Student Paper Competition—Third Prize

Shuffle Languages, Petri Nets, and Context-Sensitive Grammars

Jay Gischer
University of Washington

Flow expressions have been proposed as an extension of the regular expressions designed to model concurrency. We examine a simplification of these flow expressions which we call *shuffle expressions*. We introduce two types of machines to aid in recognizing shuffle languages and show that one such machine may be equivalent to a Petri Net. In addition, closure and containment properties of the related language classes are investigated, and we show that one machine type recognizes at least a restricted class of shuffle languages. Finally, grammars for all shuffle languages are generated, and the shuffle languages are shown to be context-sensitive.

Key Words and Phrases: regular languages, shuffle languages, vector machine, Petri net, context-sensitive
CR Categories: 4.32, 5.22, 5.23

I. Shuffle Languages

In this section we define *shuffle expressions and languages*, a simplification of the flow expressions introduced by Shaw in [5].

Definition I.1. Let Σ be a finite set of symbols. Then S is a *shuffle expression* over Σ *iff*

$S = \alpha \in \Sigma$

or S_1 and S_2 are shuffle expressions and

(1) $S = S_1 \cdot S_2$ (also written $S_1 S_2$);
(2) $S = S_1 + S_2$;
(3) $S = S_1^*$;
(4) $S = S_1 \square S_2$;
(5) $S = S_1^{[*]}$; or
(6) $S = (S_1)$.

Clearly, the shuffle expressions are an extension of the regular expressions made by adding two new operators: $\square$, called the shuffle operator, and $[*]$, called the shuffle closure operator. Provision is made for parenthesization.

Definition I.2. For each shuffle expression S we define an associated ~~*shuffle*~~ *language* $L(S)$ as follows:

(1) If $S = \alpha \in \Sigma$, then $L(S) = \{\alpha\}$. Otherwise, if S_1 and S_2 are shuffle expressions, then $x \in L(S)$ *iff*
(2) $S = S_1 \cdot S_2$ and $x = yz$ for some $y \in L(S_1)$ and $z \in L(S_2)$;
(3) $S = S_1 + S_2$ and $x \in L(S_1)$ or $x \in L(S_2)$;
(4) $S = S_1^*$ and $x = \lambda$ or $x = y_1 y_2 \ldots y_n$, for some $n \geq 1$ where each $y_i \in L(S_1)$, $1 \leq i \leq n$;
(5) $S = S_1 \square S_2$, and $x = y_1 z_1 y_2 z_2 \ldots y_n z_n$ $n \geq 1$, $y_1 y_2 \ldots y_n \in L(S_1)$, and $z_1 z_2 \ldots z_n \in L(S_2)$;
(6) $S = S_1^{[*]}$, and $x = \lambda$ or $x \in L(S_1^{[*]n})$ for some $n \geq 1$ where $S^{[*]n}$ is defined by $S^{[*]1} = S$ and $S^{[*]j} = S \square S^{[*](j-1)}$ for all $j \geq 2$; or
(7) $S = (S_1)$ and $x \in L(S_1)$.

The interpretations of $\square$ and $^{[*]}$ are related in much the same way as those of $\cdot$ and $*$, hence the name for $^{[*]}$ is shuffle *closure*, just as $*$ is often called concatenation closure.

Definition I.3. The class of all regular languages will be denoted by $\mathscr{L}_R$, and $\mathscr{L}_S$ will denote the class of all shuffle languages.

Clearly, $\mathscr{L}_R \subseteq \mathscr{L}_S$. We shall show later that $\mathscr{L}_R \neq \mathscr{L}_S$, hence containment is proper, i.e., $\mathscr{L}_R \subset \mathscr{L}_S$. Furthermore, we can extend the regular languages by adding only the $\square$ operator and not $^{[*]}$, creating another class of languages $\mathscr{L}_\square$. However, it has been shown [2] that $\mathscr{L}_R = \mathscr{L}_\square$, so that the addition of $\square$ by itself has no impact on the expressive power of regular expressions. We shall use this fact later. Ginsburg

II. Vector Machines

In this section we define *vector machines*, or n-tuple machines, and their associated languages. Central to the idea of a vector machine are functions which act on n-tuples $b = (b_1, \ldots, b_n)$ of nonnegative integers.

Definition II.1. A *transformation* t of n-tuples consists of an ordered pair of multisets of functions, $(\{p_{i_1}, \ldots, p_{i_k}\}, \{v_{j_1}, \ldots, v_{j_l}\})$, where $1 \leq i_r \leq n$, $1 \leq j_r \leq n$. The primitives which are the elements of these multisets are defined by setting, for each i, $1 \leq i \leq n$

This work supported by NSF Grant #MCS77-02474.

Author's Present Address: Jay Gischer, Department of Computer Science, Stanford University, Stanford, CA 94306.

$$p_i(b) = \begin{cases}(b_1, \ldots, b_{i-1}, b_i - 1, b_{i+1}, \ldots, b_n), & \text{if } b_i > 0\\ \text{undefined if } b_i = 0, \text{ and}\end{cases}$$

$v_i(b) = (b_1, \ldots b_{i-1}, b_i+1, b_{i+1}, \ldots, b_n)$, for all $b_i \geq 0$. Since $p_i \circ p_j = p_j \circ p_i$ and $v_i \circ v_j = v_j \circ v_i$, a multiset of p or v functions gives a well-defined partial function the composition of its constituent primitives that is independent of the ordering of the individual primitives. We denote this function by $\{p_{i_1}, \ldots, p_{i_k}\}(b)$ or $\{v_{j_1}, \ldots, v_{j_l}\}(b)$. Finally $t(b) = \{v_{j_1}, \ldots, v_{j_l}\}[\{p_{i_1}, \ldots, p_{i_k}\}(b)]$ is a well-defined partial function $t: N^n \rightarrow N^n$. We denote by T^n the set of all transformations defined on n-tuples. Although strictly speaking $T^n \cap T^{n+1} = \emptyset$, since the domains are disjoint, for $t \in T^n$ there is a natural extension to a $t' \in T^{n+1}$ given by defining $t'(b, n)$ to be $[t(b), n]$. Therefore we will subsequently identify t with t' and consider T^n to be a subset of T^{n+1}.

Definition II.2. An *n-tuple machine* is a 4-tuple (Σ, R, S, H) where Σ is a finite set of symbols, R is a finite subset of $(\Sigma \cup \{\lambda\}) \times T^n$, $S \in N^n$, and H is a finite subset of T^n.

Definition II.3. Let $M = (\Sigma, R, S, H)$ be any n-tuple machine. Any sequence $r_1, \ldots, r_p$ of elements of R is called a *computation sequence* of M. If $r_1, \ldots, r_p$ is a computation sequence, with $r_i = (\alpha_i, t_i)$, for $1 \leq i \leq p$, such that (1) There exists a sequence $h_1, \ldots, h_q$, $q \geq 0$, with $h_j \in H$ for all j, $1 \leq j \leq q$, such that $h_q \circ \ldots \circ h_1, \circ t_p \circ \ldots \circ t_1(b)$ is defined and equal to $0 = (0, \ldots, 0) \in N^n$, and (2) $x = \alpha_1 \alpha_2 \ldots \alpha_p$, then $r_1, \ldots, r_p$ is an *accepting sequence* for x of M, $t_p \circ \ldots \circ t_1$, an *accepting function* for x_1, and $h_q \circ \ldots \circ h_1$ a *clearing function* of $r_1, \ldots, r_p$. The language $L(M)$ accepted by M is the subset of Σ^* consisting of precisely those elements of Σ^* which have accepting sequences.

As an example of a vector machine and its associated language we construct the following 6-tuple machine M.

Let $M = (\Sigma, R, S, H)$, where

$$\begin{aligned}
\Sigma &= \{a, b, c\};\\
S &= (0, 0, 0, 1, 0, 0);\\
R &= \{(a, (\{p_4\}, \{v_1, v_4\})) = r_1;\\
&\quad (b, (\{p_1, p_5\}, \{v_2, v_5\})) = r_2;\\
&\quad (c, (\{p_2, p_6\}, \{v_3, v_6\})) = r_3;\\
&\quad (\lambda, (\{p_4\}, \{v_5\})) = r_4;\\
&\quad (\lambda, (\{p_5\}, \{v_6\})) = r_5\}; \text{ and}\\
H &= \{((\{p_3\}, \{\}), (\{p_6\}, \{\}))\} = \{h_1, h_2\}
\end{aligned}$$

Note: $\{\}$ for a p or v multiset denotes the identity function.

LEMMA II.1. $L(M) = \{a^n b^n c^n \mid n \geq 0\}$

PROOF. First we observe that any accepting sequence must begin with either a r_1 or a r_4. If it begins with a r_4, then there is no way of setting b_4 to 1 again, since a v_4 only occurs preceded by a p_4. Hence r_1 can never be used. But if r_1 cannot occur, neither can r_2 or r_3. Furthermore, if we wish to use transformations in H in a clearing sequence, we must have $b_5 = 0$ before applying the clearing function. Thus we are forced to use r_5. At this point, no transformations will apply, and we use h_2 as the clearing function, getting $h_2 \circ t_5 \circ t_4(S) = 0$. So λ, the empty string, is in $L(M)$, and is the only string possible for accepting sequences beginning with r_4.

