TWO CHARACTERIZATIONS OF THE CONTEXT-SENSITIVE LANGUAGES

By

Michael J. Fischer

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania
September, 1969

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-67-C-0058) and is monitored by the Air Force Office of Scientific Research. This document has been approved for public release and sale; its distribution is unlimited.

TWO CHARACTERIZATIONS OF THE CONTEXT-SENSITIVE LANGUAGES

Michael J. Fischer
Carnegie-Mellon University

SUMMARY

An n-dimensional bug-automation is generalization of a finite state acceptor to n-dimensions. With each bug B, we associate the language L(B) which is the set of top rows of the n-dimensional rectangular arrays accepted by B. One-dimensional bugs define trivially the regular sets. Two-dimensional bugs define precisely the context-sensitive languages, while bugs of dimension 3 or greater define all the recursively enumerable sets.

We consider also finite state acceptors with n two-way non-writing input tapes. For each such machine M, let domain (M) be the set of all strings which are the first component of some n-tuple of tapes accepted by M. For any $n \geq 1$, the domains of n-tape two-way finite state acceptors are precisely the same as the languages definable by n-dimensional bugs, so as a corollary, the domains of two-tape two-way finite state acceptors are precisely the context-sensitive languages.

1. Introduction

A bug-automation is defined to be a non-writing, finite state acceptor which operates on an n-dimensional, rectangular array of symbols called a scene. In a single step, the bug may change state and move its read head one square in any direction in the scene. The move depends only on the current state of the bug and on the symbol under scan. We consider non-deterministic bugs, so in general more than one move is possible at any step of the computation. A scene is accepted if, when the bug is started in the upper left-hand corner in the designated start state, there exists a sequence of moves which ends by the bug falling off the scene in an accepting state; otherwise it is rejected.

Blum and Hewitt have studied two-dimensional finite state acceptors as pattern recognition devices.[1] Their models are similar to our bugs, although their halting conventions differ, and their machines can tell when they are at the edge of the scene while ours cannot. We are not concerned here with the recognizable scenes but rather with the languages defined by their projections, so for our purposes, these differences in convention make no difference, and our theorems are true of their model as well.

With each bug B, we associate a language L(B) defined to be the set of all top rows of scenes accepted by B. In the case of $n > 2$, by "top row" we mean "the top row of the top plane of the top solid...".

Letting n vary, we get a coarse classification of languages. One-dimensional bugs trivially define exactly the regular sets, and for $n \geq 3$, n-dimensional bugs can define all the recursively enumerable sets. Our main theorem is that the languages definable by two-dimensional bugs are precisely the context-sensitive languages.

There is a close relationship between n-dimensional bugs and finite state acceptors with n non-writing two-way tapes. Define the domain of a relation $R \subseteq \Sigma^{*n}$ to be the set of all strings $x \in \Sigma^*$ such that there exist $y_2,\ldots,y_n \in \Sigma^*$ so that $\langle x,y_2,\ldots,y_n\rangle \in R$. We show for each $n \geq 1$ that the domains of the relations defined by n-tape, two-way finite state acceptors are the same as the languages definable by n-dimensional bugs. Hence, as a corollary, the domains of two-tape two-way finite state acceptors are precisely the context-sensitive languages.

2. Bug-Automata

Definition 2.1: An n-dimensional scene over Σ is an n-dimensional rectangular array of elements of Σ. The set of all n-dimensional scenes over Σ is denoted by $\Sigma^{(n)}$.

Given a scene $A \in \Sigma^{(n)}$, we let $\ell_i(A)$ be the number of planes in the i^{th} coordinate. Thus, for 2-dimensional scenes, $\ell_1(A)$ is the number of rows and $\ell_2(A)$ is the number of columns.

If $1 \leq i_k \leq \ell_i(A)$ for $k=1,\ldots,n$, we let $A_{i_1,\ldots,i_n}$ denote the symbol in A with coordinates $i_1,\ldots,i_n$. The sequence of indices $i_1,\ldots,i_n$ will sometimes be abbreviated by $\vec{i}$.

Definition 2.2: A (non-deterministic) bug-automation is a 6-tuple $(n,K,\Sigma,\delta,q_0,F)$, where n is the dimensionality of the bug; K is a finite set of states; Σ is a finite set of input symbols; $\delta: K \times \Sigma \to 2^{(K \times S^n)}$ is the control function, where $S = \{-1,0,+1\}$ is the set of shifting operations; $q_0 \in K$ is the start state; and $F \subseteq K$ is a set of accepting states.

An instantaneous description is a pair $\langle q,\vec{i}\rangle$, where $q \in K$ and $\vec{i} \in N^n$ is a position vector.*

Given a bug B and an input $A \in \Sigma^{(n)}$, we say that $\langle q,\vec{i}\rangle \vdash \langle q',\vec{i}'\rangle$ if (i) $1 \leq i_k \leq \ell_i(A)$ for $k = 1,\ldots,n$; and (ii) $\langle q',\vec{s}\rangle \in \delta(q,A_{\vec{i}})$ for some $\vec{s}$ such that $\vec{i}' = \vec{i} + \vec{s}$, where "+" denotes usual componentwise vector addition.

