THE UNIVERSITY OF MICHIGAN

INFORMATION SYSTEMS LABORATORY
SENSORY INTELLIGENCE LABORATORY

Department of Electrical Engineering
College of Engineering

Technical Report ISL-65-2

MONOTONE CONGRUENCE ALGORITHMS

Richard F. Arnold
Donald L. Richards

April 1965

This work was supported in part by Grants No. AF-AFOSR-367-63 and AF-AFOSR-367-64 of the U.S. Air Force Office of Scientific Research, Directorate of Information Sciences, Office of Aerospace Research, in part by Contract No. AF 30(602)-3546, Rome Air Development Center, and in part by NSF Grant GP-2778. Some of the material was presented at the ICMCI Conference in Tokyo, Japan under the title "Monotone Reduction Algorithms".

Nomenclature

Σ	finite set (alphabet)
s (or s_i)	arbitrary element of Σ (letter)
n	number of elements of Σ
Σ^*	set of all expressions on Σ (words)
λ	empty word
k, k'	length of a word
m	index on the letters of a word
t, u, v, w, x, y, z (or t_i, u_i, etc.)	arbitrary word
ϵ	is a member of
$\subseteq$	is a subset of
$<$	is less than (relation of total order on Σ^*)
$>$	is greater than
$\sim$	is congruent to (two-sided congruence relation on Σ^*)
$\lhd$	is included in (relation of partial order on Σ^*)
$x \rightarrow y$	simple substitution
$x \dot{\rightarrow} y$	concluding substitution
A, B, C	arbitrary algorithm
A(w)	the word which results when A is applied to w
$\equiv$	is equivalent to (equivalence relation on algorithms)
A_L	set of words appearing at the left of substitutions of A
A_R	set of words appearing at the right of substitutions of A
A'	core algorithm for A
π	auxiliary alphabet
α	class of algorithms
a(w)	number of applications of A to w
P, Q, S, S_s, T, U	specific sets of words

Abstract

Given an associative system in which each element is assigned a cost and in which an equivalence relation obtains between elements, it is often of interest to ask the question: what is the least costly word equivalent to a given word? The case in which the equivalence relation is a two-sided congruence relation is studied here, one example being the problem of optimizing a type of computer program. Such optimizing processes may be formalized as Markov normal algorithms.

A general result concerning order relations on finite alphabets is established first. Then, the properties of a class of Markov normal algorithms are investigated. It is shown that each such algorithm must always terminate, and that every class of mutually equivalent algorithms contains a unique minimal algorithm which can be obtained by applying any algorithm of the class to itself.

1. Motivation

Given an associative system (semi-group) in which each element is assigned a cost and in which an equivalence relation obtains between elements, it is often of interest to ask the question: what is the least costly word equivalent to a given word? Three examples of such problems are the travelling salesman problem studied by Dantzig, et al. (1954), the optimum control problem (cf. Pontryagin, 1962), and the problem of finding the least word which performs a given mapping upon the states of a finite automaton. Algorithms do not exist for all such problems, since the solution may in general require the solution of the word problem for semi-groups, which is known to be unsolvable (Davis, 1958).

The problems mentioned above involve natural two-sided congruence relations. Two input words x and y to a finite automaton may be regarded as equivalent if they perform the same mapping from the set of states of the automaton to itself; i.e., if $M(s,x) = M(s,y)$ for all states s where M is the transition function of the automaton. A simplified model of computer programming can be obtained by reinterpreting the input words as programs, and the initial state as data upon which the programs act; here the programs are not self-modifying and contain no instructions which transfer control. Programs P and Q are then equivalent if they reach the same result for each set of data; in that case the programs RPS and RQS must also be equivalent, R being run immediately before P (or Q) and S being run immediately after. Thus the equivalence relation satisfies the definition of a two-sided congruence.