On the other hand, assume that some accepting sequence begins with exactly $k \geq 1$ occurrences at r_1. This cannot be all of the sequence, since $t_1^k(S) = (k, 0, 0, 1, 0, 0)$; no clearing sequence exists for this vector. The only other transformation defined at this point is t_4, so r_4 must come next. Now both t_2 and t_5 are defined, but if t_5 is applied before k applications of t_2, this will make $b_1 > 0$ and $b_5 = 0$. There is no way of making $b_5 > 0$ again, hence b_1 cannot be reduced to 0 either by transformations or clearing functions. We conclude that such a sequence cannot be an accepting one. Further, there can be no more than k occurrences of r_2, since after applying t_2^k, we have $b_1 = 0$. Since the function t_1 can never be used after a t_4, we cannot make $b_1 > 0$ after this point, and another t_2 would be necessarily undefined. So it must be that any accepting sequence has the prefix $(r_1)^k, r_4, (r_2)^k$. At this point, $t_2^k \circ t_4 \circ t_1^k(S) = (0, k, 0, 0, 1, 0)$. There certainly is no clearing sequence for this computation sequence, hence there must be more elements in any accepting sequence. Since the only transformation defined at this point is t_5, r_5 must be the next element in any accepting sequence.

By an analogous argument to the above, k occurrences of r_3 must follow. Thus any accepting sequence has the form (A) $(r_1)^k, r_4, (r_2)^k, r_5, (r_3)^k$, $k \geq 0$, since t_i is undefined at this point for all i, $1 \leq i \leq 5$. But now $h_1^k \circ h_2 \circ t_3^k \circ t_5 \circ t_2^k \circ t_4 \circ t_1^k(S) = 0$. Thus we can conclude that $\sigma = r_{i_1}, \ldots, r_{i_l}$ is an accepting sequence *if and only if* σ is of the form (A) above. From this it follows immediately that $L(M) = \{a^n b^n c^n \mid n \geq 0\}$. ●

We have given the above proof in great detail so as to familiarize the reader with vector machines and their operation. We can now define the class of vector machine languages.

Definition II.4. We define $\mathscr{L}_{VM}$ to be the class of all languages L such that $L = L(M)$ for some vector machine M.

Finally, we wish to define a slightly more powerful class of machines.

Definition II.5. Define a new set of primitive partial functions $\bar{p}_i: N^n \rightarrow N^n$, $1 \leq i \leq n$, which are defined on $b = (b_1, \ldots, b_n)$ by:

$$\bar{p}_i(b) = \begin{cases} b \text{ iff } b_i = 0\\ \text{undefined otherwise.}\end{cases}$$

Next, *augmented transformations* $t: N^n \rightarrow N^n$ are the same as ordinary transformations except that the multiset of p functions in an augmented transformation may also contain $\bar{p}$ functions. However, since $p_i \circ \bar{p}_i \neq \bar{p}_i \circ p_i$, for all i, $1 \leq i \leq n$, we make the restriction that a multiset of p functions may not contain both p_i and $\bar{p}_i$, for all i, $1 \leq i \leq n$. Let T^{n+} denote the set of all augmented transformation defined on n-tuples.

An *augmented vector machine* is a 4-tuple $M = (\Sigma, R, S, H)$, where Σ and S are defined as before and $R \subset (\Sigma \cup \{\lambda\}) \times T^{n+}$, $H \subset T^{n+}$.

Finally, $\mathscr{L}_{VM+}$ is the class of all languages $L(M)$ where M is an augmented vector machine.

Immediately we have $\mathscr{L}_{VM} \subseteq \mathscr{L}_{VM+}$. However, whether $\mathscr{L}_{VM} = \mathscr{L}_{VM+}$ is still an open question.

III. Petri Nets and Vector Machines

In this section, we examine the relationship between vector machines and Generalized Petri Nets, as discussed in Hack [3].

III.1. Petri Nets and Petri Net Languages

Definition III.1. A *Generalized Petri Net* (GPN) $N = (\Pi, T, F, B, M_0)$ consists of the following:

(1) a finite set of *places*, $\Pi = \{\pi_1, \ldots, \pi_r\}$;
(2) a finite set of *transitions*, $T = \{\tau_1, \ldots, \tau_s\}$;
(3) a *forwards incidence function*, $F: \Pi \times T \rightarrow N$;
(4) a *backwards incidence function*, $B: \Pi \times T \rightarrow N$; and
(5) an *initial marking* $M_0: \Pi \rightarrow N$.

A transition τ is said to be *firable* if for every place π we have $M(\pi) \geq F(\pi, \tau)$. If a firable transition τ *fires*, then the marking M of the net is changed to a new marking M', given by setting for all $\pi \in \Pi$,

$$M'(\pi) = M(\pi) - F(\pi, \tau) + B(\pi, \tau)$$

A *firing sequence* of N can now be defined as a sequence σ of transitions such that each prefix of σ leads to a marking such that the following transition is firable.

Definition III.2. A *Labelled Petri Net* $A = (N, \Sigma, \Lambda, M_r)$ over an alphabet Σ is a GPN $N = (\Pi, T, F, B, M_0)$ together with a *labelling function* $\Lambda: T \rightarrow \Sigma$ and a *final marking* $M_f: \Pi \rightarrow N$. If the labelling function is only partial, the net is said to contain λ *transitions*, namely those transitions τ for which $\Lambda(\tau)$ is undefined.

We can extend Λ to firing sequences in a natural way: If $\tau \in \mathrm{T}$, $\sigma \in \mathrm{T}^*$, then

$$\begin{aligned}\Lambda(\sigma\tau) &= \Lambda(\sigma)\cdot\Lambda(\tau) && \textit{iff } \Lambda(\tau) \text{ is defined}\\ &= \Lambda(\sigma) && \text{otherwise, and finally, } \Lambda(\lambda) = \lambda.\end{aligned}$$

Definition III.3. $\mathscr{L}_0^\lambda$ is the class of all languages obtained as the set of all *terminal label sequences* associated with a Labelled Petri Net as follows:

$L \in \mathscr{L}_0^\lambda$ *iff* there exists $A = (N, \Sigma, \Lambda, M_v)$ a Labelled Petri Net such that $x \in L$ *iff* $\exists\ \sigma$ a firing sequence beginning at M_0 and leading to M_f such that $\Lambda(\sigma) = x$.
L is written as $L(A)$.

Another class of languages $\mathscr{L}_0$ can be defined in an analogous way with the additional requirement that Λ be total.

III.2. Petri Nets and Vector Machines

Definition III.4. Given a Labelled Petri Net $A = (N, \Sigma, \Lambda, M_f)$ where $N = (\Pi, T, F, B, M_0)$, we define $M_A = (\Sigma, R, S, H)$ to be a vector machine in the following way:

(1) Let $\mathrm{ord}(\pi) = n$. Then M_N will be an $n + 1$-tuple machine. Renumber the places in Π if necessary so that they are numbered π_1 to π_n.
(2) Define $S = (s_1, \ldots, s_n, 1)$, where $s_i = M_0(\pi_i)$, $1 \leq i \leq n$.
(3) Let $t_i \in T^n$ be defined by $t_i = (P, V)$, where p_j occurs in P exactly $F(\pi_j, \tau_i)$ times and v_k occurs in V exactly $B(\pi_k, \tau_i)$ times. Define $v_i = [\Lambda(\tau_i), t_i]$ *iff* $\Lambda(\tau_i)$ is defined and $v_i = (\lambda, t_i)$ otherwise. Finally define $R = (r_1, \ldots, r_l)$ where $l = \mathrm{ord}(T)$.
(4) Let $h = (P \cup \{p_{n+1}\}, \{\})$, where p_i occurs in P exactly $M_v(\pi_i)$ times. Define $H = \{h\}$.

LEMMA III.1. $L(A) = L(M_A)$
PROOF: Let $b_M = (b_1, \ldots, b_n, 1)$ denote, for a marking M, a vector such that $b_i = M(\pi_i)$. Now from (3) above, we have that τ_i is firable at M leading to M' *iff* $t_i(b_M) = b_{M'}$. It follows inductively that $\sigma = \tau_{i_1}\tau_{i_2}\ldots\tau_{i_l}$ is a firing sequence leading to M from M_0 *iff* $t_{i_l} \circ \ldots \circ t_{i_1}(S) = b_M$.