*N denotes the set of natural numbers, and N^n is the set of all n-tuples of natural numbers.

We let $\vdash^*$ be the reflexive, transitive closure of the relation $\vdash$.*

A scene $A \in \Sigma^{(n)}$ is accepted by a bug $B = (n,K,\Sigma,\delta,q_0,F)$ in case there exists an instantaneous description $\langle q,\vec{i}\rangle$ such that

$$\langle q_0,\langle 1,1,\ldots,1\rangle\rangle \vdash^* \langle q,\vec{i}\rangle$$

for some $q \in F$ and some position vector $\vec{i}$ denoting a position not on the scene, i.e. for some k, $1 \le k \le n$, $i_k = 0$ or $i_k = \ell_k(A) + 1$. The set of all scenes in $\Sigma^{(n)}$ accepted by B is denoted by $T(B)$.

Definition 2.3: With each bug $B = (n,K,\Sigma,\delta,q_0,F)$, we associate a language $L(B) \subseteq \Sigma^*$ by taking the top row of each scene in $T(B)$. Formally,

$$L(B) = \{w \in \Sigma^* \mid \text{there exists an } A \in T(B) \text{ such that } \ell(w) = \ell_1(A) \text{ and } w_k = A_{1,1,\ldots,1,k} \text{ for all } k, 1 \le k \le \ell_1(A)\}.$$

A language $L \subseteq \Sigma^*$ is said to be n-bug-definable if there exists an alphabet $\Sigma' \supseteq \Sigma$ and a bug $B = (n,K,\Sigma',\delta,q_0,F)$ such that $L = L(B)$.

We remark that a one-dimensional bug may be regarded as an ordinary two-way finite state machine without endmarkers, and hence, by the Shepardson construction[5], we have the theorem:

Theorem 2.4: The one-bug-definable languages are precisely the regular sets.

We remark also that for our purposes, the lack of end-markers and border symbols is of no concern, for they do not change the class of definable languages, even though they do affect the class of scenes accepted.

Lemma 2.5: A language $L \subseteq \Sigma^*$ is n-bug-definable iff the language $\vdash L \dashv$ is n-bug-definable, where $\vdash$ and $\dashv$ are new symbols not in Σ.

Proof: We prove the theorem for the case of n=2 and leave to the reader the generalization to larger n.

Suppose $B = (2,K,\Sigma,\delta,q_0,F)$ is a bug defining L. We construct a new bug B' with input alphabet $\Sigma' = \Sigma \cup \{*,\vdash,\dashv\}$, where $*$, $\vdash$, and $\dashv$ are new symbols not in Σ. B' is defined to follow the algorithm:

(1) Check to see if the input scene is of the form of figure 1 with the a_j's and the $b_{i,j}$'s in Σ.

(2) Check to see that the row of a's is the same as the top row of b's, i.e. that $a_j = b_{1,j}$ for $j = 1,\ldots,m$.

(3) Move to $b_{1,1}$ and begin simulating B.

(4) If, during the simulation, an asterisk is ever scanned and the simulated state of B is in F, then accept if there is no symbol to the right of the symbol "$\dashv$".

Clearly, B' requires only a finite number of states to perform this algorithm, and $L(B') = \vdash L(B) \dashv = \vdash L \dashv$.

Conversely, suppose $\vdash L \dashv$ is definable by a bug B. We construct a new bug B' which simulates B on a "compressed" scene, that is, near the edges three symbols are encoded into one (see figure 2). Given an input A over the expanded alphabet, B' follows the instructions:

(1) Check to see if A is of the form of figure 2, where the a_j's are in Σ and the $b_{i,j}$'s are all in Σ except for $b_{1,1}$ and $b_{1,m}$.

(2) Check to see that $b_{1,1} = \vdash$ and $b_{1,m} = \dashv$.

(3) Check to see that $a_j = b_{1,j}$ for all j, $1 < j < m$.

(4) Move to $b_{1,1}$ and begin simulating B within the subrectangle bordered by asterisks. Treat the triples as three separate symbols in the simulation.

(5) If an asterisk is ever scanned and the simulated state of B is final, then move to a_{m-1} and accept if there are no more symbols to its right.

Clearly, the above algorithm requires only finitely many states, and $L(B') = L$. □

3. Languages Definable by 2-dimensional Bugs

In this section, we consider 2-dimensional bugs and show that the languages which they define are precisely the context-sensitive languages.

Theorem 3.1: Every context-sensitive language is 2-bug-definable.