The programming and automaton problems can be solved by enumeration if necessary; however, the solution algorithms are often impractical and may give no intuitive insight into the structure of the optimizing process itself. It is hoped that greater insight and more effective algorithms can be found by studying the general class of associative systems with two-sided congruence relations, a class which contains problems of all degrees of difficulty.

Where such a two-sided congruence relation exists and, in addition, there is a total order relation on the costs of the words which has certain natural properties, the theory of monotone congruence algorithms applies. The properties of this class of algorithms are investigated below. A general result concerning order relations on finite alphabets is established first. Then it is shown that each monotone congruence algorithm must always terminate, and that every class of mutually equivalent monotone congruence algorithms contains a unique minimal algorithm which can be obtained by applying any algorithm of the class to itself.

2. Monotone Congruence Algorithms

Let Σ be a finite set called the alphabet, whose elements are called letters. Let Σ^* denote the set of all expressions or words on the letters of Σ, including the empty word λ.

(The letters may be interpreted as computer instructions or subroutines; the words, as programs or as systems of subroutines. The word uv represents the program formed when the program u is followed by the program v.)

Let $<$ ("less than") be a total order relation on Σ^*. The symbols $\leq$, $>$, and $\geq$ will be used in the usual way.

(In the interpretation, "$<$" will often be derived from a cost function on the letters, so that the cost of uv is the cost of u plus the cost of v. Within classes of words having the same cost, the total order can be completed lexicographically as follows. Assume u and v have the same cost. If u is shorter than v, let $u < v$. If u and v have the same length and $u=ws_1u_1$ and $v=ws_2v_1$ for $s_1, s_2 \in \Sigma$, $s_1 \neq s_2$ and $w, u_1, v_1 \in \Sigma^*$, define $u < v$ if $s_1 < s_2$ and $v < u$ otherwise. The total order thus defined can be shown to satisfy assumptions (1) and (2).

(The cost of a computer program might reflect its execution time, the space required, etc.).

Assume that:

(1) $\lambda \leq w$ for all $w \in \Sigma^*$;

(2) if $x < y$, then $uxv < uyv$, for all u, v, x, and $y \in \Sigma^*$.

(The first assumption implies that the "empty program" is less costly than any other program; the second assumption holds wherever the cost of uv is the cost of u plus the cost of v, and in many other cases as well.)

Let $\sim$ (" is congruent to") be an equivalence relation on Σ such that

(3) if $x \sim y$, then $uxv \sim uyv$ for all u, v, x, and $y \in \Sigma^*$.

(x is congruent to y if these two programs compute the same function upon the internal states of the computer.)

Definition

If $v \sim w$ implies $v \geq w$ for all v in $\in \Sigma^*$, w is a minimal word (over $<$ and $\sim$). If $v \sim w$ and w is a minimal word, then w is the minimal equivalent word for v.

Definition

If there exist u, $v \in \Sigma^*$ such that $uxv = y$, then x is a subword of y. If $u \neq \lambda$ or $v \neq \lambda$, then x is a proper subword of y.

Theorem 1.

Any subword of a minimal word is a minimal word.

PROOF: Suppose, for purposes of indirect proof, that $w = uxv$ where w is minimal and x is not. There must be a word y such that $y \sim x$, $y < x$; but then $uyv \sim uxv = w$ and $uyv < w$ by (2) and (3), which contradicts the minimality of w.

Theorem 1 is a natural generalization of the Principle of Optimality of Bellman (1957) to two-sided congruence relations. Note that it does not depend upon the introduction of a particular algorithm.

A general notion of algorithm is now defined following Markov (1961).

Definition

Let "$\rightarrow$" and "$\dot{\rightarrow}$" be symbols not in Σ. Then "$x \rightarrow y$" is a simple substitution of Σ^* and "$x \dot{\rightarrow} y$" is a concluding substitution of Σ^* if x, $y \in \Sigma^*$.

Definition

A Markov normal algorithm on Σ^* is a finite ordered list of substitutions of Σ^*.