So σ is a terminal firing sequence (i.e., σ leads to M_f from M_0) *iff* $t_{i_l} \circ \ldots \circ t_{i_1}(S) = b_{M_f}$. But $h(b_{M_f}) = \mathbf{0}$, $h(b')$ is undefined for $b' < b_{M_f}$, and $h[h(b)]$ is undefined for all b_M. We conclude that $r_{i_1}, \ldots, r_{i_l}$ is an accepting sequence *iff* σ is a terminal sequence. Since $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_l} = \Lambda(\sigma)$, we have $L(A) = L(M_A)$. ●

Definition III.5. Given an n-tuple machine $M = (\Sigma, R, S, H)$ we define the Labelled Petri Net $A_M = (N, \Sigma, \Lambda, M_f)$ where $N = (\Pi, T, F, B, M_0)$ by the following

(1) $\Pi = \{\pi_1, \ldots, \pi_n, \pi_{\mathrm{run}}, \pi_{\mathrm{clear}}\}$,
(2) For all $r_i = (\alpha_i, t_i) \in R$, let $\tau_i \in T'(1 \leq i \leq m)$ (renumbering r's, α, and t's if necessary to include all $r \in R$).
(3) Let $F(\pi_i, \tau_i) =$ the number of occurrences of p_i in t_j (that is, if $t_j = (P, V)$ the number of p_i's in P) for all i, j $1 \leq i \leq n$, $1 \leq j \leq m$ and $F(\pi_{\mathrm{run}}, \tau_j) = 1$, for all j, $1 \leq j \leq m$.
(4) Let $B(\pi_i, \tau_j) =$ the number of v_i's in t_j, for all i, j $1 \leq i \leq n$, $1 \leq j \leq m$ and define $B(\pi_{\mathrm{run}}, \tau_j) = 1$, for all j, $1 \leq j \leq m$.
(5) Define M_0 by $M_0(\pi_i) = S_i$, $M_0(\pi_{\mathrm{run}}) = 1$, and $M_0(\pi_{\mathrm{clear}}) = 0$.
(6) Define a new set of transitions T_h, including τ_s. Let $F(\pi_{\mathrm{run}}, \tau_s) = 1$ and $F(\pi, \tau_s) = 0$ for $\pi \neq \pi_{\mathrm{run}}$. Also $B(\pi_{\mathrm{clear}}, \tau_s) = 1$ and $B(\pi, \tau_s) = 0$ for $\pi \neq \pi_{\mathrm{clear}}$. For each $h_i \in H$ include τ_{h_i} in T_h. Let $F(\pi_j, \tau_{h_i}) =$ the number of p_j's in h_i, $1 \leq j \leq n$. Let $B(\pi_j, \tau_{h_i}) =$ the number of v_j's in h_i, $1 \leq j \leq n$. Let $F(\pi_{\mathrm{clear}}, \tau_h) = B(\pi_{\mathrm{clear}}, \tau_h) = 1$ for all $h \in H$. Finally, include in T_n τ_{stop}, with $F(\pi_{\mathrm{clear}}, \tau_{\mathrm{stop}}) = 1$, $F(\pi, \tau_{\mathrm{stop}}) = 0$ for $\pi \neq \pi_{\mathrm{clear}}$, and $B(\pi, \tau_{\mathrm{stop}}) = 0$, $\pi \in \Pi$.
(7) Let $T = T' \cup T_h$.

(8) For τ_i in T' let $\Lambda(\tau_i) = \alpha_i$, while for all other $\tau \in T$ $\Lambda(\tau_i)$ is undefined.
(9) Define $M_f(\pi) = 0$ for all $\pi \in \Pi$.
This completely defines A_M.

LEMMA III.2. $L(A_M) = L(M)$.
PROOF. Again we use the notation $b_M = (b_1, \ldots, b_n)$, for a marking M, to denote the vector such that $b_i = M(\pi_i)$ $1 \le i \le n$. Also, for a firing sequence σ beginning at M_0, let M_σ be the marking led to by σ. Note that $M_0(\pi_{\text{run}}) = 1$. Further, since $B(\pi_{\text{run}}, \tau) = 1$ for all $\tau \in T'$, if σ is a firing sequence with $M_\sigma(\pi_{\text{run}}) = 0$, then $\sigma = \sigma_1\tau_s\sigma_2$, and for all firing sequences $\eta = \sigma\gamma(\eta, \gamma \in T^*)$ we have that $M_\eta(\pi_{\text{run}}) = 0$. In the same way $M(\pi_{\text{clear}}) = 0$ until τ_s is fired. Finally, if σ is a firing sequence leading to M_f, then τ_s occurs in σ for otherwise we would have $M_\sigma(\pi_{\text{run}}) = 1$, implying $M_\sigma \neq M_f$. So every terminal firing sequence σ can be decomposed into $\sigma = \sigma_1\tau_s\sigma_2$, where $\sigma_1 \in T'^*$ and $\sigma_2 \in T_h^* - \{\tau_s\}$.

Now observe that $b_{M_0} = S$. Let $\sigma = \tau_{i_1} \ldots \tau_{i_k} \in T'^*$ and assume that $t_{i_k} \circ \ldots \circ t_{i_1}(S) = b_{M_\sigma}$. It is clear from the definition of F and B that for $\sigma' = \sigma\tau_{i_{k+1}}$, $t_{i_{k+1}} \circ t_{i_k} \circ \ldots \circ t_{i_1}(S) = b_{M_{\sigma'}}$. Hence by induction we have $\sigma = \tau_{i_1} \ldots \tau_{i_m}$ is a firing sequence beginning on M_0 *iff* $t_{i_m} \circ \ldots \circ t_{i_1}(S) = b_{M_\sigma}$.

Let $r_{i_1}, \ldots, r_{i_m}$ be an accepting sequence for $x \in \Sigma^*$. There exists a clearing sequence $h_{j_1}, \ldots, h_{j_l}$ for the accepting sequence, i.e., $h_{j_l} \circ \ldots \circ h_{j_1} \circ t_{i_m} \circ \ldots \circ t_{i_1}(S) = 0$. Then there is a corresponding sequence σ' in T^* with $\sigma' = \sigma_1\tau_s\sigma_2$, where $\sigma_1 \in T''^*$ and $\sigma_2 \in T_h^*$ such that $b_{M_{\sigma'}} = 0$, $M_{\sigma'}(\pi_{\text{clear}}) = 1$, and $M_{\sigma'}(\pi_{\text{run}}) = 0$. Thus $\sigma\tau_{\text{stop}}$ is a terminal firing sequence with

$$\begin{aligned}\Lambda(\sigma\tau_{\text{stop}}) &= \Lambda(\sigma_1)\\ &= \Lambda(\tau_{i_1})\cdot\Lambda(\tau_{i_2})\cdot \ldots \cdot\Lambda(\tau_{i_m})\\ &= \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_n} = x.\end{aligned}$$

Hence $L(A_M) \supset L(M)$.

Conversely, suppose σ is a terminal firing sequence and $\Lambda(\sigma) = x$. Then $\sigma = \sigma'\tau_{\text{stop}}$ since a firing of τ_{stop} leads to M such that $M(\pi_{\text{run}}) = M(\pi_{\text{clear}}) = 0$. Now $\sigma' = \sigma_1\tau_s\sigma_2$, $\sigma_1 \in T'^*$, $\sigma_2 \in T_h^*$. Let $r_{i_1}, \ldots, r_{i_n}$ be the computation sequence corresponding to $\sigma_1 = \tau_{i_1}, \ldots \tau_{i_n}$, then $t_{i_n} \circ \ldots \circ t_{i_1}(s) = b_{M_{\sigma_1}}$; further let $h_i, \ldots h_m$ be the corresponding sequence to σ_2. Then since $b_{M_{\sigma'}} = 0$, we must have that $h_m \circ \ldots \circ h_1 \circ t_{i_n} \circ \ldots \circ t_{i_1}(S) = 0$ and $r_{i_1} \ldots r_{i_n}$ is an accepting sequence for some $y \in \Sigma^*$. But now,

$$\begin{aligned}y &= x_{i_1}x_{i_2}\ldots x_{i_n}\\ &= \Lambda(\tau_{i_1})\cdot \ldots \cdot\Lambda(\tau_{i_n})\\ &= \Lambda(\sigma_1)\\ &= \Lambda(\sigma_1\tau_s\sigma_2\tau_{\text{stop}})\\ &= \Lambda(\sigma)\\ &= x.\end{aligned}$$

Thus $L(A_M) = L(M_{\bar{M}})$. ●

We have proved the following theorem:

THEOREM III.1 $\mathscr{L}_0^\lambda = \mathscr{L}_{VM}$

PROOF: Immediate from Lemmas III.1, and III.2. ●

IV. Vector Machines and $\mathscr{L}_s$

In this section we examine some of the closure properties of vector machine languages. Then we will use these properties to examine the relationship between $\mathscr{L}_{VM}$, $\mathscr{L}_{VM+}$, and $\mathscr{L}_s$, the shuffle languages. We begin by showing containment of the regular languages, $\mathscr{L}_R$ in $\mathscr{L}_{VM}$.