Proof: Let L be a context-sensitive language generated by the context-sensitive

* $\vdash$ is defined inductively by $\langle q,\vec{i}\rangle \vdash^* \langle q',\vec{i}'\rangle$ if $\langle q,\vec{i}\rangle = \langle q',\vec{i}'\rangle$ or if $\langle q,\vec{i}\rangle \vdash^* \langle q'',\vec{i}''\rangle$ and $\langle q'',\vec{i}''\rangle \vdash \langle q',\vec{i}'\rangle$ for some instantaneous description $\langle q'',\vec{i}''\rangle$.

grammar $G = (V_N, V_T, P, S)$, and suppose $w \in L^*$. We define a 2-dimensional bug B which tests if the scene given it represents a derivation (when read from bottom to top) of the context-sensitive grammar G (see example 3.2). B follows the instructions:

(1) Check the input to see if it is of the form of figure 3, where the $a_{i,j}$'s are in $V_N \cup V_T \cup \{\#\}$.

(2) For each $i, 1 \le i \le n$, let $\alpha_i = a_{i,1} \cdots a_{i,m}$ be the contents of the i^{th} row (exclusive of the asterisks). Check to see that each α_i is of the form $\beta_i \cdot \gamma_i$ for some $\beta_i \in (V_N \cup V_T)^*$ and $\gamma_i \in \{\#\}^*$.

(3) Check that $\alpha_1 \in V_T^*$.

(4) Check that $\beta_n = S$.

(5) For each i, $1 \le i < n$, see that $\beta_{i+1} \underset{G}{\Longrightarrow} \beta_i$.

(6) When all of the above conditions have been verified, then accept if there are no symbols to the right of the "$\dashv$".

Clearly, B requires only finitely many states to perform the above checks. Hence, B accepts all and only those scenes which "encode" a derivation of the grammar G, so $L(B) = \vdash L \dashv$. By lemma 2.5, L itself is also 2-bug-definable. □

Example 3.2: Figure 4 gives the representation of a derivation of the string $a^3b^3c^3$ using the context-sensitive grammar $G = (V_N, V_T, P, S)$, where $V_N = \{S, A, B, C\}$, $V_T = \{a, b, c\}$, and P has the productions:

$$S \to aSBC$$
$$S \to abC$$
$$CB \to BC$$
$$bB \to bb$$
$$bC \to bc$$
$$cC \to cc\ .$$

---

*A context-sensitive grammar is a quadruple $G = (V_N, V_T, P, S)$ where V_N is a finite set of non-terminal symbols, V_T is a finite set of terminal symbols, $P \subseteq (V_N \cup V_T)^* V_N (V_N \cup V_T)^* \times (V_N \cup V_T)^+$ is a set of productions with the property that if $\langle \alpha, \beta \rangle \in P$, then $\ell(\alpha) \le \ell(\beta)$, and $S \in V_N$ is the start symbol. Define $\sigma \underset{G}{=} \tau$ if $\sigma = \sigma_1 \alpha \sigma_2$ and $\tau = \sigma_1 \beta \sigma_2$ are strings in $(V_N \cup V_T)^*$ and $\langle \alpha, \beta \rangle \in P$. Let $\underset{G}{\overset{*}{\Longrightarrow}}$ be the reflexive, transitive closure of $\underset{G}{\Longrightarrow}$. The language generated by G is $L(G) = \{w \in V_T^* \mid S \underset{G}{\overset{*}{\Longrightarrow}} w\}$. See [2] for more details.

Before proving the converse to theorem 3.1, we observe that it is not necessary for the bug to be able to move diagonally; any diagonal move can be replaced by a horizontal move followed by a vertical move. Moreover, it is never necessary to stay on the same square without moving. Hence, we have (generalizing to n-dimensions):

Lemma 3.3: Let $B = (n, K, \Sigma, \delta, q_0, F)$ be a bug-automaton. Then there exists a bug $B' = (n, K', \Sigma, \delta', q_0', F')$ such that $T(B) = T(B')$ and for each $q \in K'$ and $a \in \Sigma$, $\langle q', \vec{s} \rangle \in \delta'(q, a)$ implies that $\vec{s}$ has exactly one non-zero component.

We now state the most difficult theorem of this paper.

Theorem 3.4: Every 2-bug-definable language is context-sensitive.

Proof: Let $B = (2, K, \Sigma, \delta, q_0, F)$ be a bug. By lemma 3.3, we may assume that on every step of the computation, B moves exactly one square up, down, left, or right. We let $S' = \{\langle -1, 0\rangle, \langle 1, 0\rangle, \langle 0, -1\rangle, \langle 0, 1\rangle\}$ be the set of these four shift instructions, so we are assuming that $\delta: K \times \Sigma \to 2^{K \times S'}$.

Given a scene accepted by B, we associate with it a set of scenes over an expanded alphabet which "describes", in a sense to be made precise later, the possible accepting computations of the bug on the original scene. This set will be easy to recognize: it will be the set of all scenes over a subset of the expanded alphabet in which each square satisfies certain local conditions.

We will then have that a string w is in L(B) iff there exists a description scene whose top row describes w. But since description scenes are defined by local conditions, we can find a non-deterministic linear bounded automaton (LBA) which "guesses" the description scene a row at a time, remembering only the previous row in order to insure that the local conditions are satisfied. The input w will be accepted if and only if the LBA is able to complete the description scene starting from w.