The letters A, B, and C will be used to denote algorithms. The set of words which appear to the left of the arrows in the substitutions of A will be denoted A_L ("A-left") while those which appear to the right of the arrows will be denoted A_R ("A-right").

Definition

The substitution "$x \to y$" (or "$x \mathring{\to} y$") applies to $w \in \Sigma^*$ if there exist $u, v \in \Sigma^*$ such that $w = uxv$. An algorithm applies to w if any of its substitutions applies to w.

By convention, the operation of a Markov normal algorithm upon a word w is interpreted as follows. If "$x \to y$" (or "$x \mathring{\to} y$") is the first substitution of the algorithm, $w = uxv$, and u_1 is at least as long as u wherever $w = u_1'xv_1'$, the first application consists of replacing uxv by uyv. The same procedure is then repeated for uyv. If the first substitution does not apply to w the second substitution is considered instead; if neither applies, the third substitution is considered, etc. Whenever a concluding substitution applies, the process stops immediately after it has been applied. Otherwise, the process stops only when a word is reached to which none of the substitutions apply. Since the process is deterministic, it must produce a unique sequence of words, $w, w_1, w_2, \ldots$ of length 1 or more. The sequence is of length 1 only if the algorithm does not apply to w. The number of applications $a(w)$ of A to w is one less than the length of the sequence. The algorithm A produces x from w if x is a member of the sequence. If z is the last word in the sequence, define $A(w) = z$.

Definition

$A \equiv B$ ("A is equivalent to B") if A and B are algorithms and $A(w) = B(w)$ whenever either $A(w)$ or $B(w)$ exists.

The algorithms which form the main subject of investigation can now be defined.

Definition

Let Σ be an alphabet, $<$ be an order relation satisfying (1) and (2), and $\sim$ be an equivalence relation satisfying (3). A monotone congruence algorithm (MCA) over $<$ and $\sim$ is a Markov normal algorithm A for which:

(a) $A(w)$ is minimal for all $w \in \Sigma^*$ for which $A(w)$ exists; and

(b) for each substitution "$x \rightarrow y$" (or "$x \dot{\rightarrow} y$") of A, $x > y$ and $x \sim y$.

(In terms of computer programs, each substitution gives a way of replacing a subprogram by a less costly program which computes the same function; if the algorithm terminates, it must produce the least costly program which computes the same function.)

Example. For $\Sigma = \{1, 2, 3\}$, the following MCA maps $x, y \in \Sigma^*$ to the same word if and only if the sum of the digits of x is equal to the sum of the digits of y modulo 4. Let the letters 1, 2, and 3 cost 7, 2, and 4 units, respectively, and let "$<$" be induced by these costs. The minimal words are 22, 23, 2, and 3. The algorithm is:

$$
\begin{aligned}
222 &\dot{\rightarrow} 2 \\
32 &\rightarrow 23 \\
1 &\rightarrow 32 \\
33 &\rightarrow 2 \\
223 &\rightarrow 3.
\end{aligned}
$$

(If the letters 1, 2, and 3 are interpreted as shift instructions for a four-bit circular shift register, the algorithm maps each program of shift instructions to the least costly equivalent program.)

3. Termination

The first principal result to be established is that for any MCA A and any word w, A(w) exists; i.e., that every MCA always terminates. The proof depends on a quite general result about total order relations on Σ^*.

Definition

For x,y distinct words of Σ^*, $x \lhd y$ ("x is included in y") if $x = x_1 s_2 \ldots s_m$ for $s_i \in \Sigma$ (i=1,2,...,m) and there exist words $v_0, v_1, \ldots, v_m \in \Sigma^*$ such that $y = v_0 s_1 v_1 \ldots s_m v_m$. The partial ordering thus defined is called the inclusion partial ordering.

Observe that every total ordering of Σ^* satisfying (1) and (2) is a refinement of the inclusion partial ordering in the sense that $x \lhd y$ implies $x < y$.

Theorem 2.

Every total order which is a refinement of the inclusion partial ordering is a well-ordering.

A proof of Theorem 2 is given in the Appendix.