IV.1. Vector Machines and Finite Automata

We give the definition of a finite automata without comment. Further reference may be found in Hopcroft and Ullman [4].

Definition IV.1. A deterministic finite automaton (dfa) is a 5-tuple $A = (\Sigma, \delta, Q, q_0, H)$ where

(1) Σ is a finite set of *symbols*;
(2) Q is a finite set of *states* q;
(3) δ is a function $\delta: Q \times \Sigma \rightarrow Q$;
(4) $q_0 \subset Q$ is called the *start state*; and
(5) $H \subset Q$ is a set of *halt states*.

δ can be extended to map $Q \times \Sigma^* \rightarrow Q$ in the following way: Let $x \in \Sigma^*$ and $x \in \Sigma$. Then we define $\delta(q_0, x\alpha) = \delta[\delta(q_0, x), \alpha]$ and $\delta(q_0, \lambda) = q_0$, which allows us to define the language $L(A)$.

Definition IV.2. $x \in L(A)$ *iff* $\delta(q_0, x) \in H$. $\mathscr{L}_{fa} = \{L(A) | A$ is a dfa$\}$. It has been shown that $\mathscr{L}_R = \mathscr{L}_{fa}$. Therefore, to show $\mathscr{L}_R \subset \mathscr{L}_{VM}$, we show $L(A) \in \mathscr{L}_{VM}$ for every dfa A.

Definition IV.3. Given $A = (\Sigma, \delta, Q, q_0, H)$, a dfa defines the vector machine $M_A = (\Sigma, R, S, \bar{H})$ in the following way:

(1) Let $n = \text{ord}(Q)$ M_A will be an n-tuple machine.
(2) Renumber the states in Q, if necessary, as $q_1, \ldots q_n$ such that q_1 is the start state.
(3) For all i $1 \le i \le n$ and all $\alpha \in \Sigma$, let $t_{i,\alpha} = (\{P_i\}, \{v_j\})$ where $\delta(q_i, \alpha) = q_j$ and let $r_{i,\alpha} = (\alpha, t_{i,\alpha})$. Then $R = \{r_{i,\alpha} | 1 \le i \le n, \lambda \in \Sigma\}$.
(4) Define $S = (1, 0, \ldots 0)$.
(5) Let $\bar{H} = \{t | t = (\{p_i\}, \{\})$ and $q_i \in H\}$.

LEMMA IV.1. $L(M_A) = L(A)$
First we introduce some notation. T_i, for $1 \le i \le n$, is a predicate such that $T_i(b)$, where $b = (b_1, \ldots b_n)$ holds *iff* $b_i = 1$ and $b_j = 0$ $i \neq i$. We assert that for any defined computation sequence $r_1, \ldots r_n$, with $x = \alpha_1 \ldots \alpha_n$, and $t_n \circ \ldots \circ t_1(s) = b$, $T_i(b)$ holds *iff* $\delta(q_1, x) = q_i$. Clearly, $T_1(S)$ holds and $\sigma(q_1, \lambda) = q_1$. Now inductively assume that we have our assertion for defined computation sequences of length k. Let $r_1 \ldots r_k, r_{k+1}$ be a computation sequence, $x = \alpha_1 \ldots \alpha_k$, and $t_k \circ \ldots \circ t_1(S)$

$= b$. By the induction hypothesis, $T_i(b)$ holds where $\delta(q_1, x) = q_i$. Now $t_{k+1} = t_{i,\alpha}$ for some $\alpha \in \Sigma$, since $r_1, \ldots, r_k, r_{k+1}$ is a defined computation sequence. Furthermore $t_{i,\alpha} = (\{p_i\}, \{v_j\})$ *iff* $\delta(q_i, x) = q_j$. Therefore, $T_j[t_{i,\alpha}(b)] = T_j[t_{k+1}\circ, t_k\circ \ldots \circ t_1(S)]$ holds *iff* $\delta[\delta(q_1, x), \alpha] = \delta(q_i, \alpha) = q_j$. So by induction we have proved our assertion.

Now suppose that $r_1, \ldots r_m$ is an accepting sequence. Then $t_m\circ \ldots \circ t_1(s)$ is defined and for $x = \alpha_1 \ldots a_m$ and $\delta(q_1, x) = q_i$, $T_i[t_m\circ \ldots \circ t_1(s)]$ holds for some i, $1 \le i \le n$. Since the sequence is accepting, $\exists$ h $\in$ H such that $h\circ t_m\circ \ldots t_1(S) = \mathbf{0}$. Therefore, we know that $h = (\{p_i\}, \{\})$, but $(\{p_i\}, \{\}) \in H$ *if and only if* $q_i \in H$, hence $x \in L(M_A)$ *iff* $x \in L(A)$, giving us $L(M_A) = L(A)$. ●

THEOREM IV.1. $\mathscr{L}_R \subseteq \mathscr{L}_{VM}$ ($\subseteq \mathscr{L}_{VM+}$)

PROOF. Immediate from Lemma IV.1 and $\mathscr{L}_{fa} = \mathscr{L}_R$. ●

IV.2. Closure Properties of $\mathscr{L}_{VM}$ and $\mathscr{L}_{VM+}$.

In this section we prove some closure properties for $\mathscr{L}_{VM}$ and $\mathscr{L}_{VM+}$. In order to aid in the constructions we adopt some notation: For any transformation t and any positive integer k, let $t + k$. be the transformation which results from t by increasing the indices of its component p and v functions by k.

LEMMA IV.2. *If $M = (\Sigma_M, R_M, S_M, H_M)$ and $N = (\Sigma_N, R_N, S_N, H_N)$ are m and n-tuple machines, respectively, then $\exists\ \bar M = (\bar\Sigma, \bar R, \bar S, \bar H)$, an $m + n + 3$-tuple machine with $L(\bar M) = L(M) \cup L(N)$.*
PROOF. Let $\bar\Sigma = \Sigma_M \cup \Sigma_N$. The idea is to let the 4th through $(m + 3)$rd place of $\bar M$ simulate M while the $(m + 4)$th through $(m + n+3)$rd simulate N. The first three places are used in starting the machine properly.
Define $\bar S = (1, 0, 0, \ldots, 0)$. Now let $\bar r \in \bar R$ *iff*

(1) $\bar r = (\alpha, \bar t_i)$, where $r = (\alpha, t_i) \in R_M$, $\bar t_i = (\bar P \cup \{p_2\}, \bar V \cup \{v_2\})$, where $(\bar P, \bar V) = t_i + 3$. (The set of all such $\bar r$ we call $\bar R_M$);
(2) $\bar r = (\alpha, \bar t_j)$, where $r = (\alpha, t_j) \in R_M$, $\bar t_j = (\bar P \cup \{p_3\}, \bar V \cup \{v_3\})$, where $(\bar P, \bar V) = t_i + m + 3$. (The set of all these $\bar t_j$'s is $\bar R_N$);
(3) $\bar r = r_{S_M} = [\lambda, (\{p_1\}, \{v_2\} \cup V_M)$, where $v_{i+3} \in V_M$ *iff* $S_{M_i} = 1$; or
(4) $\bar r = r_{S_N} = [\lambda, (\{p_1\}, \{v_3\} \cup V_N)$, where $v_{i+m+3} \in V_N$ *iff* $S_{N_i} = 1$.

Finally, $\bar H = \{h + 3 \mid h \in H_M\} \cup \{h + m + 3 \mid h \in H_N\} \cup \{(\{p_2\}, \{\}), (\{p_3\}, \{\})\}$. Suppose $x \in L(M)$. There exists an accepting sequence $r_1, \ldots, r_l$ in R_M and $h_1, \ldots, h_k$ in H_M such that $h_k\circ \ldots \circ h_1 \circ t_l\circ \ldots \circ t_1(S_M) = \mathbf{0}$. Then $(h_k + 3)\circ \ldots \circ (h_1 + 3)\circ \bar t_l\circ, \ldots \circ \bar t_l\circ \ldots \circ \bar t_1 \circ t_{S_K}(\bar S) = \mathbf{0}$ ($\bar t_i \in \bar R_M$). Thus $r_{S_M}, r_1, \ldots, r_l$ is an accepting sequence for x in $\bar M$. We conclude that $L(M) \subset L(\bar M)$. In a similar manner it can be shown that $L(N) \subset L(\bar M)$, giving us $L(\bar M) \supset L(M) \cup L(N)$.

Suppose $x \in L(\bar M)$. The first component of any accepting sequence for x must be either r_{S_M} or r_{S_N} since only t_{S_M} and t_{S_N} are defined on $\bar S$. Suppose r_{S_M} is first. Then h_1 will never again be nonzero and r_{S_N} cannot occur in any accepting sequence for x, nor can any $\bar r \in \bar R_N$. Hence the sequence has the form $r_{S_M}, \bar r_1, \ldots \bar r_l$ where $\bar r_i \in \bar R_M$.