For clarity of exposition, we will call an element of the description alphabet a tile. Let $C = K \times S'$ be the set of crossing transitions. A member of C tells the state and the direction from which a given square is entered or left. If $c = \langle q, \vec{s} \rangle \in C$, let state (c) = q and shift (c) $= \vec{s}$.

Let $D = 2^C \times 2^C$ be the set of description tiles. If $d = \langle c_1, c_2 \rangle \in D$, let entry (d) $= c_1$ and exit (d) $= c_2$. A description tile is a guess of the behavior of the bug on the corresponding square of the scene being described. $\langle q, \vec{s} \rangle \in$ entry (d) means that the bug may enter the

square by performing shift $\vec{s}$ (from one of the four surrounding squares) and then be in state q; $\langle q,\vec{s}\rangle \in$ exit (d) means that the bug may leave the square by performing shift $\vec{s}$ and entering state q.

A symbol $a\in\Sigma$ admits a description $d\in D$ if there is a one-to-one correspondence m between the entry and exit sets of d such that for every $c\in$ entry (d), m(c) is a possible behavior of the bug on input a, that is, $m(c)\in\delta(\text{state}(c), a)$. We may think of the function m as defining a set of paths through the square which the bug may take in the course of accepting a scene: when the bug enters via transition c, it may leave via m(c). Since m is a one-one correspondence, the paths it defines do not merge.

Now, we wish to assemble tiles to form all and only those descriptions of accepting computations. The basic requirement is that whenever two tiles are placed adjacent, the exit set of one along the common edge must match the entry set of the other and vice versa. This insures that no paths disappear or spring up from nowhere at the boundary. In other words, every path entering or leaving a square connects to a path of its neighbor. In addition, the tile used in the upper left hand corner must be the only tile to originate a path (by pretending the bug enters it from the left in the start state), and every tile used on an edge of the scene must be such that any path which drops off the scene does so in an accepting state.

Any description scene with these properties must describe an accepting computation, for the only place a path can begin is in the upper left hand corner; the only place a path can end is at the edge of the scene in an accepting state and finally, no paths can merge in between, eliminating the possibility of a path ending in a loop. Conversely, from an accepting path, we can clearly find such a description scene.

We now formalize these ideas. For each $\vec{s}\in S'$ and $C'\subseteq C$, let $K_{\vec{s}}(C') = \{q\in K \mid \langle q,\vec{s}\rangle\in C'\}$.

H and V are the horizontal and vertical adjacenty conditions, respectively. Let $d_1,d_2\in D$. $\langle d_1,d_2\rangle\in H$ (d_2 may be placed to the right of d_1) iff (i) $K_{\langle 0,-1\rangle}(\text{entry}(d_1)) = K_{\langle 0,-1\rangle}(\text{exit}(d_2))$ and (ii) $K_{\langle 0,1\rangle}(\text{exit}(d_1)) = K_{\langle 0,1\rangle}(\text{entry}(d_2))$. Similarly, $\langle d_1,d_2\rangle\in V$ (d_2 may be placed below d_1) iff

(i) $K_{\langle -1,0\rangle}(\text{entry}(d_1)) = K_{\langle -1,0\rangle}(\text{exit}(d_2))$ and (ii) $K_{\langle 1,0\rangle}(\text{exit}(d_1)) = K_{\langle 1,0\rangle}(\text{entry}(d_2))$.

For each $a\in\Sigma$, D_a denotes the set of tiles admitted by a. We define $d\in D_a$ iff $d\in D$ and there exists a bijection m: entry(d) $\rightarrow$ exit(d) such that for all $c\in$ entry(d), $m(c)\in\delta(\text{state}(c),a)$.

Now, let $X\in D^{(2)}$ and let $n = \ell_1(X)$ and $m = \ell_2(X)$. X is satisfactory if for all i,j, $1 \le i \le n$ & $1 \le j \le m$:

(i) $\langle X_{i,j}, X_{i+1,j}\rangle \in V$ when $i \neq n$, and

$\langle X_{i,j}, X_{i,j+1}\rangle \in H$ when $j \neq m$;

(ii) $K_{\langle -1,0\rangle}(\text{entry}(X_{n,j})) = \emptyset$
& $K_{\langle 0,-1\rangle}(\text{entry}(X_{i,m})) = \emptyset$
& $K_{\langle 1,0\rangle}(\text{entry}(X_{1,j})) = \emptyset$
& $K_{\langle 0,1\rangle}(\text{entry}(X_{i,1})) = \begin{cases} \{q_0\} \text{ if } i=1, \\ \emptyset \text{ if } i\neq 1; \end{cases}$

and (iii) $K_{\langle -1,0\rangle}(\text{exit}(X_{1,j}))\subseteq F$
& $K_{\langle 0,-1\rangle}(\text{exit}(X_{i,1}))\subseteq F$
& $K_{\langle 1,0\rangle}(\text{exit}(X_{n,j}))\subseteq F$
& $K_{\langle 0,1\rangle}(\text{exit}(X_{i,m}))\subseteq F$.