Theorem 3.

For any MCA A and any $w \in \Sigma^*$, A(w) exists.

PROOF: A(w) must exist unless A produces an infinite descending sequence of words $w > w_1 > w_2 > \ldots$, which contradicts the fact that "<" is a well-ordering.

Markov normal algorithms may in general involve the use of auxiliary letters. That is, the alphabet of A may consist of disjoint subsets Σ and π such that for all $w \in \Sigma^*$, $A(w) \in \Sigma^*$. Where attention is concentrated on the words of Σ^*, the letters of π may then be called auxiliary letters, since

they appear only in the intermediate stages of the application of A to the words of Σ. The words with auxiliary letters are required to obey the usual conditions on the ordering and congruence relations.

The next two theorems show that every monotone congruence algorithm is equivalent to one which has neither auxiliary letters nor concluding substitutions.

Theorem 4.

If A is an MCA such that $A(w) \in \Sigma^*$ for every word $w \in \Sigma^*$, then A is equivalent on Σ^* to an MCA whose substitutions involve only words of Σ^*.

PROOF: Given A, form the algorithm B by deleting each substitution "$x \rightarrow y$" (or "$x \dot{\rightarrow} y$"), for which $x \notin \Sigma^*$ and then replacing each of the remaining words of A_R by its minimal equivalent word. B_L and B_R then contain only words of Σ^*. To show that $A(w) = B(w)$ for all $w \in \Sigma^*$, note that if w is minimal, $A(w) = B(w) = w$. If w is not minimal, some substitution "$x \rightarrow y$" (or "$x \dot{\rightarrow} y$") of A must apply to w, with $w = uxv$ and $u, x, v \in \Sigma^*$. Then, by the choice of B, the substitution "$x \rightarrow z$" (or "$x \dot{\rightarrow} z$") must be in B, where z is the minimal equivalent word for x. It follows from the hypothesis of the theorem that z is also in Σ^*, whence $uzv \in \Sigma^*$ as well. Thus, after one application, B has replaced w by a lesser equivalent word from Σ^*. The argument can be repeated for the new word, etc.; after a finite number of repetitions, B will produce the minimal equivalent word for w. Thus $B(w) = A(w)$ for every word $w \in \Sigma^*$, which was to be shown.

Theorem 5.

Each MCA is equivalent to an MCA which has no concluding substitutions.

PROOF: Given an MCA A, form the MCA B by replacing each concluding substitution "$x \dot{\rightarrow} y$" by the simple substitution "$x \rightarrow y$." If $B \not\equiv A$, there must be a word w such that $B(w) \neq A(w)$, and the application of A to w must involve atleast one concluding substitution. By the nature of concluding substitutions, only one can have been applied, and it must have been applied only at the last step, producing A(w). Since B has the same substitutions as A, the sequence of words produced by B from w must be identical up to and including the step which produces A(w); since $B(w) \neq A(w)$, some substitution of B must apply to A(w) also, reducing it to a lesser word which is congruent to A(w). Then A(w) cannot be a minimum word, and A cannot be an MCA. The contradiction establishes that $A \equiv B$.

Note that if the substitutions of an MCA which has no concluding substitutions are reordered, the resulting algorithm is equivalent.

4. The Core Algorithm

The second main result is that every class of mutually equivalent monotone congruence algorithms contains a unique minimal algorithm which can be obtained by applying any member of the class to itself. The algorithm is unique up to the order of the substitutions and minimal with respect to the number of substitutions.

Definition

For any MCA A, the core algorithm A' is formed by:

(1) replacing all concluding substitutions by the identical simple substitutions;

(2) where two or more substitutions have the same left-hand word, deleting all but one of them;

(3) deleting every substitution "$x \rightarrow y$" for which some proper subword of x is in A_L; and

(4) replacing each remaining word of A_R by its minimal equivalent word.

Theorem 6.

For any MCA A,

(1) A' is an MCA;

(2) $A' \equiv A$;

(3) If $B \equiv A$, then $A'_L \subseteq B_L$.

PROOF: A' is chosen so that $x > y$ and $x \sim y$ for each of its substitutions "$x \rightarrow y$". Thus to show that A' is an MCA and that $A' \equiv A$, it is only necessary to show that A' applies to every word to which A applies. If A applies to w, some word $v \in A_L$ must be a subword of w; but then, by the choice of A', either $v \in A'_L$ or some proper subword of v is in both A_L and A'_L. In either

case, A' applies to w also. Thus A' successively reduces each word to lesser words to which it is congruent until, eventually, the minimal equivalent word is reached; $A' \equiv A$.

Observe that A' was chosen in such a way that each proper subword of a word of $x \in A'_L$ is a minimal word; for otherwise, the substitution "$x \rightarrow y$" would have been deleted in forming A'. Now suppose that $B \equiv A$. B must apply to each word of A'_L, since none of them is minimal. At the same time, B cannot apply to any proper subword of any of the words of A'_L since all the proper subwords <u>are</u> minimal. Thus each word of A'_L must appear as the left side of a substitution in B. The words of A'_L being distinct, it follows that $A'_L \subseteq B_L$. This completes the proof.

For any class of equivalent MCA containing A, A'_L is the unique set of non-minimal words whose proper subwords are all minimal, each word of A'_R is the unique minimal equivalent word for the corresponding word of A'_L, and there are no concluding substitutions. Thus A' is unique except for the order in which the elements of A'_L appear.

5. Uniform Optimality

The core algorithm is minimal with respect to the number of substitutions. A further kind of minimality based upon the number of steps required to reduce a word is now considered.

Definition

An algorithm C is uniformly optimal within a class α of equivalent algorithms if, for all $A \in \alpha$ and all $w \in \Sigma$, $a(w) \leq c(w)$.

Note that a(w) (defined earlier as the number of applications of A to w), usually depends on the order of the substitutions of A.

Theorem 7.

The only MCA which is uniformly optimal within a complete class of equivalent MCA is the algorithm which has no substitutions.

PROOF: The algorithm which has no substitutions is not equivalent to any other MCA. If, on the other hand, A applies to any word w, it must also apply to each word of the sequence ww, www, ... Let z be any member of this sequence which is not in the finite set A_L; then $a(z) > 1$. Form B from A by adding the substitution "$z \rightarrow y$" at the beginning, where y is the minimal equivalent word for z. Then $b(z) = 1 < a(z)$, and A is not uniformly optimal within its complete class of equivalent algorithms.

Theorem 8.

If C is uniformly optimal within the class of α of equivalent MCA such that $A_L = C_L$ for every $A \in \alpha$, and no two substitutions of C have the same left-hand word, then each word of C_R is a minimal word.

PROOF: Suppose that in the substitution "$x \rightarrow y$" of an MCA A, y is not a minimal word. Then $a(x) > 1$. If y is replaced by its minimal equivalent word to form the algorithm B, then $B \equiv A$ and $b(x) = 1 < a(x)$, so A is not uniformly optimal within the class.

It was shown in Theorem 6 that any algorithm with a minimum number of substitutions must have the same set of left-hand words as a core algorithm. Theorem 8 shows that within those with the same set of left-hand words, only the ones which have the same set of right-hand words as the core algorithm can be uniformly optimal. Thus the study of optimality reduces to the study of the algorithms which can be produced by reordering the substitutions of the core algorithm. It will be shown in conclusion that a uniformly optimal ordering of the subsitutions does not always exist, and that more than one uniformly optimal ordering may exist.

Example. It can be shown that

$$01 \rightarrow \lambda$$
$$10 \rightarrow \lambda$$
$$00 \rightarrow 1$$
$$11 \rightarrow 0$$

is a core algorithm which is uniformly optimal for the class of equivalent MCA which have the same left sides. Other uniformly optimal orderings of the same core algorithm can be obtained by interchanging the first and second substitutions or the third and fourth substitutions. Note that the left-hand sides are not ordered according to the $<$ relation for any of these optimal orderings; for if $0 < 1$, then $00 < 01$.

Finally, there exist core algorithms which cannot be reordered to produce an algorithm which is uniformly optimal within the class of equivalent algorithms with the same left sides.

Example.

$$01 \to 2$$
$$10 \to 2$$
$$02 \to 20$$
$$12 \to 21.$$

Let $u = 010$ and $v = 101$. If A is any ordering of these substitutions such that "$01 \to 2$" precedes "$10 \to 2$", then $a(u) = 1$ and $a(v) = 2$. If, on the other hand, B is any ordering of the substitutions such that "$10 \to 2$" precedes "$01 \to 2$", then $b(u) = 2$ and $b(v) = 1$. Since $a(u) < b(u)$ and $a(v) > b(v)$, neither A nor B can be uniformly optimal.

Appendix

Included here are the proofs of Theorem 2 and of a lemma concerning the inclusion partial order; the proof of the latter is the principal step in proving the theorem.

The definition of the partial order $\lhd$ is restated here for reference.

Definition

For x,y distinct words of Σ , $x < y$ ("x is included in y") if $x = s_1 s_2 \cdots s_m$ for $s_i \in \Sigma$ $(i=1,2,\ldots,m)$ and there exist words $v_0, v_1, \ldots, v_m \in \Sigma$ such that $y = v_0 s_1 v_1 s_2 v_2 \cdots s_m v_m$.

Lemma 1.

Every infinite subset of Σ contains two words v and w such that $v \lhd w$.

PROOF: A set of words such that no word in the set is included in any other word of the set will be called independent. An infinite independent set of words whose alphabet has at most n letters and whose shortest word is of length k will be called an (n,k) i.i. set. The proof that no infinite independent set exists proceeds by an indirect proof involving simultaneous backward induction on n and k; i.e., it is shown that if an (n,k) i.i. set exists, then either an (n,k-1) i.i. set exists or an (n-1, k') i.i. set exists for some k'. Repeated application of the argument must yield a (1,1) i.i. set. Since such a set obviously does not exist, the hypothesis that an infinite independent subset exists must also be false.

Suppose, then, that an (n,k) i.i. set S is given. If the alphabet has less than n letters, the induction step is immediate. Otherwise, pick any word w from S which is of length k and let s denote the first letter of w.

Throughout the proof, it is frequently convenient to divide the infinite set under consideration into two subsets and to prove for one of them that if it is infinite, either an (n, k-1) i.i. set or an (n-1, k') i.i. set exists. Since the induction step is completed in that event, the subset can be discarded, and the other of the two subsets can be assumed infinite.

The process just outlined can be applied to the set S_s of all words of S which start with the letter s. The set of words formed from S_s by removing the initial letter of each word of S_s contains a word of length k-1 (namely, the word formed by deleting the initial letter of w) and is an independent set (or else, trivially, S_s is not independent). Thus if S_s is infinite, an (n, k-1) i.i. set exists.

Similarly, the set of words of S which do not include the letter s is either finite or it is an (n-1, k) i.i. set. When both this set, and all of S_s except w are eliminated from S, there remains a set $T = \{t_i\}$ which can be assumed to be infinite. Thus T is an (n, k) i.i. set in which each word contains at least one occurrence of the letter s, and no word except w begins with s.

The word $t_o = w$ is of the form sy_o for some word y_o; each other word t_i of T is of the form $x_i s y_i$, where x_i does not contain the letter s and each x_i is of length 1 or more. Let X denote the set $\{x_1, x_2, \ldots\}$, reordered so that if $x_i \lhd x_j$, then $i < j$.

It will now be shown that X contains either an infinite ascending chain $P = \{p_1, p_2, \ldots\}$ where $p_i \lhd p_{i+1}$ for all i, or else an infinite independent sequence Q. The two sequences P and Q of elements of X can be chosen as follows, beginning with the element x_1. At each stage, ask whether the element x_i then under consideration is included in (or equal to) an

infinite number of elements of X. If so, include x_i in P and go on to consider the next element in the order which includes (or is equal to) x_i. If not, include x_i in Q and go on to consider the next element in the order which includes no earlier element of Q (and which is not already in Q). At each stage only a finite number of elements of Q have been chosen, and each has been chosen so that it includes (or is equal to) only a finite number of elements of X. Therefore an infinite number of elements which include no earlier element of Q (and which are not already in Q) must remain at each stage, and the process can be continued indefinitely.

Q is an independent set on an alphabet which does not contain the letter s; it is therefore either an (n-1, k') i.i. set or finite. Accordingly, P can be assumed infinite.

The elements $t_i = x_i s y_i$ of T for which x_i is not a member of P may be discarded. The remaining members of T can be ordered according to the order of the elements in P; then if $t_i = x_i s y_i$ appears before $t_j = x_j s y_j$, either $x_i = x_j$ or $x_i \lhd x_j$. Here the words y_i must all be distinct; otherwise $x_i s y_i \trianglelefteq x_j s y_j$ for some i, j, which contradicts the independence of T.

Wherever $y_j \lhd y_i$, $j > i$, in the above ordering, remove t_j leaving a set U. U is independent because it is a subset of the independent set T.

Now suppose that U is infinite. The set of t_j just removed must then be infinite, and each y_j must be included in one of the y_i which remains. The number of words included in each remaining y_i must be finite, since none can be longer than y_i itself. Since the number of y_i is also finite, the infinite number of y_j included in them cannot all be distinct, which is a contradiction. Hence U is infinite.

So far it has been shown that either an (n, k-1) i.i. set, an (n-1, k') i.i. set, or the infinite independent set U constructed above must exist. In conclusion, it is shown that the set Y of all the y_i which appear as tails of elements of U is an (n, k-1) i.i. set. The word y_o of Y is of length k-1, since $w = sy_o$ and w was assumed to have length k. If $y_i \lhd y_j$, $i < j$, then $x_i sy_i \lhd x_j sy_j$, which contradicts the independence of T, since already $x_i \trianglelefteq x_j$. On the other hand, U was defined so as to exclude the case in which $y_i \lhd y_j$ for $i > j$. Thus Y is an (n, k-1) i.i. set and the lemma is proved.

Theorem 2.

Every total order which is a refinement of the inclusion partial ordering is a well-ordering.

PROOF: Let $<$ be a total order on Σ^* such that $x \lhd y$ implies $x < y$. If $\leq$ is not a well-ordering, there exists an infinite sequence of words $w_1 > w_2 > \ldots$. An infinite independent set $V = \{v_1, v_2, \ldots\}$ can be selected from the sequence as follows. Let $v_1 = w_1$. For $i \geq 1$, let v_{i+1} be the first word of the sequence which is not included in $v_1, v_2, \ldots,$ or v_i. Since only a finite number of words is included in any given word, such a v_{i+1} always exists and must precede an infinite number of words of the sequence. Thus V is an infinite independent set, contradicting the lemma above, and the hypothesis that $<$ is not a well-ordering must be false.

References

Aris, R. (1963) Discrete Dynamic Programming, Blaisdell, New York.

Bellman, R. E. (1957) Dynamic Programming, Princeton University Press, Princeton, New Jersey.

Dantzig, G. B., Fulkerson, D. R., and Johnson, S. (1954) Solution of a Large-Scale Travelling-Salesman Problem, Journal of the Operations Research Society of America, Vol. 2, No. 4, pp. 393-410.

Davis, M. D. (1956) Computability and Unsolvability, McGraw-Hill, New York.

Markov, A. A. (1961) Theory of Algorithms The Israel Program for Scientific Translations, Jerusalem.

Pontryagin, L. S. (1962) The Mathematical Theory of Optimal Processes, Interscience Publishers, New York.