$t_l\circ \ldots \circ t_1(S_M) = b$ for $t_i \in T_M$, *iff* $\bar t_l\circ \ldots \circ \bar t_1 \circ t_{S_M}(\bar S) = (0, 1, 0, b, 0, \ldots, 0)$, by the definitions of $\bar t_i$ and $\bar t_{S_M}$.

$\exists\ \bar h_1, \ldots \bar h_k$ such that $\bar h_k\circ \ldots \circ \bar h_1 \circ t_l \circ \ldots \bar t_1\, t_{S_M}(\bar S) = \mathbf{0}$. One of these h_i's must be $(\{p_2\}, \{\})$. We can assume without loss of generality that this is $\bar h_1$. Then $\bar h_2, \ldots, h_k$ are simply $h_2 + 3, \ldots, h_n + 3$ where $h_i \in H_M$. Otherwise they would not be defined. Then $r_1, \ldots\ r_n$ is an accepting sequence in M, since $h_k\circ \ldots \circ h_2 \circ t_l \circ \ldots \circ t_1(S_M) = \mathbf{0}$ must hold. Therefore we have $x \in L(M)$. If r_{S_N} is the first element of an accepting sequence for x, then we can show in the same way that $x \in L(N)$. So $L(\bar M) \subset L(M) \cup L(N)$.

Hence we have $L(\bar M) = L(M) \cup L(N)$. ●

LEMMA IV.3. *If M and N are m and n-tuple machines, respectively, then there exists $\bar M = (\bar\Sigma, \bar R, \bar S, \bar H)$, an $(m + n + 2)$-tuple machine with $L(M) = L(\bar M)\cdot L(N)$.*
PROOF. Let $\bar\Sigma = \Sigma_M \cup \Sigma_N$. Define $\bar R$ by letting $\bar r \in \bar R$ *iff*

(1) $\bar r = (\alpha, \bar t_i)$, where $(\alpha, t_i) \in R_M$, $\bar t_i = (\bar P \cup \{p_1\}, \bar V \cup \{v_1\})$, and $\{\bar P, \bar V\} = t_i + 2$ (call the set of these $\bar R_M$);
(2) $\bar r = (\alpha, \bar t_j$, where $(\alpha, t_j) \in R_M$, $\bar t_j = (\bar P \cup \{p_2\}, \bar V \cup \{v_2\})$, and $\{\bar P, \bar V\} = t_j + (2 + m)$ (the set of these is $\bar R_N$); or
(3) $\bar r = r^* = (\lambda, t^*)$, $t^* = (\{p_1\}, \{v_2\})$.

Define $\bar H = (H_M + 2) \cup (H_N + m + 2) \cup \{(\{p_2\}, \{\})\}$, (where $H + k = \{h + k \mid h \in H\}$). Finally define $\bar S = (1, 0, S_M, S_N)$. The idea of this machine is that it runs simulating M until r^* occurs, and then it runs simulating N. Let $x \in L(M)\cdot L(N)$. Then $\exists\ r_1, \ldots r_l \in R_M$ and $r'_1, \ldots, r'_k \in R_N$, accepting sequences for y_1 in M and y_2 in N, respectively, with $x = y_1y_2$. Then $\bar r_1, \ldots \bar r_l, r^*, \bar r', \ldots \bar r'_k$ is an accepting sequence for x in $\bar M$, giving $L(\bar M) \supset L(M)\cdot L(N)$.

Conversely, let $r_1, \ldots r_p$ be an accepting sequence for x in $\bar M$. Then for some i_0, $1 \le i_0 \le p$, $r_{i_0} = r^*$, since $\bar S_1 = 1$ and the only transformation with a net decrease of b_1 is t^*. Further for every j, $1 \le j \le i_0$, $r_j \in \bar R_M$, i.e., $r_j = \bar r_{k_0}$ for $r_{k_c} \in R_M$. Also, for all k $i_0 < k \le p$ $r_k \in \bar R_N$. So we rewrite the accepting sequence as $\bar r_1, \ldots, \bar r_k, r^*, \bar r'_1, \ldots \bar r'_l$. Since this is an accepting sequence $\exists\ h_1, \ldots, h_q$ in $\bar H$ such that $h_q\circ \ldots \circ h_1 \circ \bar t'_l \circ \ldots \bar t'_1 \circ t^* \circ \bar t_k \circ \ldots \circ \bar t\,(\bar S) = \mathbf{0}$. One of the h_i's must be $(\{p_2\}, \{\})$, to get $b_2 = 0$. Since this function commutes with all other $h \in H$ assume that $h_1 = (\{p_2\}, \{\})$. Now $h_2, \ldots, h_q$ can be divided into two disjoint sequences of functions $\bar h_{i_1}, \ldots, \bar h_{i_l} \in (H_M + 2)$ and $\bar h_{i'_1}, \ldots, \bar h_{i'_q} \in (H_N + m + 2)$. This gives us, since $\bar S = (1, 0, S_M, S_N)$, that it must be $h_{i_l}\circ \ldots h_{i_1} \circ t_k \circ \ldots \circ t_1(S_M) = \mathbf{0}$ and $h_{i'_q}\circ \ldots \circ h_{i'_1} \circ t'_l \circ \ldots \circ t'_1(S_N) = \mathbf{0}$, making $r_1, \ldots, r_k$ and $r'_1, \ldots r'_l$ accepting sequences in M and N of $y_1 = \alpha_1 \ldots \alpha_l$ and $y_2 = \alpha'_1 \ldots \alpha'_q$, respectively.

But $x = \alpha_1 \ldots \alpha_l \alpha'_1 \ldots \alpha'_q = y_1 \cdot y_2$, hence $x \in L(M) \cdot L(N)$.

This gives us $L(\bar{M}) = L(M) \cdot L(N)$. ●

LEMMA IV.4. *Let M, N be m- and n-tuple machines. Then $\exists\ \bar{M}$ and $(m + n)$-tuple machines with $L(\bar{M}) = L(M) \square L(N)$.*

PROOF. Suppose $M = (\Sigma_M, R_M, S_M, A_M)$ and $N = (\Sigma_N, R_N, S_N, H_N)$. We construct $\bar{M} = (\bar{\Sigma}, \bar{R}, \bar{S}, \bar{H})$ as follows:

Let $\bar{\Sigma} = \Sigma_M \cup \Sigma_N$. Define the following sets by $\bar{R}_M = \{r \mid r \in R_M\}$ and $\bar{R}_N = \{(\alpha, t + m) \mid (\alpha, t) \in R_N\}$. Then define $\bar{R} = \bar{R}_M \cup \bar{R}_N$. Next, let $\bar{S} = (S_M, S_N)$. Finally, let $H_N = (H_N + m)$. Then $H = H_M \cup \bar{H}_N$.

Clearly, transformations originally from different machines commute with one another, i.e., for $r_1 \in \bar{R}_M$ and $r_2 \in \bar{R}_N$. Then $t_1 \circ t_2 = t_2 \circ t_1$. It follows that if $r_1, \ldots r_k$ and $r'_1, \ldots, r'_l$ are accepting sequences for x and y in M and N, respectively, then for $z \in x \square y$, there is a corresponding sequence in $(r_1, \ldots, r_k) \square (\bar{r}'_1, \ldots, \bar{r}'_l) = \bar{r}_{i_1}, \ldots \bar{r}_{i_{l+k}}$, with $\alpha_{i_1} \ldots \alpha_{i_{l+k}} = z$, which has a computational function defined on $\bar{S}$. Also, $\exists\ h_1 \ldots h_p \in H_M$ and $\bar{h}_1 \ldots \bar{h}_q \in H_N$ such that $h_p \circ \ldots h_1 \circ \bar{h}_q \circ \ldots \circ \bar{h}_1 \circ \bar{t}_{i_{k+l}} \circ \ldots \circ t_{i_1}(\bar{S}) = \mathbf{0}$, because the sequences r_i and $\bar{r}_j$ are accepting sequences. Thus $L(\bar{M}) \supset L(M) \square L(N)$.