Claim 1: If $A\in\Sigma^{(2)}$ is in T(B), then there exists a satisfactory description scene $X\in D^{(2)}$ such that for each i,j, $X_{i,j}\in D_{A_{i,j}}$.

Claim 2: If $X\in D^{(2)}$ is a satisfactory description scene, and $A\in D^{(2)}$ has the property that for each i,j, $X_{i,j}\in D_{A_{i,j}}$, then $A\in T(B)$.

To prove claim (1), observe that given $A\in T(B)$, there exists a loop-free computation which accepts A, that is, the bug never enters the same square in the same state twice. Now, for each i,j, look at this loop-free accepting computation and see what states and directions square i,j is entered and exited from. Call these transitions $c^{\text{entry}}_{i,j}$ and $c^{\text{exit}}_{i,j}$. For i=j=1 (the initial square), also put $\langle q_0,\langle 0,1\rangle\rangle$ into $c^{\text{entry}}_{1,1}$. We then let $X_{i,j} = \langle c^{\text{entry}}_{i,j}, c^{\text{exit}}_{i,m}\rangle$. The reader may verify that X defined in this way satisfies claim (1).

To prove claim (2), assume that $X\in D^{(2)}$ is a satisfactory description scene and that $A\in\Sigma^{(2)}$ has the property that for each i,j, $X_{i,j}\in D_{A_{i,j}}$. For each i,j, choose a particular path-defining bijection $m_{i,j}$: entry $(X_{i,j})\rightarrow$ exit $(X_{i,j})$ such that $m_{i,j}(c)\in\delta(\text{state}(c),A_{i,j})$ for all $c\in$ entry $(X_{i,j})$.

We use the functions $m_{i,j}$ so chosen to define a sequence of crossing transitions $c_0, c_1, \ldots$ and a sequence of instantaneous descriptions $\langle p_0, \vec{i}_0 \rangle, \langle p_1, \vec{i}_1 \rangle, \ldots$:

(a) $c_0 = \langle q_0, \langle 0,1 \rangle \rangle$, $p_0 = q_0$, and $\vec{i}_0 = \langle 1,1 \rangle$.

(b) For each $k > 0$, if $\vec{i}_{k-1}$ is the position of a square on X, then the sequences are defined at k, and $c_k = m_{\vec{i}_{k-1}}(c_{k-1})$, $p_k = \text{state}(c_k)$, and $\vec{i}_k = \vec{i}_{k-1} + \text{shift}(c_k)$.

Now, these sequences must terminate since all the $m_{i,j}$ are bijections and X is satisfactory. The only way the sequence can terminate, however, is to have some $\vec{i}_k$ be off the scene, so again by the condition that X be satisfactory, $q_k \in F$.

Finally, it can be verified that $\langle q_j, \vec{i}_j \rangle \vdash \langle q_{j+1}, \vec{i}_{j+1} \rangle$ for each j, $0 \leq j < k$, so the sequence of instantaneous descriptions is an accepting computation of the bug, and hence $A \in T(B)$, proving claim (2).

From these claims, we see that a string w is in L(B) if and only if there exists a satisfactory description scene $X \in D^{(2)}$ with the property that $\ell_2(X) = \ell(w)$ and $X_{1,j} \in D_{w_j}$ for each j, $1 \leq j \leq \ell(w)$. But, as outlined before, we can find an LBA M to test if such a description scene X exists, and hence L(B) is the language accepted by M. By the theorem of Kuroda, L(B) is context-sensitive.[3] □

4. Two-Tape Two-Way Finite Automata

We use the results of the preceeding section to settle some questions about two-tape two-way finite automata.

Definition 4.1: A (non-deterministic) n-tape two-way finite automaton M with end-markers is a 6-tuple $(n, K, \Sigma, \delta, q_0, q_f)$ where n is the number of tapes; K is a finite set of states; Σ is a finite set of input symbols; δ: $K \times (\Sigma \cup \{\vdash, \dashv\})^n \to 2^{(K \times S^n)}$, where $S = \{-1, 0, +1\}$ is the set of shifting operations; and q_0 and q_f are the start and final states respectively.

An instantaneous description is a pair $\langle q, \vec{i} \rangle$, where $q \in K$ and $\vec{i} \in N^n$ is a tape head position vector. Given machine M and a tape vector $\langle a^{(1)}, \ldots, a^{(n)} \rangle \in (\vdash \Sigma^* \dashv)^n$, define

$$\langle q, \vec{i} \rangle \vdash \langle q', \vec{i}' \rangle$$

if $1 \leq i_k \leq \ell(a^{(k)})$ for every k, $1 \leq k \leq n$, and for some $\vec{s} \in S^n$, $\langle q', \vec{s} \rangle \in \delta(q, a^{(1)}_{i_1}, a^{(2)}_{i_2}, \ldots, a^{(n)}_{i_n})$ and $\vec{i}' = \vec{i} + \vec{s}$.

As usual, define $\vdash^*$ to be the reflexive, transitive closure of $\vdash$.

