{P}((\bar{\Sigma} \cup \Gamma \cup \underline{\Gamma}) \times Q)$.

Thus, given a letter from the alphabet of read actions and a state, δ defines a set of pairs, each consisting of a string of write actions and a next state.

Now, given any automaton A represented in the above manner, we associate a regular language H_A defined by the right linear grammar $<Q, \Delta \cup \bar{\Delta}, q_0, P>$ where

$q_i \to a w q_j \in P$ iff $<w, q_j> \in \delta(<a, q_i>)$

$q_i \to a w \in P$ iff $<w, q_j> \in \delta(<a, q_i>)$ and $q_j \in F$.

The language H_A represents all sequences of actions taking the finite control from its initial state to a final state. However, most of these sequences are not valid computation histories because nothing in the definition of H_A insures us that a symbol popped from a stack at a particular point in the sequence of actions is actually the symbol that should be at the top of the stack at this point, in view of the preceding sequence of actions. We will use the semi-Dyck languages $D_{\Sigma \cup \Gamma}$ and $D_{\underline{\Gamma}}$ to sort out those computation histories of H_A in which the stacks are handled correctly from those in which they are not. To implement this strategy, let us stipulate that:

1. The automaton begins by pushing the bottom of stack marker on each stack.

2. Every time a letter $a \in \Sigma$ is read from the read only input, the next write action is $\overline{a}$ and in no other case is an action from $\overline{\Sigma}$ performed.

3. Before accepting, the automaton empties both stacks completely, including their bottom of stack markers. Thus, the automaton accepts by simultaneously entering into a final state and emptying both stacks.

With these stipulations, it is apparent that any $w \in H_A$ for which $\Phi_{\Gamma \cup \overline{\Gamma}}(w) \in D_\Gamma$ and $\Phi_{\underline{\Gamma} \cup \underline{\overline{\Gamma}}}(w) \in D_{\underline{\Gamma}}$ will be a valid computation history for some word $v = \Phi_\Sigma(w)$. Further, since the pairs corresponding to read-only input actions are all simple, unnested strings of the form $a\overline{a}$ for $a \in \Sigma$, these pairs may be taken as embedded in one of the semi-Dyck languages D_Γ or $D_{\underline{\Gamma}}$. Thus w will be a valid computation history if

$$w \in H_A \cap (D_{\Sigma \cup \Gamma} \| D_{\underline{\Gamma}}).$$

On the other hand, by our stipulations, every valid computation history is in this set. Thus,

$$L(A) = \Phi_\Sigma (H_A \cap D_{\Sigma \cup \Gamma} \| D_{\underline{\Gamma}})$$

where L(A) is the language accepted by A. This shows that

$D_{\Sigma \cup \Gamma} \| D_{\underline{\Gamma}} \in VSL_\Sigma$ as desired. □

As the reader has undoubtedly already noticed, not only is this result a direct analog of the Chomsky-Shutzenberger theorem, but the proof itself can easily be adapted to a proof of the Chomsky result by allowing the automaton A only one stack.

Before continuing on to our next theorem, let us mention a few corollaries of Theorem I. The following is a fairly well known result which already has two very different derivations (see [SAL 73] and [FIS,RAN 69]).

Corollary I: For any finite alphabet Σ we can define a finite alphabet $\Delta \supset \Sigma$ and two fixed deterministic context free languages (DCFL's) L_1 and L_2 such that for any recursively enumerable set $K \subset \Sigma^*$ there exists a regular language $R \subset \Delta^*$ such that

$$K = \Phi_\Sigma(R \cap L_1 \cap L_2).$$

Proof: Given the semi-Dyck languages $D_1 \subset \Delta_1^*$ and $D_2 \subset \Delta_2^*$ where $\Delta_1 \cap \Delta_2 = \emptyset$ asserted to exist by Theorem I, let

$$L_1 = \Phi_{\Delta_1}^{-1} (D_1) \text{ and } L_2 = \Phi_{\Delta_2}^{-1} (D_2)$$

where $\Phi_{\Delta_1} : \Delta_1 \cup \Delta_2 \to \Delta_1$ and $\Phi_{\Delta_2} : \Delta_1 \cup \Delta_2 \to \Delta_2$. Then both L_1 and L_2 are DCFL's and

$$D_1 \| D_2 = L_1 \cap L_2.$$ □

Using a technique developed by G. Rozenberg [EHR,ROZ 79a], [EHR,ROZ 79b] it is not hard to show that the languages L_1 and L_2 of Corollary I may be replaced by 2 fixed DOS languages at the cost of further enlarging the alaphabets Δ_1 and Δ_2. This yields an analogous representation theorem for RE_Σ in terms of the intersection of two fixed DOS languages.

For the purposes of our next theorem, let us introduce the following definition:

Definition: Given alphabets Δ, Σ_1 and Σ_2 such that $\Delta \supset \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$, the mapping $CANCEL_{\Sigma_1,\Sigma_2} : (\Delta \cup \overline{\Delta})^* \to (\Delta \cup \overline{\Delta})^*$ is the mapping that results from exhaustive application of the rule

$$u\,a\,v\,\overline{a}\,w \to u\,v\,w$$

where either $\left\{ \begin{array}{ll} a \in \Sigma_1 & \text{and } v \in ((\Delta \cup \overline{\Delta}) - (\Sigma_1 \cup \overline{\Sigma}_1))^* \\ \text{or} & \\ a \in \Sigma_2 & \text{and } v \in ((\Delta \cup \overline{\Delta}) - (\Sigma_2 \cup \overline{\Sigma}_2))^* \end{array} \right\}$

Theorem II: For any finite alphabet Σ we can define a finite alphabet Γ disjoint from Σ such that for any recursively enumerable language $K \subset \Sigma^*$ there exists a regular language $R \subset \Sigma \cup \Gamma \cup \overline{\Gamma} \cup \underline{\Gamma} \cup \overline{\underline{\Gamma}}$ such that $K = CANCEL_{\Gamma,\underline{\Gamma}}(R) \cap \Sigma^*$.

Proof: We use the same automata and encoding scheme used in Theorem I except that we do not follow letters $a \in \Sigma$ with $\overline{a}$. A and H_A are defined accordingly and it is easy to see that $w \in H_A$ is a valid computation iff $CANCEL_{\Gamma,\underline{\Gamma}}(w) \subset \Sigma^*$, i.e., iff all stack letters are canceled. From this it follows that $L(A) = CANCEL_{\Gamma,\underline{\Gamma}}(H_A) \cap \Sigma^*$, which yields our representation. □

One of the classical results in computability theory is that the finite automata with one potentially infinite queue are equivalent to the Turing machines in computational power [SHE,STU 63]. In our next theorem we will apply the techniques used in Theorem I to this class of machines and introduce a different very special language, called the bar right twin shuffle, which characterizes the valid computation histories on a queue.

Definition: BRL_Σ, the bar right twin shuffle over the alphabet Σ, is defined as

$$BRL_\Sigma = \{w_1 \bar{v}_1 \ldots w_n \bar{v}_n : w_i, u_i \in \Sigma^* \text{ for } 1 \le i \le n,\ w_1 \ldots w_n = v_1 \ldots v_n$$
$$\text{and } \forall i : 1 \le i < n, v_1 \ldots v_i \text{ is a prefix of } w_1 \ldots w_i\}$$

Theorem III: Given a finite alphabet Σ, we can define a finite alphabet $\Delta \supset \Sigma$ such that for any recursively enumerable language $K \subset \Sigma^*$ there exists a regular language $R \subset (\Delta \cup \bar{\Delta})^*$ such that $K = \Phi_\Sigma(R \cap BRL_\Delta)$, i.e., $BRL_\Delta \in VSL_\Sigma$.

Proof: We will assume K is given as the language accepted by a finite automaton A with a potentially infinite queue with read alphabet $\Delta = \Sigma \cup \Gamma$ and write alphabet Γ where $\Sigma \cap \Gamma = \emptyset$. Initially, a word $w \in \Sigma^*$ will appear on the queue to be tested. A will accept by simultaneously emptying the queue and entering into a final state. The actions of A are encoded as follows:

for $a \in \Delta$, $\bar{a}$ will encode "pop a off the front of the queue"

for $a \in \Gamma$, a will encode "write a on the end of the queue."

We will have no special action "test for empty queue" but rather assume that one symbol $p \in \Gamma$ is used exclusively as a place holder. Thus, initially p will be written on the end of the queue and every time p is popped from the front of the queue it will be replaced on the end of the queue until A enters a "final sequence" in which the queue is completely emptied prior to A's acceptance. An "empty" queue can then be detected as two successive pops of p from the front of the queue.

With this encoding A can be given as a sextuple $<Q, \Sigma, \Gamma, \delta, q_0, F>$ as in Theorem I, this time with $\delta : \overline{\Delta} \times Q \to P(\Gamma^* \times Q)$. The regular set H_A of sequences of actions leading from q_0 to an element of F is defined as in the proof of Theorem I as well. We will use the language BRL_Δ to check if a word $w \in H_A$ represents a valid computation history in which a word $v \in \Sigma^*$ is accepted. To this end we consider the language $\Sigma^* H_A$ formed by concatenating all possible input words with all possible computation sequences from H_A. A word $u \in \Sigma^* H_A$ is a valid computation history for its prefix $v \in \Sigma^*$ iff

1. each letter popped from the front of the queue (i.e., appearing barred) had previously been written or placed on the queue (i.e., appears previously unbarred) and

2. the letters are popped off the queue in exactly the same order that they were placed or written on the queue (i.e., the barred subword reads the same as the unbarred).

Thus u is a valid computation history iff $u \in \Sigma^* H_A \cap BRL_\Delta$. It follows that $L(A) = \Phi_\Sigma(\Sigma^* H_A \cap BRL_\Delta)$, which implies that $BRL_\Delta \in VSL_\Sigma$ as desired. □

Using the technique of this proof, we can also derive the result from [ENG,ROZ 79] that the complete twin shuffle is a very special language. The bar right twin shuffle is just a restricted version of the complete twin shuffle in which barred letters must always appear to the right of their corresponding unbarred symbols. This is, of course, a natural restriction when modelling computations on a queue, unless one wants to consider a sort of debtor's queue, in which nonexistant letters may be popped off with the proviso that they must be written back in the same order at some later time. Words in the complete twin shuffle may be looked upon as computation histories on such debtors queues but it should be noted that the structure of the complete twin shuffle dictates that the queue be emptied before the computation goes into debt to it. Since the convention of using a place holder p used in the proof of Theorem III prevents the queue from becoming empty until the end of the computation, we can just as well use L_Δ, the complete twin shuffle over Δ, in place of BRL_Δ with no fear of allowing spurious debtor's queue computation histories to enter into our representation. This yields an alternate proof that $L_\Delta \in VSL_\Sigma$, one which is quite different from the proof in [ENG,ROZ 79].

One of the remarkable corollaries of these representation theorems follows from the fact that each of the languages BRL_Δ and L_Δ is recognized by a one-way, two headed deterministic finite automaton (2DFA). To be specific, BRL_Δ is recognized by a non-crossing 2DFA, i.e., one in which one head must always remain to the right of the other, and L_Δ is recognized by a blind 2DFA, i.e., one in which the heads are allowed to cross, but coincidence of the heads is not detectable. (see [ROS A.L. 66], [YAO,RIV 78]).

Corollary II: For any recursively enumerable language $K \subset \Sigma^*$ there exists a language L_1 accepted by a non-crossing 2DFA and a language L_2 accepted by a blind 2DFA such that

$$K = \Phi_\Sigma(L_1) = \Phi_\Sigma(L_2).$$

Proof: Follows directly from the fact that BRL_Δ and L_Δ are in VSL_Σ since the class of languages accepted by 2DFA's is closed under intersection with regular languages. □

Our final theorem is a result analogous to Theorem II, this time considering a natural cancellation mapping on the language BRL_Δ.

Definition: Given an alphabet Σ, $KANCEL_\Sigma : (\Sigma \cup \overline{\Sigma})^* \rightarrow (\Sigma \cup \overline{\Sigma})^*$ is the mapping that results from exhaustive application of the rule $a\,v\,\overline{a}\,w \rightarrow v\,w$ where $a \in \Sigma$ and $v \in \Sigma^*$.

Theorem IV: Given a finite alphabet Σ we can define a finite alphabet $\Delta \supset \Sigma$ such that for any recursively enumerable language $K \subset \Sigma^*$ there exists a regular language $R \subset (\Delta \cup \overline{\Delta})^*$ such that

$$K = KANCEL_\Delta(R) \cap \Sigma^*.$$

Proof: Let A and H_A be as given in the proof of Theorem III. Consider a computation history $v\,x \in \Sigma^* H_A \cap BRL_\Delta$ in which a word v is accepted. If we apply $KANCEL_\Delta$ to the word $w = ((v\,x)^R)^C = (x^R)^C\,\overline{v}^R$, then $KANCEL_\Delta(w) = \lambda$ and the last $|v|$ steps of the cancellation process cancel letters from v with their barred images. Thus, when we apply $KANCEL_\Delta$ to the prefix $(x^R)^C$ of w which does not include the barred and reversed image of the initial input word v, the result is precisely the unbarred reversed image of v, i.e., $KANCEL_\Delta((x^R)^C) = v^R$. On the other hand, if we consider a word $y \in H_A$ such that $\forall\ u \in \Sigma^*, u\,y \notin BRL_\Delta$ then $KANCEL(y^R)^C$ contains some barred symbols, otherwise $(KANCEL(y^R)^C)^R\ y \in BRL_\Delta$, contrary to our assumption. Thus, we may use $KANCEL_\Delta$ to recover the language accepted by A in the following manner:

$$L(A)^R = \text{KANCEL}_\Delta\ ((H_A^R)^C) \cap \Sigma^* .$$

Since RE_Σ is closed under reversal, our result follows. □

A similar result holds for the mapping REDUCT, which characterizes cancellations on L_Δ, defined in [HAU 79]. There, it is demonstrated that despite the fact that any recursively enumerable language $K \subset \Sigma^*$ can be represented as $\text{REDUCT}(R) \cap \Sigma^*$ for some regular language R, it is decidable whether or not REDUCT(R) is finite for a regular language R. From this result it follows that it is decidable whether or not two DGSM mappings are equivalent on a regular set (see [BLA,HEA 79], [CUL,SAL 78]

It should also be noted that the mappings KANCEL and REDUCT can each be achieved by a one-way, two-headed deterministic finite state transducers similar to the recognizers of Corollary II. These results then give some indication of the power of such transducers. In particular, we have that the image of a regular set under such a transduction is not necessarily recursive. In fact, we get the following stronger result.

Corollary III: For any finite alphabet Σ we can define an alphabet $\Delta \supset \Sigma$ such that for any recursively enumerable language $K \subset \Sigma^*$ there exists a partial mapping M_1 induced by a non-crossing 2DFA transducer and a partial mapping M_2 induced by a blind 2DFA transducer such that

$$K = M_1(\Delta^*) = M_2(\Delta^*).$$

Proof: We modify the transducers discussed above so that they accept an input word w only if w is an element of the regular language R and if their output is entirely contained in Σ^*. The mapping $M_i(w)$ for $i \in \{1,2\}$ is then defined on w only if the transducer accepts. □

Added in Press

We have noted that Theorem III has several more important corollaries.

DEFINITION: A single reversal stack is a stack with the following additional access limitation: Once a letter has been popped from the stack, no further letters may be pushed onto the stack. A single reset queue is a queue such that once a letter is popped from the queue, no further letters can be written onto the back of the queue.

Corollary IV: The non deterministic finite automata with one read only input tape and two single reversal stacks (reset queues) which accept by final state and empty stack (queue) are equivalent in computing power to turing machines.

Proof: Given a recursively enumerable language $k \subset \Sigma^*$ represented as $K = \Phi_\Sigma(R \cap L_{\Sigma \cup \Gamma})$ we design a nondeterministic finite automaton which reads in a word w, nondeterministically expands it to a word v in $\Phi_\Sigma^{-1}(w)$, checks to see that $v \in R$ and stores the unbarred letters from v in one stack (queue) and the barred letters in the other. Then it simply checks to see if one stack (queue) is the barred image of the other by popping off letters. $\square$

DEFINITION: Given a finite alphabet Σ let

$$PAL_\Sigma = \{w w^R : w \in \Sigma^*\} \quad \text{and}$$

$$COPY_\Sigma = \{w w : w \in \Sigma^*\}.$$

Corollary V: $PAL_{\Sigma \cup \Gamma} \| PAL_{\overline{\Sigma} \cup \overline{\Gamma}}$ and $COPY_{\Sigma \cup \Gamma} \| COPY_{\overline{\Sigma} \cup \overline{\Gamma}}$ are both in VSL_Σ.

Proof: Using the machines described in the proof of Corollary IV, we represent a queue or stack access with the letter accessed, regardless of whether it was a pop or a push. Sequences of accesses permitted by the automaton then form a regular set which when intersected with $PAL_{\Sigma \cup \Gamma} \| PAL_{\overline{\Sigma} \cup \overline{\Gamma}}$ $(COPY_{\Sigma \cup \Gamma} \| COPY_{\overline{\Sigma} \cup \overline{\Gamma}})$ yields the set of all valid computation histories of the automaton using the single reversal stacks (reset queues). The set accepted is then recovered by applying Φ_{Σ}. □

NOTE: Part of Corollary IV was proved using other methods in [BAK,BOO 74] and the other part was announced in [BOO,GRE,WRA 78]. [BAK,BOO74], [GIN,GRE 70], [GIN,GRE 73], [BRA 79] have considered many of the very special languages we have presented and demonstrated them to be full semi-AFL generators of the recursively enumerable languages.