Conversely, let $r_1, \ldots r_l$ be an accepting sequence for x. Taking the subsequence of r_i obtained by taking all $r_k \in \bar{R}_M$ in order as they appear in the main sequence, we observe that this must be an accepting sequence in M for some $y \in L(M)$. Those r_i's not in $\bar{R}_M$ must be in $\bar{R}_N$, and we obtain that these elements taken in order from the sequence $\bar{r}_{i'_1}, \ldots, \bar{r}_{i'_k}$ such that $r_{i'_1}, \ldots, r_{i'_k}$ is an accepting sequence of N for some $z \in L(N)$. Similar subsequences of the clearing sequence of $r_1, \ldots, r_l$ can be taken to get clearing sequences $h_{i_1}, \ldots, h_{i_p}, h_{i'_1}, \ldots, h_{i'_q}$ in M and N, respectively, of the two accepting sequences already so obtained. Thus since the sequence $r_1, \ldots, r_l$ is a shuffle of accepting sequences in M and N it follows that $x \in y \square z$ and hence $x \in L(M) \square L(N)$. This concludes the lemma. ●

We have only proved closure properties for ordinary vector machines thus far. However, each of the proofs of the preceding Lemmas (IV.1–4) work equally well for augmented n-tuple machines. We summarize the results thus far in the following theorem:

THEOREM IV.1. *Both $\mathscr{L}_{VM}$ and $\mathscr{L}_{VM+}$ are closed under the operations $\cdot$, $+$, $\square$.*

LEMMA IV.5. *Let $M = (\Sigma, R, S, H)$ be an augmented n-tuple machine. Then there exists M^* an augmented $(n + 1)$-tuple machine, with $L(M^*) = L(M)^*$.*

PROOF. Let $M^* = (\Sigma, R^*, S^*, H^*)$ be defined as follows. Let $S^* = (0, \ldots, 0)$, $H^* = \{\}$. Define $t_s = (\{\bar{p}_1, \ldots, \bar{p}_n, \bar{p}_{n+1}\}, V \cup \{v_{n+1}\})$, where $v_i \in V$ exactly once *iff* $S_i = 1$ for all i, $1 \leq i \leq n$. Define $t_h = (p \cup \{\bar{p}_{n+1}\}, V)$, for each $h = (P, V) \in H$. Define $r_x = (\lambda, t_x)$ and $R_H = \{r_h = (\lambda, t_h) \mid h \in H\}$. Finally let $r_c = (\lambda, t_c)$, where $t_c = (\{p_{n+1}\}, \{\})$. For $r = [\alpha, (P, V) \in R$, let $r^* = (\alpha, (P \cup \{p_{n+1}\}, V \cup \{v_{n+1}\})]$, and then let $R_M = \{r^* \mid r \in R\}$. Now we can define R^* by $R^* = R_M \cup R_H \cup \{r_s, r_c, r_{\text{stop}}\}$.

This completes the definition of M^*.

Let $x \in [L(M)]^*$. Then $x = x_1 x_2 \ldots x_m$, for some $m \geq 0$, with $x_i \in L(M)$. Since $S^* = (0, \ldots, 0)$, certainly $\lambda \in L(M^*)$. So we can assume $m \geq 1$. For each x_i, there exists $r^i_1, \ldots, r^i_{l_i}$ and $h^i_1, \ldots, h^i_{k_i}$, accepting and clearing sequences, respectively, in M for x_i. But then the function $\sigma_i(\mathbf{0}) = t_{h_{k_i}} \circ \ldots \circ t_{h^i_1} \circ t_c \circ t^i_{l_i} \circ \ldots \circ t^i_1 \circ t_s(\mathbf{0}) = \mathbf{0}$. It follows that $\sigma_m \circ \ldots \circ \sigma_1(S^*) = \mathbf{0}$ and we have an accepting sequence for x in M^*. We conclude that $L(M^*) \supset L(M)^*$.

Conversely, let $r^*_1, \ldots, r^*_l$ be an accepting sequence for x in M^*. First we observe that any accepting sequence either accepts λ, the empty string, or else another sequence accepting the same string can be found that begins with r_s. For $r = (\alpha, t)$, $\alpha \neq \lambda$ *iff* $r \in R_M$ and in this case, the p multiset of r contains a p_{n+1}, making it undefined unless preceded by a t_s with no t_c intervening. We note that $\lambda \in L(M)^*$ and then assume that $r^*_1 = r_s$. Now we divide $r^*_1, \ldots, r^*_l$ into $k > 0$ sequences $r^i_1, \ldots, r^i_{l_i}$, $1 \leq i \leq k$, such that $r^i = r_s$ for all i. By examining the value of b_{n+1} we can see that each such sequence has the form $r_s, r^*_1, \ldots, r^*_p, r_c, r_{h_1}, \ldots, r_{h_q}$. Since t_s is only defined on 0, $S^* = 0$, and the sequence is accepting we can conclude that for each i, $1 \leq i \leq k$ $t^i_{l_i} \circ \ldots \circ t^i_1(\mathbf{0}) = t_{h_q} \circ \ldots \circ t_{h_1} \circ t_c \circ t^*_p \circ \ldots \circ t^*_1 \circ t_s(\mathbf{0}) = \mathbf{0}$. It follows that, in M, $h_q \circ \ldots \circ h, \circ t_p \circ \ldots \circ t_1(S) = \mathbf{0}$. Thus $r^i_1, \ldots, r^i_{l_i}$ is an accepting sequence in M for some x_i, for i, $1 \leq i \leq k$. Therefore, $x = x_1 x_2 \ldots x_k \in L(M)^*$. ●

We have proven the following:

THEOREM IV.2. *$\mathscr{L}_{VM+}$ is closed under the Kleene star operation $(*)$.*

IV.3. Simple Shuffle Languages and Containment Relationships

shuffle closure can be used only once

Definition IV.4. A *simple shuffle expression S* is a shuffle expression generated by the following grammar:

$$S \to S + S \mid S \cdot S \mid S \square S \mid S^* \mid C^{[*]} \mid C$$
$$C \to C + C \mid C \cdot C \mid C \square C \mid C^* \mid x \in \Sigma.$$

A shuffle language L is called a *simple shuffle language* if $\exists S$, a simple shuffle expression, with $L(S) = L$. Let $\mathscr{L}_{ss}$ denote the class of all simple shuffle languages.

LEMMA IV.6. *Let $L \in \mathscr{L}_R$. Then $\exists$ a vector machine M with $L(M) = L^{[*]}$.*

PROOF. Let A be a dfa recognizing L. Define $M_A = (\Sigma, R_A, S_A, H_A)$ as the n-tuple machine given by Lemma IV.1 recognizing L. Define $M = (\Sigma, R, S, H)$ as an n-tuple machine in the following way:

Let $S = 0$,

$H = H_A$, and

$R = R_A \cup \{r_s\}$, where $r_s = (\lambda, t_s)$ and

$t_s = (\{\}, \{v_1\})$.

Now let $x \in L^{[*]}$, then $x \in x_1\Box\ldots\Box x_m$, where $x_i \in L = L(M_A)$. Let $r_1^i, \ldots, r_{l_i}^i$ be an accepting sequence for x_i in M_A. Then for each $x \in x_1\Box\ldots\Box x_m$, $\exists \sigma \in [(r_s)^m, (r_1^1, \ldots r_{l_1}^1)\Box\ldots\Box(r_1^m, \ldots, r_{l_m}^m)$ such that σ is an accepting sequence of x in M. For any such accepting function $\tau = \tau_1 \circ t_p^i \circ t_q^j \circ \tau_z$, if $i \neq j$, then $\tau_1 \circ t_p^i \circ t_q^j \circ \tau_z = \tau_1 \circ t_q^j \circ t_p^i \circ \tau_z$. Hence $x \in L(M)$.

Conversely, let $r_1 \ldots r_m$ be an accepting sequence of x in M. If $m = 0$, that is $x = \lambda$ and $\lambda \in L^{[*]}$. Otherwise $r_1 = r_s$ since only t_s is defined on 0. Now choose $i_1, \ldots, i_l$ where $j < k$ implies that $i_j < i_k$ such that $t_{i_j} \neq t_s$ and l is as large as possible. Then $h_k \circ t_{i_l} \circ \ldots \circ t_{i_1}(S_A) = \mathbf{0}$ for some k, (otherwise maximality of l is violated), and $\alpha_{i_1} \ldots \alpha_{i_l} = x \in L$. Furthermore if $r_1' \ldots r_{l'}'$ is the subsequence obtained by removing the r_{i_j}'s, $r_1', \ldots r_{l'}'$ is an accepting sequence in M.

Let us assume this is not the case. If $\sigma = t_k \circ \ldots \circ t_1$, let $N_{p_i}(\sigma)$ be the number of times transformations at the form $(\{P_i\}, \{v_k\})$ $(1 \le k \le n)$ appear in α and let $N_{v_i}(\sigma)$ be defined analogously for v_i. Note that $(\{p_i\}, \{v_n\}) \circ \sigma$ $(\mathbf{0})$ is defined *iff* $N_{p_i}(\sigma) < N_{v_i}(\sigma)$. Now suppose $t_{l'}' \circ \ldots \circ t'$ $(\mathbf{0})$ is not defined. Then $\exists$ $\sigma = t_{k-1}' \circ \ldots \circ t_1'$ and some i $1 \le i \le n$ such that $N_{p_i}(\sigma) \ge N_{v_i}(\sigma)$. Also observe that for $\tau = t_{i_l} \circ \ldots t_{i_1} \circ t_s$, $N_{p_j}(\tau) \le N_{v_j}(\tau) \le N_{p_j}(\tau) + 1$, for all j. If $\gamma \in \sigma\Box\tau$ is the subsequence of the original sequence, we have two possiblities: (1) If $N_{p_i}(\tau) = N_{v_i}(\tau)$, then $N_{p_i}(\sigma) + N_{p_i}(\tau) = N_{p_i}(\gamma) \ge N_{v_i}(\gamma) = N_{v_i}(\sigma) + N_{v_i}(\tau)$ and t_k' was blocked in the original string; (2) If $N_{p_i}(\tau) = N_{v_i}(\tau) - 1$, then $t_{i_{l'}}$, the last transformation in τ is of the form $(\{p_k\} \{v_i\})$ for some k, and since $N_{p_i}(\gamma) < N_{v_i}(\gamma)$, because γ is defined, t_{i_l} occurs before t_k' in the original sequence, contradicting maximality of l.