M accepts an n-tuple of tapes $\vec{w} = \langle w_1, \ldots, w_n \rangle \in \Sigma^{*n}$ if, for each k, $1 \leq k \leq n$, when $\vdash w_k \dashv$ is initially written on the k^{th} tape of M, then

$$\langle q_0, \langle 1,1,\ldots,1 \rangle \rangle \vdash^* \langle q_f, \vec{i} \rangle$$

for some $\vec{i}$. Let T(M) denote the set of all n-tuples of words accepted by M.

T(M) may be thought of as a relation on Σ^*. For every relation $R \in \Sigma^{*n}$, let domain (R) = $\{w \in \Sigma^* \mid$ there exist $y_2, y_3, \ldots, y_n \in \Sigma^*$ such that $\langle w, y_2, y_3, \ldots, y_n \rangle \in R\}$.

Theorem 4.2: For every 2-dimensional bug B, there exists a 2-tape two-way finite automaton M such that L(B) = domain (T(M)).

Proof: Let $B = (2, K, \Sigma, \delta, q_0, F)$ be a bug. We construct a 2-tape two-way finite automaton M which simulates B. B's input scene is represented on M's second tape by simply stringing out the rows, one after the next, in order, separated by a new separator symbol # (see figure 5). Another copy of the top row of the scene is placed on M's first tape. M follows the instructions:

(1) Check that the segments of the second tape delimited by # are all equal in length to the length of the first tape.

(2) Check that the first tape is equal to the first segment of the second tape (i.e., that portion up to the first #).

(3) Begin simulating B, treating each segment of the second tape as a row of the scene. If B would shift left or right, shift the second tape left or right accordingly. If B would shift up or down, shift to the corresponding square in the preceeding or following segment of the second tape respectively by measuring off

n + 1 squares, using the first tape as a counter, where n is the length of the first tape.

(4) If B would ever fall off the represented scene in an accepting state, then accept.

Clearly, $\vec{w} \in T(M)$ iff $\vec{w}$ is the representation of some scene $A \in T(B)$, so domain $(T(M)) = L(B)$. □

Theorem 4.3: For every 2-tape two-way finite automaton M, there exists a 2-dimensional bug B such that domain $(T(M)) = L(B)$.

Proof: Let $M = (2, K, \Sigma, \delta, q_0, q_f)$ be a 2-tape two-way finite state acceptor. We represent a pair of tapes $u, v \in \vdash\Sigma^*\dashv$ by a scene whose top row is u and whose successive rows are the rows of the array A, where $A_{i,j} = \langle u_j, v_i \rangle$, $1 \le i \le \ell(v)$ and $1 \le j \le \ell(u)$ (see figure 6). We define a bug B which follows the instructions:

(1) Check that the input scene is a representation of a pair of tapes.

(2) Simulate the action of M, interpreting the first member of each pair as the symbol scanned on the first tape, and the second member as the symbol from the second tape. Use horizontal shifts to mimic shifts on the first tape and vertical shifts to mimic shifts on the second tape. Accept only if you discover that M would.

Clearly, $A \in T(B)$ iff A is the representation of a pair of tapes $w \in T(M)$, so $L(B) = \vdash \text{domain}(T(M)) \dashv$. By lemma 2.5, domain $(T(M))$ is also 2-bug-definable. □

The above two theorems together with theorems 3.1 and 3.4 establish a second characterization of the context-sensitive languages.

Theorem 4.4: A language is the domain of some 2-tape two-way finite state acceptor if and only if it is context-sensitive.

5. Higher Dimensional Bugs and Multitape Two-Way Finite Automata.

We remark first that theorems 4.2 and 4.3 can be generalized to n-dimensions, for arbitrary n, to give the result:

Theorem 5.1: A language is n-bug-definable if and only if it is the domain of some n-tape two-way finite state acceptor.

We now show that for each $n \ge 3$, the entire class of recursively enumerable sets is definable.

Theorem 5.2: For any $n \ge 3$, a language is recursively enumerable iff it is n-bug-definable.

Proof: By theorem 5.1, the n-bug-definable languages are the same as the domains of n-tape finite state acceptors. With 3 tapes, a finite state acceptor M can simulate a 2-counter machine using its second and third tapes as counters and its first tape as the input to the counter machine. Since a 2-counter machine can recognize any recursively enumerable set,[4] any recursively enumerable set can, therefore, be the domain of some 3 (or more) tape finite state acceptor.

Conversely, one can clearly enumerate the domain of an n-tape finite state acceptor by trying all possible n-tuples of tapes and all possible computations involving those tapes. □

Acknowledgement

The author is indebted to J.D. Ullman for bringing the problem to his attention and for the theorems of section 4. The theorems of section 5 were observed in conversations with W.C. Rounds and W.F. Ogden.

References

[1] Blum, M. and Hewitt, C., "Automata on a 2-dimensional tape," IEEE Conference Record of the 1967 Eighth Annual Symposium on Switching and Automata Theory, 155-160.

[2] Hopcroft, J.E. and Ullman, J.D., Formal Languages and their Relation to Automata, Addison-Wesley, Reading, Mass. (1969).