IV CONCLUSION

A powerful method of obtaining representation theorems of classes of languages has been presented using the computation histories of automata which read and write to and from various potentially infinite data structures. One direction to go from here is to try to represent classes of languages properly contained in the class of recursively enumerable languages with this method. If a class of languages L has the property that

1. L is closed under homomorphism and intersection with regular languages and
2. L is determined by a class of machine acceptors of the type described above whose checking language is itself a member of L,

then we are assured a representation for L in the form of the Chomsky-Shutzenberger theorem. Can we find appropriate machine acceptors for the EOL, ETAG or ETOL languages? (See [HER,ROZ 75], [ROZ 80].

The other direction is to explore the class VSL_Σ of its own accord. What properties must a language have to be in VSL_Σ? What properties guarantee that a language will be in VSL_Σ? It is our hope that the theory of very special languages will shed new light on the structure of sets generated by finite procedures, as do the specific examples of very special languages we have presented here.

ACKNOWLEDGMENT

We would like to thank Andrzej Ehrenfeucht and Gregorz Rozenberg for many helpful discussions concerning this material.

REFERENCES

[BAK,BOO 74]	Baker, B. and Book, R.V. (1974), Reversal-Bounded Multipushdown Machines, JCSS 8, pp. 315-352.
[BLA,HEA 79]	Blattner, M. and Head T., (1979), "The Decidability of Equivalence for Deterministic Finite Transducers," JCSS 19, pp. 45-49.
[BOO,GRE,WRA 78]	Book, R.V., Greibach, S.A., Wrathall, C. (1978), Comparison and reset machines, in "Proc. 5th Colloquium on Automata, Languages and Programming, Lecture Notes in Comp. Sci.", Vol. 62, pp. 113-124.
[BRA 79]	Brandenburg, F. J. (1979), "Analogies of certain families of languages arising from PAL and COPY," presented at the ACM Symposium on Formal Language Theory, Santa Barbara, Calif., Dec. 1979.
[CUL,SAL 78]	Culik, K. and Salomaa A., "On the Decidability of Homomorphism Equivalence for Languages," JCSS 17, pp. 163-175.
[EHR,ROZ 79a]	Ehrenfeucht, A. and Rozenberg, G., 1979, "On the Emptiness of the Intersection of Two DOS Languages Problem," Technical Report CU-CS-159-79, Dept. of Computer Science, Univ. of Colo., Boulder, Colo.
[EHR,ROZ 79b]	Ehrenfeucht, A., Rozenberg, G., 1979, "Representation Theorems Using DOS Languages," Technical Report CU-CS-161-79, Dept. of Computer Science, Univ. of Colo., Boulder, Colo.

[ENG ROZ 80] Engelfriet, J and Rozenberg, G. (1980), "Fixed Point Languages, Equality Languages and Representations of Recursively Enumerable Languages," JACM (27), pp. 499-518.

[FIS RAN 69] Fisher, G. and Raney, G., "On the Representation of Formal Languages using Automata on Networks," IEEE Conference Record, 10th Annual Symposium on Switching and Automata Theory, pp. 157-165.

[HAU 79] Haussler, D., 1979, "Some Results on Symmetric DGSM's and DGSM Equivalence," TechnicalReport CU-CS-199-79, Dept. of Computer Science, Univ. of Colo., Boulder, Colo.

[HER ROZ 75] Herman, G.T., Rozenberg, G., 1975, Developmental Systems and Languages, North-Holland.

[HOP ULL 79] Hopcroft and Ullman, 1979, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, p. 171.

[ROS 66] Rosenberg, A.L.1966, "On Multi-head Finite Automata," IBM Journal of Research and Development, 10, 1966, pp. 388-394

[ROZ 80] Rozenberg, G., Personal communication.

[SAL 73] Salomaa, A., Formal Languages, Academic Press, pp. 107-109.

[SHE STU 63] Shepherdson, J.C. and Sturgis, H.E., 1963, "Computability of Recursive Functions," JACM 10, (2) 1963, pp. 217255.

[YAO RIV 78] Yao, A. and Rivest, R., "K+1 Heads are Better Than K," JACM 25 (2), pp. 337-340.