Since $\sigma_o\, \mathbf{0} = t_{l'}' \circ \ldots \circ t'$ $(\mathbf{0})$ must then be defined, it must be that no clearing sequence exists. If $F = \{i \mid h = (\{p_i\}, \{\}) \in H\}$, then $N_{p_i}(\sigma_o) = N_{v_i}(\sigma_o) - 1$ for some $i \notin F$. But if τ is an accepting sequence, it must be that $N_{p_j}(\tau) = N_{v_j}(\tau)$ for all $j \notin F$. Thus the original sequence γ must have $N_{p_i}(\gamma) = N_{v_i}(\gamma)$. This means that for $\tau = t_{i_l} \circ \ldots \circ t_i \circ t_s$

$$
\begin{aligned}
N_{v_i}(\tau) &= N_{v_i}(\gamma) - N_{v_i}(\sigma_o) \\
&= N_{p_i}(\gamma) - N_{v_i}(\sigma_o) \\
&= N_{p_i}(\gamma) - N_{p_i}(\sigma_o) - 1 \\
&= N_{p_i}(\gamma) - N_{p_i}(\sigma_o) \\
&= N_{p_i}(\tau)
\end{aligned}
$$

which contradicts the fact that $\tau(0)$ is defined.

We are forced to conclude that $r_1', \ldots r_{l'}'$ is an accepting sequence in M. Then $x \in y\Box y'$ where $y \in L$ and $y' \in L(M)$ and furthermore $y \neq \lambda$. We repeat the process on y', reducing $l(y')$ on every iteration, until $y' = \lambda$ and we get $x \in y_1\Box y_2 \Box \ldots \Box y_m$, $y_i \in L$. ●

This proves the lemma. Note that the construction can be carried out for an augmented vector machine, also.

This lemma allows us to prove the following theorem about $\mathscr{L}_{ss}$.

THEOREM IV.3. $\mathscr{L}_{ss} \subseteq \mathscr{L}_{VM+}$.

PROOF. We recall that regular languages are closed under $\cdot$, $+$, $\Box$, and $*$. Then by Lemma IV.6, we know that we can construct a vector machine to accept $L(c^{[*]})$ in Definition IV.4, since c must be regular. Finally $\mathscr{L}_{VM+}$ is closed under t, $\cdot$, $\Box$, and $*$ by Theorems IV.1. and IV.2. ●

$\mathscr{L}_{ss}$ is the set of all shuffle languages that have an expression that does not contain a nesting of the operation $^{[*]}$, i.e., a form like $(L_1{}^{[*]} o L_2)^{[*]}$, where o can be $\cdot$, $+$, or $\Box$. A similar containment relationship has been shown by Araki et al. in [1] for flow languages.

We now demonstrate a few results which deny equality of certain classes. Although all of the results in Sec. (IV.3) were independently discovered by the author, Lemma IV.8 was known previously to R. W. Ritchie, and Lemma IV.7 to W. L. Ruzzo.

LEMMA IV.7. *If a shuffle language L has an alphabet Σ with* ord$(\Sigma) > 1$, *then* $\exists \alpha, \beta \in \Sigma$, *with* $\alpha \neq \beta$, *such that* $(L)^{[*]}$ *contains strings x and y such that* $x = z_1\alpha z_2\beta z_3$ *and* $y = z_4\beta z_5\alpha z_6$, *for some* $z_i \in \Sigma^*$.

PROOF. Suppose L has a string x with two different symbols α, β. Then if we let $y = xx \in (L)^{[*]}$, clearly x and y meet the conditions above. For certainly $x = z_1\alpha z_2\beta z_3$ but $y = z_4\beta z_5\alpha z_6$ where $z_4 = z_1\alpha z_2$, $z_5 = z_3 z_1$ and $z_6 = z_2\beta z_3$.

But now if no string with two different symbols is in L, then there must be $\alpha, \beta \in \Sigma$ with α^m and β^n in L for some m and $n > 0$. Hence $x = \alpha^m\beta^n$ and $y = \beta^n\alpha^m$ satisfy the lemma.✓●

We use this lemma to prove the following result.

LEMMA IV.8. $\{a^nb^nc^n \mid n \ge o\} \notin \mathscr{L}_s$.

PROOF. Let L_1 denote the language $\{a^nb^nc^n \mid n \ge o\}$. It is well known that L_1 is not regular. Hence a shuffle expression E for L must contain a subexpression of the form $(L')^{[*]}$, for some shuffle language L'. If L' uses only one symbol, it follows that $(L')^{[*]} = L$, making L a regular language. We conclude that Σ', the alphabet of Σ', has ord $(\Sigma') > 1$.

By Lemma IV.7 we know that we can find $\alpha, \beta \in \Sigma' \subset \Sigma$ such that α and β occur in either order in strings L', hence in L_1. But this is impossible for strings in L_1, so we conclude that E cannot determine L_1. ●

THEOREM IV.4. $\mathscr{L}_s \neq \mathscr{L}_{VM}$.

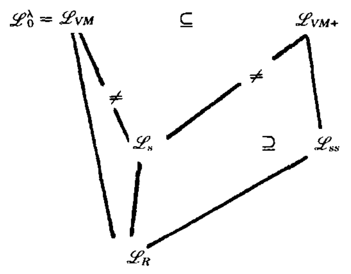
PROOF. We know that $L_1 = \{a^nb^nc^n \mid n \ge 0\} \in \mathscr{L}_{VM}$ by the construction of Lemma I.1. However, Lemma IV.8

shows that $L_1 \notin \mathcal{L}_s$. ●

THEOREM IV.5. $\mathcal{L}_R \neq \mathcal{L}_{ss}$.

PROOF. Let $E_1 = (abc)^{[*]}$ and $E_2 = a^*b^*c^*$. Then $L(E_1) \in \mathcal{L}_{ss}$ and $L(E_2) \in \mathcal{L}_R \subseteq \mathcal{L}_{ss}$. But $L(E_1) \cap L(E_2) = \{a^nb^nc^n \mid n \geq 0\} \notin \mathcal{L}_R$. Since the regular sets are closed under intersection we are forced to conclude that $L(E_1) \notin \mathcal{L}_R$, and $\mathcal{L}_R$ is properly contained in $\mathcal{L}_{ss}$.

We summarize the results of this section with the following lattice diagram.



Note: An unbroken line indicates proper containment of the lower class in the higher, while a line broken by $\neq$ indicates that containment is not known but equality is known not to be true.

Perhaps the most important unsolved problem shown here is whether $\mathcal{L}_{ss} = \mathcal{L}_s$. ●

V. Grammars for Shuffle Languages

The chief goal of this section is proving that all shuffle languages are context-sensitive. We do this by showing that the context-sensitive languages are closed under the operations $+$, $\cdot$, $*$, □, $^{[*]}$.

THEOREM V.1. *The context-sensitive languages are closed under the operations* $+$, $\cdot$, $*$.

PROOF. This is a well-known result which we will not ~~state~~ PROVE here. For a complete proof, the reader should consult Hopcroft and Ullman [4]. ●

THEOREM V.2. *The context-sensitive languages are closed under the operation* □.

PROOF. Let $G_1 = (V_{N_1}, V_{T_1}, S_1, P_1)$ and $G_2 = (V_{N_2}, V_{T_2}, S_2, P_2)$ be context-sensitive grammars. Construct the grammar $G = (V_N, V_T, S, P)$ in the following way: Let $V_T = V_{T_1} \cup V_{T_2}$. For each $\alpha_i \in V_T$ define a new, unique nonterminal A_i. Then $V_N = V_{N_1} \cup V_{N_2} \cup \{A_i \mid \alpha_i \in V_T\} \cup \{M, S\}$. Finally, let $p \in P$ *iff*

(1) $p' \in P_1 \cup P_2$, and $p = p'$ with each terminal α_i replaced by A_i;
(2) $p = A_iM \rightarrow M\alpha_i$, $\alpha_i \in V_t$;
(3) $p = \alpha_iM \rightarrow M\alpha_i$, $\alpha_i \in V_T$;
(4) $p = \alpha_iA_j \rightarrow A_j\alpha_i$, for i, j with $\alpha_i, \alpha_j \in V_t$; or
(5) $p \in \{S \rightarrow S_1MS_2M, M \rightarrow \lambda\}$.

Now clearly $L(G) \supseteq L(G_1)$ □ $L(G_2)$, since we can use productions defined in (1) above to derive $A_{i_1} \ldots A_{i_m}MA_{j_1} \ldots A_{j_n}M$, where α_{i_k} and α_{j_l} are in V_T, and $x_{i_1} \ldots x_{i_m}$, and $\alpha_{j_1} \ldots \alpha_{j_n}$ are in $L(G_1)$ and $L(G_2)$, respectively. The first M is then moved to the extreme left using the rules $A_iM \rightarrow M\alpha_i$. The A_j's may then be "shuffled" into the terminals by means at the rules $\alpha_iA_j \rightarrow A_j\alpha_i$. Finally M is moved to the far left changing all remaining nonterminals to terminals and both M's are disposed of by $M \rightarrow \lambda$. To show that this is the only possible derivation, we observe first that no string of the form $x'\alpha_jy'A_iz'$ can be derived from a string of the form $xA_iy\alpha_jz$ where $A_i \notin z$, and $\alpha_j \notin x$. That is to say, terminals cannot be moved left past nonterminals; only the reverse is possible. Further note that for any substring MxM, x is a string for which all nonterminals were derived from S_2 and all terminals from S_1. The order of the nonterminals in x cannot be changed nor can that of the terminals, they can only be "shuffled" into each other. Let $xMyMz$ be the entire string at some point in a derivation, x is entirely nonterminals, z is entirely terminals, hence their orders cannot be changed. If y contains nonterminals, then using $M \rightarrow \lambda$ to get $xMyz$ will never result in a terminal string; if y is all terminals, this is permissible but the order of yz will not change. If $xyMz$ is produced, the order in y might be changed but x will not be shuffled with y, since nonterminals cannot be moved right. We must conclude that the derivation described above is the only kind possible, thus we have $L(G) = L(G_1)$ □ $L(G_2)$.

However, G is not, strictly speaking, a context-sensitive grammar, since we have the rule $M \rightarrow \lambda$. If we make a new grammar G' by making M a terminal and removing the production $M \rightarrow \lambda$, then G' is certainly context sensitive. Now define a mapping σ: $V_T \cup \{M\} \rightarrow V_T$ by

$$\sigma(\alpha) = \begin{cases} \alpha & \textit{iff } \alpha \in V_T \\ \lambda & \text{if } \alpha = M \end{cases}$$

Now σ can be extended to $x \in V_T^*$ by the rule $\sigma(x\alpha) = \sigma(x) \cdot \sigma(\alpha)$. Now σ has the property that, when applied to strings in $L(G')$, no more than two symbols are mapped to λ. We call σ a 2-limited erasing of $L(G')$. It is well known that context-sensitive languages are closed under any k limited erasing (see [4]). Therefore since $\sigma[L(G')] = L(G)$, we conclude that $L(G) = L(G_1)$ □ $L(G_2)$ is indeed context sensitive. ●

In order to show closure under the $^{[*]}$ operation we first demonstrate a non-context-sensitive grammar G for $L(G_1)^{[*]}$, given G_1, and then give a context-sensitive grammar G' derived from G with $L(G) = L(G')$.

Definition V.1. For $G_1 = (V_{N_1}, V_{T_1}, S_1, P_1)$ a context-sensitive grammar, define $G = (V_N, V_T, S, P)$ as follows: Let $V_T = V_{T_1}$. With each $x_i \in V_T$, associate a new, unique nonterminal A_i. Then $V_N = V_{N_1} - \{A_i \mid \alpha_i \in V_T\} \cup \{M, S\}$. Let $p \in P$ *iff*

(1) $p' \in P_1$ where p is p' with A_i substituted, for each $\alpha_i \in V_t$;

(2) $p = A_iM \rightarrow M\alpha_i$ or $p = \alpha_iM \rightarrow M\alpha_i$, for $\alpha_i \in V_T$;
(3) $p = \alpha_jA_i \rightarrow A_i\alpha_j$, for i_j such that $\alpha_i, \alpha_j \in V_T$; or
(4) $p \in \{S \rightarrow \lambda, S \rightarrow S', S' \rightarrow S_1MS', S' \rightarrow S_1M, M \rightarrow \lambda\}$.

This completes the definition of G.

LEMMA V.1. $L(G) = L(G_1)^{[*]}$.
PROOF. The proof of this is entirely analogous to that in Theorem V.1 showing that the construction there actually produces $L(G_1) \square L(G_2)$. ●

We now define a context-sensitive G' with $L(G') = L(G)$.

Definition V.2. Let G be a grammar as defined in Definition V.1. Then define $G' = (V'_N, V'_T, S', H')$ as follows. Let $V'_T = V_T$. With each $Q \in V_N$ and $\alpha \in V_T$ associate new, unique nonterminals Q_M and α_M, respectively. Let $S = S'$.

(1) $p = S_1 \rightarrow xQ_M$, where $(S_1 \rightarrow xQ) \in P$, $Q \in V_N \cup V_T$ and x is some string;
(2) $p = xQ_M \rightarrow yR_M$, where $(xQ \rightarrow yR) \in P$, and $Q, R \in V_N \cup V_t$;
(3) $p \in P$ and p does not involve M or S_1;
(4) $p = LA_{i,M} \rightarrow L_M\alpha_i$, where $L \in V_N \cup V_t - \{M\}$, $\alpha_i \in V_T$;
(5) $p = L\alpha_{i,M} \rightarrow L_M\alpha_i$, where $L \in V_N \cup V_T - \{M\}$, $\alpha_i \in V_T$;
(6) $p = \alpha_{i,M} \rightarrow \alpha_i$ or $p = A_{i,M} \rightarrow \alpha_i$, for $\alpha_i \in V_T$; or
(7) $p \in \{S \rightarrow \lambda, S \rightarrow S', S' \rightarrow S_1, S' \rightarrow S_1S'\}$.

This completely defines G'.

LEMMA V.2. $L(G') = L(G)$.
PROOF. Loosely speaking, the subscript M means "M is immediately to the right of symbol". We note that $xPQMy \underset{G}{\rightarrow} xPMQ'y$ *iff* $xPQ_My \underset{G'}{\overset{*}{\rightarrow}} xP_MQ'y$-, where $Q, P \in V_T \cup V_N$ and $Q' = \alpha_i$ if $Q = A_i$ and $Q' = Q$ if $Q \in V_T$. Also the productions given in (1) and (2) of Def. V.2 above give $S \underset{G'}{\overset{*}{\rightarrow}} x_1A_{1,M}x_2A_{2,M} \ldots x_nA_{n,M}$ *iff* $S \underset{G}{\overset{*}{\rightarrow}} x'_1Mx'_2M \ldots x'_nM$, when $x'_i = x_iA_i$. We have assumed that G_1 was a grammar such that S_1 does not appear on the right side of any $p \in P_1$. All context-sensitive languages have such a grammar. This assumption is necessary to avoid trouble with parts (1) and (3) above. Keeping the above remarks in mind it is easy to see that $L(G') = L(G)$. Therefore we have proved the following:

THEOREM V.3. *The context-sensitive languages are closed under* $^{[*]}$.

PROOF. From Lemmas V.1 and V.2, we have that $L(G_1)^{[*]} = L(G) = L(G')$. But we notice that G' is a context-sensitive grammar. ●

We also could have proven $L(G)$ context sensitive by considerations of linear bounded automata since $L(G)$ is a linear-bounded erasing of a context-sensitive language. We conclude with a theorem summarizing the results of this section:

THEOREM V.4. *If* $L \in \mathscr{L}_s$, *then L is context sensitive.*

PROOF. If $L \in \mathscr{L}_s$, then $L = L(E)$ for some shuffle expression E. Certainly the language $\{\alpha\}$ for $\alpha \in \Sigma$ is context sensitive, and the context-sensitive languages are closed under $+, \cdot, *, \square, ^{[*]}$. Therefore $L = L(E)$ is context sensitive. ●

Acknowledgments. This work was done as a senior research project while the author was a senior at the University of Washington. Many thanks are due to R. W. Ritchie for his suggestions on this manuscript and all his invaluable help to me.

Received 9/79

References
1. Araki, T., Kagimasa, T., and Tokura, N. Relations of flow languages to Petri Net languages. Programming Languages Group Memo No. 79-01, Dept of Infor. and Comp. Sci., Osaka, Japan, Feb. 1979.
2. Ginsburg, S. *The Mathematical Theory of Context Free Languages.* McGraw-Hill, New York, 1966.
3. Hack, M. Petri net languages. Computation Structures Group Memo 124, Project MAC, M.I.T., Cambridge, MA June, 1975.
4. Hopcroft, J. and Ullman, J. *Formal Languages and Their Relation to Automata.* Addison-Wesley, Reading, MA. 1969.
5. Shaw, A. C., Software descriptions with flow expressions. *IEEE Trans. Software Eng. SE-4*, 3, (May 1978) 243–254.