[3] Kuroda, S.Y., "Classes of languages and linear-bounded automata," Inf. and Control 7, no. 2, 207-223 (1964).

[4] Minsky, M.L., Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, N.J. (1967).

[5] Shepardson, J.C., "The reduction of two-way automata to one-way automata," IBM Journal of Research and Development 3, no. 2, 198-200 (1959).

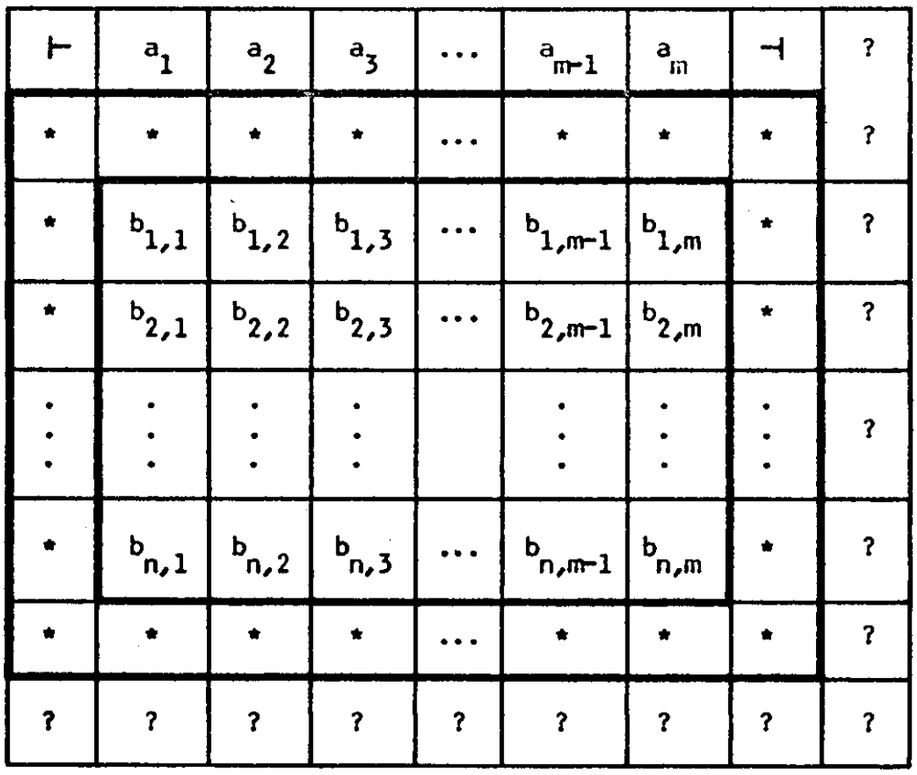

Figure 1: Proof of lemma 2.5, addition of endmarkers.

a_2	a_3	a_4	a_5	...	a_{m-3}	a_{m-2}	a_{m-1}	?
*	*	*	*	...	*	*	*	?
*	$\langle b_{1,1}, b_{1,2}, b_{1,3} \rangle$	$b_{1,4}$	$b_{1,5}$	...	$b_{1,m-3}$	$\langle b_{1,m-2}, b_{1,m-1}, b_{1,m} \rangle$	*	?
*	$\langle b_{2,1}, b_{2,2}, b_{2,3} \rangle$	$b_{2,4}$	$b_{2,5}$	...	$b_{2,m-3}$	$\langle b_{2,m-2}, b_{2,m-1}, b_{2,m} \rangle$	*	?
. . .	. . .	. . .	. . .		. . .	. . .	. . .	?
*	$\langle b_{n,1}, b_{n,2}, b_{n,3} \rangle$	$b_{n,4}$	$b_{n,5}$	...	$b_{n,m-3}$	$\langle b_{n,m-2}, b_{n,m-1}, b_{n,m} \rangle$	*	?
*	*	*	*	...	*	*	*	?
?	?	?	?	?	?	?	?	?

Figure 2: Proof of lemma 2.5, removal of endmarkers.

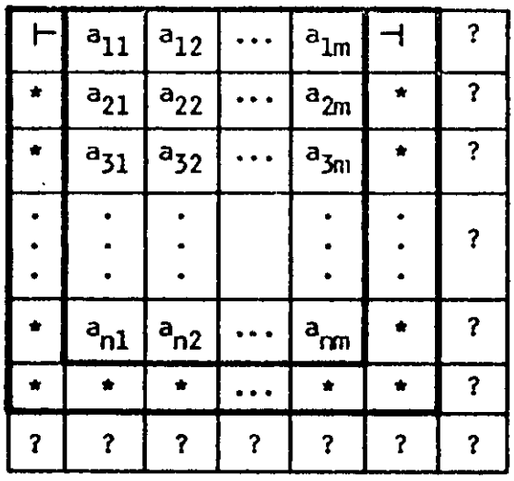

Figure 3: Form of scene for proof of theorem 3.1.

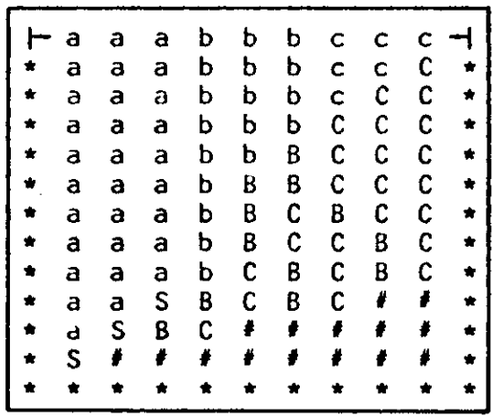

Figure 4: Representation of a derivation of a context-sensitive grammar.

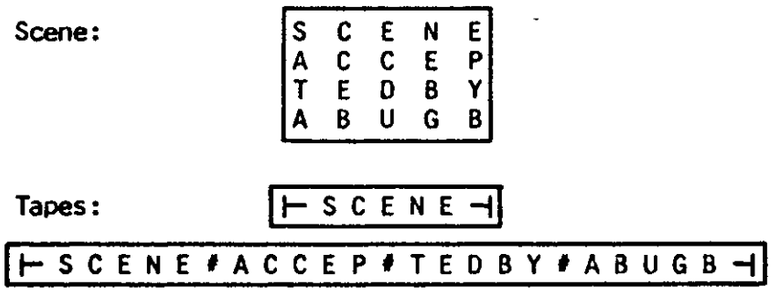

Figure 5: Representation of a 2-dimensional scene by a pair of tapes.

Tapes:

Scene:

⊢	F	I	R	S	T	⊣
⟨⊢,⊢⟩	⟨F,⊢⟩	⟨I,⊢⟩	⟨R,⊢⟩	⟨S,⊢⟩	⟨T,⊢⟩	⟨⊣,⊢⟩
⟨⊢,S⟩	⟨F,S⟩	⟨I,S⟩	⟨R,S⟩	⟨S,S⟩	⟨T,S⟩	⟨⊣,S⟩
⟨⊢,E⟩	⟨F,E⟩	⟨I,E⟩	⟨R,E⟩	⟨S,E⟩	⟨T,E⟩	⟨⊣,E⟩
⟨⊢,C⟩	⟨F,C⟩	⟨I,C⟩	⟨R,C⟩	⟨S,C⟩	⟨T,C⟩	⟨⊣,C⟩
⟨⊢,O⟩	⟨F,O⟩	⟨I,O⟩	⟨R,O⟩	⟨S,O⟩	⟨T,O⟩	⟨⊣,O⟩
⟨⊢,N⟩	⟨F,N⟩	⟨I,N⟩	⟨R,N⟩	⟨S,N⟩	⟨T,N⟩	⟨⊣,N⟩
⟨⊢,D⟩	⟨F,D⟩	⟨I,D⟩	⟨R,D⟩	⟨S,D⟩	⟨T,D⟩	⟨⊣,D⟩
⟨⊢,⊣⟩	⟨F,⊣⟩	⟨I,⊣⟩	⟨R,⊣⟩	⟨S,⊣⟩	⟨T,⊣⟩	⟨⊣,⊣⟩

Figure 6: Representation of a pair of tapes by a 2-dimensional scene.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)

Carnegie-Mellon University
Department of Computer Science
Pittsburgh, Pennsylvania 15213

2a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED

2b. GROUP:

3. REPORT TITLE

TWO CHARACTERIZATIONS OF THE CONTEXT-SENSITIVE LANGUAGES

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Scientific Interim

5. AUTHOR(S) (First name, middle initial, last name)

Michael J. Fischer

6. REPORT DATE: September 1969

7a. TOTAL NO. OF PAGES: 13

7b. NO. OF REFS: 5

8a. CONTRACT OR GRANT NO.: F44620-67-C-0058

b. PROJECT NO.: 9718

c. 6154501R

d. 681304

9a. ORIGINATOR'S REPORT NUMBER(S):

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report):

10. DISTRIBUTION STATEMENT

1. This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES: TECH, OTHER

12. SPONSORING MILITARY ACTIVITY: Air Force Office of Scientific Research (SRMA)
1400 Wilson Boulevard
Arlington, Virginia 22209

13. ABSTRACT

An n-dimensional bug-automation is generalization of a finite state acceptor to n-dimensions. With each bug B, we associate the language L(B) which is the set of top rows of n-dimensional rectangular arrays accepted by B. One-dimensional bugs define trivially the regular sets. Two-dimensional bugs define precisely the context-sensitive languages, while bugs of dimension 3 or greater define all the recursively enumerable sets.

We consider also finite state acceptors with n two-way non-writing input tapes. For each such machine M, let domain (M) be the set of all strings which are the first component of some n-tuple of tapes accepted by M. For any $n \geq 1$, the domains of n-tape two-way finite state acceptors are precisely the same as the languages definable by n-dimensional bugs, so as a corollary, the domains of two-tape two-way finite state acceptors are precisely the context-sensitive languages.

DD FORM 1 NOV 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT