COMPLEXITY MEASURES FOR LANGUAGE RECOGNITION BY CANONIC SYSTEMS

Joseph P. Haggerty

October 1970

PROJECT MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE MASSACHUSETTS 02139

COMPLEXITY MEASURES FOR LANGUAGE RECOGNITION BY CANONIC SYSTEMS

by

JOSEPH PATRICK HAGGERTY

Submitted to the Department of Electrical Engineering on January 20, 1969 in partial fulfillment of the requirements for the Degree of Master of Science in Electrical Engineering.

ABSTRACT

A canonic system C is a specification of a recursively enumerable set, such as a set of strings over a finite alphabet. From this description C, it is possible to generate a system C_m, called a proof measure function, which is an indication of the complexity of the language defined. For certain simple but important classes of canonic systems, algebraic bounds on these functions can be derived from the structure of the system. Another transformation on C produces a system C^{-1} which characterizes the recognition of strings generated by C. A relationship exists between the measure functions of C and C^{-1}, thus relating the complexity of the recognition procedure to that of the language description.

Thesis Supervisor: John J. Donovan
Title: Assistant Professor of Electrical Engineering

Acknowledgements

Perhaps the most vital ingredient in a successful thesis is a generous dash of enthusiasm, both on the part of the student and his supervisor. I feel fortunate in having encountered John Donovan, whose Irish optimism encouraged this work from the very beginning.

To fellow graduate students Robert Mandl and Amitava Bagchi I owe many technical discussions, some of them even relevant.

Finally, I wish to thank Lynn Foster, who assisted with the typing.

Table of Contents

Chapter 1 - Review of Canonic Systems

Introduction

A canonic system is a recursive definition of the members of a set. Instead of enumerating its elements, we specify a set G by first giving a finite number of basic members, then rules of the form "If x is in G, then $f(x)$ is also in G." Canonic systems provide the framework for expressing such a procedure in a formal way.

For example, let $G = \{a^k \mid k \text{ is an even integer}\}$ be a set of strings over the alphabet $\{a\}$. Membership in G can be allowed thus:

1. $aa \in G$
2. If $x \in G$, then $xaa \in G$
3. No other strings are in G.

A canonic system generating G could be written

1. $\vdash aa\underline{G}$
2. $x\underline{G} \vdash xaa\underline{G}$

where "$\vdash aa\underline{G}$" is another form of the membership axiom "$aa \in G$," and "<premise> $\vdash$ <conclusion>" is a contraction of the "If...then..." statement. The fact that no other strings are to be allowed in G is implicit in the way the canonic system is to be interpreted.

Definition 1. A canonic system C is a 5-tuple

$$C = (S, N, P, R, C')$$

where

S = an algebraic system $(A, *_1, \ldots, *_k)$ consisting of a set A and a collection of operations $*_1, \ldots, *_k$ on A.

N = a finite set of variable symbols, usually $\{x, y, \ldots, z\}$.

P = a finite set of predicates, each of which is assigned a unique positive integer called its degree.

R = a finite set of rules of inference, called canons, whose form will be described in Definition 2.

C' = a canonic system which defines the interpretation of the rules of inference in C; its structure and significance will be discussed in the next section.

Previous formulations of canonic systems have tacitly assumed the algebraic system S to be the monoid $(V^*, \cdot)$ of all finite strings over an alphabet V under the operation of concatenation. By making the algebraic system explicit, we are able to specify canonic systems generating sets of integers, real numbers, and so on, as well as sets of strings.

<u>Definition 2</u>. A <u>canon</u> is a rule of inference of the form

$$f_1(x_1,\ldots,x_n)\underline{F_1} \wedge \ldots \wedge f_m(x_1,\ldots,x_n)\underline{F_m} \vdash g(x_1,\ldots,x_n)\underline{G}$$

where F_1, ..., F_m, and G are predicates in P; and f_1, ..., f_m, and g are functions on A formed from $*_1,\ldots,*_k$ by composition. This canon would be read:

> If $f_1(x_1,\ldots,x_n)$ has the property F_1, and ..., and $f_m(x_1,\ldots,x_n)$ has the property F_m, then $g(x_1,\ldots,x_n)$ has the property G.

If $\tilde{Q}$ is interpreted as the extension of a predicate, $\tilde{Q} = \{x \mid Q(x)\}$, then the meaning of a canon may be defined as

$$\tilde{G} = \{u \mid \exists z_1,\ldots,z_n \in A\ [f_1(z_1,\ldots,z_n) \in \tilde{F}_1 \wedge \ldots \wedge f_m(z_1,\ldots,z_n) \in \tilde{F}_m \wedge g(z_1,\ldots,z_n) = u]\}.$$

The quantity $f(x_1,\ldots,x_\ell)\underline{Q}$ is called a <u>remark</u>; the remarks

$f_i(x_1,\ldots,x_n)\underline{F_i}$ are the <u>premises</u> of the canon, and $g(x_1,\ldots,x_n)\underline{G}$ is its <u>conclusion</u>. A remark $f(x_1,\ldots,x_\ell)\ \underline{Q}$ is said to be <u>true</u> iff there exist $z_1,\ldots,z_\ell \in A$ such that $f(z_1,\ldots,z_\ell) \in \tilde{Q}$.

For example, consider this canon over the algebraic system $(V^*, \cdot)$, where V is a finite alphabet and $\cdot$ is the concatenation operation:

$$x\underline{B} \vdash axa\underline{C} \quad .$$

This is interpreted as, "If x has the property B, then the string axa has the property C." Formally,

$$\begin{aligned}\tilde{C} &= \{u \mid \exists\, z \in V^* \ [z \in \tilde{B} \ \wedge \ u = aza]\}\\ &= \{axa \mid x \in \tilde{B}\} \ \text{ if } \tilde{B} \text{ is a non-empty set of strings over } V^*.\end{aligned}$$

Another canon might be

$$xa\underline{D} \wedge y\underline{E} \vdash \langle x, ya\rangle\ \underline{F}$$

whose interpretation is

$$\tilde{F} = \{\langle u, v\rangle \mid \exists\, x,y \in V^* \ [xa \in \tilde{D} \ \wedge \ y \in \tilde{E} \ \wedge \langle u, v\rangle = \langle x, ya\rangle]\},$$

where the notation $\langle u, v\rangle$ denotes the ordered pair of u and v. Informally, we are searching $\tilde{D}$ for a string that ends in "a," then forming an ordered pair from part of it and any string from $\tilde{E}$.

An <u>axiom</u> is a canon with no premises; it is therefore universally true. If a function $g(x_1,\ldots,x_n)$ appears in an axiom, then $x_1,\ldots,x_n$ must be allowed to range over all elements of A. Axioms might be $\vdash 2\underline{N}$ or $\vdash x^2\underline{S}$.

<u>Definition 3</u>. A remark $\omega\ \underline{G}$ is <u>immediately derived</u> from a sequence of remarks $\omega_1\underline{F_1}, \ldots, \omega_m\underline{F_m}$, $\omega_i \in A$, iff

1. $f_1(x_1,\ldots,x_n)\underline{F_1} \wedge \ldots \wedge f_m(x_1,\ldots,x_n)\underline{F_m} \vdash g(x_1,\ldots,x_n)\underline{G}$ is a canon.

2. $\exists\ z_1,\ldots,z_n \in A$ such that $f_i(z_1,\ldots,z_n) = \omega_i$, and $g(z_1,\ldots,z_n) = \omega$.

Definition 4. A proof of a remark r is a finite list of remarks $r_1,\ldots,r_p$ such that

1. r_1 is an axiom,
2. r_i $(1 < i \leq p)$ is either an axiom or a remark immediately derivable from some subset of $\{r_1,\ldots,r_{i-1}\}$,
3. $r_p = r$.

To illustrate these definitions, let us consider the system $C = ((\{a, b\}^*, \cdot), \{x\}, \{A\}, R, C')$, where the canons in R are

$$\vdash b\underline{A}$$
$$x\underline{A} \vdash axa\underline{A}\ .$$

This system contains the single axiom $\vdash b\underline{A}$, so it is trivially provable that $b\underline{A}$. A single application of the canon $x\underline{A} \vdash axa\underline{A}$ yields the conclusion $aba\underline{A}$. Continuing this process, we obtain the proof

$$b\underline{A},\ aba\underline{A},\ aabaa\underline{A},\ \ldots,\ a^n b a^n \underline{A}\ .$$

Thus $\widetilde{A} = \{a^n b a^n \mid n \geq 0\}$, a context-free language.

We occasionally use the phrase "the language (or the set) generated by the system C" to mean the set $\widetilde{G}$ defined by some predicate G in C. Either the appropriate predicate will be obvious, or the selection of G will depend upon the situation in which the result is applied.

Hierarchy of Canonic Systems

A proof in a formal system usually consists of a finite number of repetitions of two operations:

1. Substituting elements of A for variables,

2. Inferring Y from X and $X \rightarrow Y$.

These are operations which can be carried out by a mathematician with paper and pencil. In order to be able to formalize the idea of a proof, Moore and Donovan [3] allowed a canonic system C' to describe the concept of a "proof" in a system C. A hierarchy is thus formed, since a proof in C' is, in turn, described by a system C". For some k, $C^{(k+1)} = C^{(k)}$, so that no new features will be found in higher level systems.

For example, consider the system

$C = ((\{a, b\}^*, \cdot), \{x\}, \{A\}, R, C')$, where R consists of the canons

1. $\vdash b\underline{A}$
2. $x\underline{A} \vdash axa\underline{A}$.

$\tilde{A} = \{b, aba, aabaa, \ldots\}$. The second level system C' will generate the <u>proofs</u> that each of these strings is defined by A. In other words, C' will generate the set

$\{b\underline{A}, b\underline{A};aba\underline{A}, b\underline{A};aba\underline{A};aabaaa\underline{A}, \ldots\}$

where we have used the semicolon only to avoid a collision of punctuation.

This set can be considered a set of strings over the alphabet $\{b, \underline{A}, ;\}$, and as such it may be generated by some canonic system. Define C' to be that system. Its canons describe the effect on a proof of applying each of the rules of C:

1. $\vdash \lambda\ \underline{\text{proof}}$
2. $\alpha\ \underline{\text{proof}} \vdash \alpha;b\underline{A}\ \underline{\text{proof}}$
3. $\alpha\ \underline{\text{proof}} \wedge \beta;\gamma\underline{A}\delta\ \underline{\text{proof}} \vdash \alpha;a\gamma a\underline{A}\delta\ \underline{\text{proof}}$

Then $C' = ((\{;, b, \underline{A}\}^*, \cdot), \{\alpha, \beta, \gamma, \delta\}, \{\text{proof}\}, R, C")$. It is not difficult to see how C' operates. First, for convenience, it defines the null string as a proof. Then, if some proof α has already been derived, α followed by the symbols ";b$\underline{A}$" is still a proof because $\vdash b\underline{A}$ is an axiom of C. Finally, if a proof α exists such that the pattern ";<some string γ> $\underline{A}$" appears within it, α followed by the symbols "; a<some string γ>a $\underline{A}$" is also a proof. This corresponds to applying the canon $\underline{x}\underline{A} \vdash a\underline{x}a\underline{A}$ in C.

This formulation of higher-level systems is considerably simpler than their original description [4]. Since we shall have occasion to use such systems later, this transparency will be beneficial.

Summary

This chapter has been a review of the definitions and terminology associated with canonic systems. We have revised some of the formulations so that they are applicable over arbitrary algebraic systems; this will allow us to write canons describing, for example, sets of integers, as well as sets of strings.

Chapter 2 - Proof Measure Functions

Introduction

This section develops a measure of complexity for sets of strings specified by a canonic systems. Such a measure is related to the time it would take a generator operating from the canonic system to produce an element of the language being described.

Definition 1. A proof of a theorem t in a system C is a finite list of remarks $r_1,\ldots,r_k$ such that

1. r_1 is an axiom
2. r_i $(1 < i \leq k)$ is either an axiom or a theorem immediately derivable from some subset of $\{r_1,\ldots,r_{i-1}\}$.
3. $r_k = t$.

Recall that, in a given canonic system C, the meaning and derivation of a proof is given in the next higher level system C'. Since C' is capable of generating all possible proofs of theorems in C, we make the following definition.

Definition 2. The proof measure function m(C, t) of a theorem t in a canonic system C is the number of remarks in the shortest proof of t. That is, if $r_1, r_2,\ldots,r_k$ is the shortest proof of t in C, then m(C, t) = k.

For example, in a system S containing the canons

1. $\vdash b\underline{A}$
2. $x\underline{A} \vdash axa\underline{A}$

we would have the proof $b\underline{A}$, $aba\underline{A}$, $aabaa\underline{A}$. Therefore m(S, aabaa) = 3 since there is precisely one proof for each string in this system.

Properties of m(C, t)

We may now prove several properties of proof measure functions.

Theorem 1. If t is a theorem of C, then m(C, t) is computable.

Proof. If t is a theorem of C, then there exists a proof of t in C: $r_1, r_2, \ldots, r_k$ such that $t = r_k$. This proof, considered as a string of symbols, is a string derived by the next higher level system C'. Modify C' into C_m so that it keeps count of the length of the proof it is constructing for t. Change the axiom

$$\vdash \lambda \; \underline{\text{proof}}$$

into

$$\vdash \langle\lambda, 0\rangle \; \underline{\text{proof \& length}}.$$

In each canon of C' which places a remark in the proof, replace the predicate proof with a predicate over ordered pairs, proof & length. Thus, if C' has the canon

$$\alpha \; \underline{\text{proof}} \wedge \ldots \vdash \alpha;\beta \quad \underline{\text{proof}},$$

let C_m have the canon

$$\langle\alpha, n\rangle \; \underline{\text{proof \& length}} \wedge \ldots \vdash \langle\alpha;\beta, n+1\rangle \; \underline{\text{proof \& length}}.$$

Finally, let C_m have the canon

$$\langle\alpha;\beta, n\rangle \; \underline{\text{proof \& length}} \vdash \langle\beta, n\rangle \; \underline{\text{theorem \& m(C, t)}}$$

to associate with each t the length of its proof. Then the canonic system C_m computes m(C, t). QED

An Alternative Function

Given that t is a theorem of C, the function m(C, t) gives us a measure of the difficulty of proving t in C. Unfortunately m is a function of each possible theorem. It would be desirable to sacrifice some of the exactness of m in favor of a more macroscopic quantity.

Definition 3. Let $m'(C, n) = \sup \{m(C, t) \mid t$ is provable in C and $|t| = n\}$.

The function m' looks at all proofs of theorems of length n in C and takes the number of steps in the longest proof as its value. Thus, given a theorem t of length n, we can, under certain conditions, say that it may be proven in no more than $m'(C, n)$ steps. By definition, $m'(C, n)$ is the upper bound on the length of proof of a theorem of length n. We now consider the conditions under which it is meaningful to speak of m'.

Theorem 2. $m'(C, n)$ is computable if and only if C generates a recursive set.

Proof. Consider the system C to be over a terminal alphabet T. Generate the set

$$S_n = \{\omega \in T^* \mid \omega = n\},$$

the set of all possible formulas of length n over T. For each $\omega \in S_n$, we can determine whether or not it is a theorem in C since C generates a recursive set. That is, generate

$$S_n' = \{\omega \in S_n \mid \omega \text{ is provable in } C\}.$$

Then, since $m(C, \omega)$ is computable when ω is known to be provable in C,

$$m'(C, n) = \sup \{m(C, \omega) \mid \omega \in S'_n\}.$$

If $m'(C, n)$ is computable, then if any formula α is provable at all, it is provable in no more than $m'(C, |\alpha|)$ steps. Since C contains a finite number of axioms and canons, it is possible to generate all proofs of length 1 (the axioms), length 2 (one canon applied to each axiom), and so on. If the theorem α has not been proven after $m'(C, |\alpha|)$ steps, it will never be. Since the membership question is thus decidable, C must generate a recursive set. QED

There is a property of m' which is significant for its application to complexity measures for programming languages.

<u>Theorem 3</u>. m'(C,n) is not necessarily monotonic increasing in n.

<u>Proof</u>. Consider the counterexample

$\vdash aa\underline{A}$

$x\underline{A} \vdash xaa\underline{A}$

$\vdash a\underline{B}$

$x\underline{B} \vdash xaa\underline{B}$

$x\underline{B} \vdash x\underline{A}$

which generates the proofs

1. $a\underline{B}$, $a\underline{A}$
2. $aa\underline{A}$
3. $a\underline{B}$, $aaa\underline{B}$, $aaa\underline{A}$
4. $aa\underline{A}$, $aaaa\underline{A}$.

Then m'(C,1) = 2, m'(C,2) = 1, m'(C,3) = 3, m'(C,4) = 2, and so on.
<u>QED</u>

This property is entirely expected, for some short theorems may be much more difficult to prove than long ones.

<u>A More Precise Function</u>

The functions m and m' evolve naturally from a consideration of the process of proof in a canonic system. They do provide an indication of the difficulty of deriving a given theorem, but they make the simplifying assumption that one canon is no different from another since each extends the proof one step. But certain rules may contain several premises, each of which must be examined before the conclusion may be drawn. Let us define another function which takes this into account.

<u>Definition 4</u>. The proof measure function n(C, t) of a theorem t in a canonic system C is the total number of premises evaluated in the shortest proof of t. That is, if $r_1,\ldots,r_k$ is the shortest proof of t in C, and if the canons applied during this proof were

$$r_{11} \wedge r_{21} \wedge \cdots \wedge r_{m_1 1} \vdash r_1$$

$$r_{12} \wedge r_{22} \wedge \cdots \wedge r_{m_2 2} \vdash r_2$$

$$\vdots$$

$$r_{1k} \wedge r_{2k} \wedge \cdots \wedge r_{m_k k} \vdash r_k,$$

then $n(C,t) = \sum_{j=1}^{k} m_j$.

Theorem 4. $n(C, t)$ is computable iff t is provable in C.

Proof. The proof is almost identical to that of Theorem 1. Build the canonic system C_m so that, at each step in the proof of t, a count is kept of the number of premises which have been evaluated so far. QED

Definition 5. Let $n'(C, \ell) = \sup\{n(C, t) \mid t \text{ is provable in } C \text{ and } |t| = \ell\}$.

Theorem 5. $n'(C, \ell)$ is computable iff C generates a recursive set.

Proof. Similar to that of Theorem 2. QED

Theorem 6. If p is the number of axioms appearing in the shortest proof of t in C, then $n(C, t) \geq m(C, t) - p$.

Proof. If p axioms appear in the proof, then $m(C, t) = p + x$, where x steps are taken in the proof after the axioms are written down. The number of premise remarks evaluated for each of the axioms is zero; for the x remaining steps, $y \geq x$ premises must be examined since each canon has at least one premise. Thus $n(C, t) = 0 + y$. These three conditions imply $n(C, t) \geq m(C, t) - p$. QED

Theorem 7. If C is a canonic system possessing q axioms, then $n'(C, \ell) \geq m'(C, \ell) - q$. for all ℓ.

Proof. At worst, some theorem of length ℓ may require the use of all q axioms in its proof; in any case, it requires at least one. The conclusion follows from this observation and Theorem 6. QED

Bounds on Measure Functions

Consider a canonic system C generating sets of strings. We would like to relate at least one of the proof measure functions to the structure of the system C in such a way that we could avoid having the function described by only another canonic system. Let

$$D(k) = \{x \mid x \text{ is provable in exactly } k \text{ steps in } C\}.$$

Then, by definition, D(1) is the set of all the axioms of C, D(2) is the set of all theorems provable by applying exactly one canon to the strings in D(1), and so on. Furthermore, let

$$L(k) = \sup \{x \mid x \in D(k)\},$$
$$S(k) = \inf \{x \mid x \in D(k)\}.$$

That is, L(k) is the length of the longest theorem provable in k steps, and S(k) is the length of the shortest.

Assume that the canonic system C has the property that no canon ever "shortens" a theorem. That is, if t is provable in k steps and t' in k + 1, then $|t'| \geqq |t|$. Under these conditions, we can say that $L(k + 1) \geqq L(k)$ and $S(k + 1) \geqq S(k)$ for all k.

Recall that L(k) is the length of the longest string possible after a derivation of k steps. Then the inverse function $L^{-1}(z)$ is the minimum number of proof steps required to generate <u>any</u> theorem of length z; all proofs of fewer steps yield strings shorter than z. Now the proof measure function m'(C, z) is, by definition, the maximum number of proof steps required to generate <u>all</u> theorems of length z. We therefore obtain

<u>Lemma 1</u>. $L^{-1}(z) \leqq m'(C, z)$ if C is non-erasing.

A similar argument may be advanced for the S(k) function. Since $S^{-1}(z)$ is the maximum number of proof steps required to generate all theorems of length z, all longer proofs yield strings of length greater than z. This observation produces

<u>Lemma 2</u>. $m'(C, z) \leqq S^{-1}(z)$ if C is non-erasing.

and

Theorem 8. If C is a canonic system with the property that t provable in k steps and t' in k + 1 implies $|t'| \geq |t|$, then $L^{-1}(z) \leq m'(C, z) \leq S^{-1}(z)$ for all z.

Computation of L(k) and S(k)

Consider a system C generating strings over an alphabet V in which each canon is either an axiom or a rule of the restricted form

$$x\underline{F} \vdash \omega_1 x \omega_2 \underline{G}$$

where $\omega_1, \omega_2 \in V^*$. Let ℓ_o be the length of the longest axiom in C ($\ell_o = \sup \{|\omega| \mid \vdash \omega \text{ is an axiom of } C\}$) and ℓ be the maximum number of symbols added to a derived string by the application of a single canon ($\ell = \sup \{|\omega_1\omega_2| \mid x\underline{F} \vdash \omega_1 x \omega_2 \underline{G} \text{ is a canon of } C\}$). Then we observe that

$$L(1) = \ell_o$$
$$L(2) = L(1) + \ell$$
$$\vdots$$
$$L(k) = L(k-1) + \ell$$

so that

$$L(k) = \ell_o + (k-1)\ell.$$

Similarly, if $s_o = \inf \{|\omega| \mid \vdash \omega \text{ is an axiom of } C\}$ and $s = \inf \{|\omega_1\omega_2| \mid x\underline{F} \vdash \omega_1 x \omega_2 \underline{G} \text{ is a canon of } C\}$ describe the shortest starting string and the least number of symbols added by any canon, then

$$S(k) = s_o + (k-1)s.$$

The inverses are easily found to be

$$L^{-1}(z) = \frac{z - \ell_o + \ell}{\ell}$$

$$S^{-1}(z) = \frac{z - s_o + s}{s}.$$

Noting that canons of the form used never "shorten" a theorem, we may combine these results with Theorem 8 to yield

Theorem 9. If C is a canonic system in which all canons are either of the form $\vdash \omega$ or $x\underline{F} \vdash \omega_1 x \omega_2 \underline{G}$, then

$$\frac{z - \ell_o + \ell}{\ell} \leq m'(C, z) \leq \frac{z - s_o + s}{s},$$ where ℓ_o, ℓ, s_o, s are as defined previously.

The class of canonic systems yielding exactly the regular sets requires canons of the form either $\vdash a\underline{G}$ or $x\underline{F} \vdash xa\underline{G}$, where a is a single symbol from V. Thus $\ell_o = \ell = s_o = s = 1$, and Theorem 2 produces $z \leq m'(C, z) \leq z$. We have proven what may be obvious:

Corollary 1. If C defines a regular set, then $m'(C, z) = z$.

Now consider a system C generating strings over an alphabet V in which each canon is either an axiom or a rule of the restricted form

$$x_1\underline{F}_1 \wedge \cdots \wedge x_n\underline{F}_n \vdash \omega_o x_{i_1} \omega_1 x_{i_2} \cdots \omega_{n-1} x_{i_n} \omega_n \underline{G}$$

where $\omega_i \in V^*$, $x_{i_j} \in \{x_1, \ldots, x_n\}$, and $n > 1$.

Define

1. $r_{max} = \sup \{n \mid x_1\underline{F}_1 \wedge x_2\underline{F}_2 \wedge \cdots \wedge x_n\underline{F}_n \vdash g(x_1, \ldots, x_n)\underline{G}$ is a canon of C$\}$

2. $r_{min} = \inf \{n \mid x_1\underline{F}_1 \wedge x_2\underline{F}_2 \wedge \cdots \wedge x_n\underline{F}_n \vdash g(x_1, \ldots, x_n)\underline{G}$ is a canon of C$\}$

3. $\ell_o = \sup \{ |\omega| \mid \vdash \omega$ is an axiom of C$\}$

4. $\ell = \sup \{ |\omega_o \omega_1 \cdots \omega_n| \mid x_1\underline{F}_1 \wedge \cdots \wedge x_n\underline{F}_n \vdash \omega_o x_{i_1} \omega_1 x_{i_2} \cdots \omega_{n-1} x_{i_n} \omega_n \underline{G}$ is a canon of C$\}$

5. $s_o = \inf \{ |\omega| \mid \vdash \omega$ is an axiom of C$\}$

6. $$s = \inf \{|\omega_o\omega_1 \ldots \omega_n| \mid x_1\underline{F}_1 \wedge \ldots \wedge x_n\underline{F}_n \vdash \omega_o x_{i_1} \omega_1 x_{i_2} \ldots \omega_{n-1} x_{i_n} \omega_n \underline{G} \text{ is a canon of } C \text{ and } n = r_{min}\}.$$

The definitions of ℓ_o, ℓ, s_o, and s are similar to those given in the case where each canon is allowed to have no more than one premise. In addition, we desire to know r_{max} and r_{min}, the maximum and the minimum number of variables appearing in any canon. Again we desire to find L(k) and S(k), the lengths of the longest and the shortest string derivable after a proof of k steps.

By definition, $L(1) = \ell_o$, the length of the longest axiom. Now the longest string derivable by the application of a single canon would be produced when the longest strings possible were assigned to $x_1, \ldots, x_{r_{max}}$ in the canon

$$x_1\underline{F}_1 \wedge x_2\underline{F}_2 \wedge \ldots \wedge x_{r_{max}}\underline{F}_{r_{max}} \vdash \omega_o x_{i_1} \omega_1 x_{i_2} \ldots \omega_{n-1} x_{i_{r_{max}}} \omega_{r_{max}} \underline{G}$$

and $|\omega_o\omega_1 \ldots \omega_{r_{max}}| = \ell$. Such a canon may or may not exist in C, but no canon in C produces strings growing at a faster rate. Thus we observe that

$$L(1) = \ell_o$$
$$L(2) = r_{max}L(1) + \ell$$
$$\vdots$$
$$L(k) = r_{max}L(k-1) + \ell$$

so that

$$L(k) = \ell_o(r_{max})^{k-1} + \ell\sum_{i=0}^{k-2}(r_{max})^i$$
$$= \ell_o(r_{max})^{k-1} + \ell\left(\frac{(r_{max})^{k-1} - 1}{r_{max} - 1}\right)$$

Solving this equation for k to produce the inverse function, we obtain

$$L^{-1}(z) = 1 + \log_{r_{max}}\left(\frac{z(r_{max} - 1) + \ell}{\ell_o}\right) \quad \text{where } r_{max} > 1.$$

We now desire to find an expression for S(k), the length of the shortest string derivable after k applications of canons of the form

$$x_1 \underline{F}_1 \wedge \ldots \wedge x_n \underline{F}_n \vdash \omega_o x_{i_1} \omega_1 x_{i_2} \ldots \omega_{n-1} x_{i_n} \omega_n \underline{G} .$$

In this canon, we must substitute strings for n variables. If the length of the string produced as a conclusion is to be as short as possible, each variable should be assigned the shortest possible string. However, <u>some</u> variable must refer to a string produced in the previous proof step, or there would be no reason to have that step. This would contradict our assumption that all proofs are of minimum length. We thus produce the shortest permissible result when one string x_p is assigned the length S(k-1) and n-1 have the shortest possible length, s_o. Once again, there may or may not be a canon which allows this, but all other substitutions will produce longer conclusions.

In addition, we desire to use the canon with the least number of variables in its conclusion. This canon, by definition, contains constant strings of total length s. For sufficiently large string lengths, the effect of concatenating several long strings is greater than just adding a constant number of symbols each time. Thus we expect this analysis of S(k) to be valid for only large k.

The expression we have justified for S(k) is

$$S(k) = S(k-1) + s_o(r_{min} - 1) + s .$$

The solution of this equation is

$$S(k) = (k-1)(s_o r_{min} + s) - (k-2)s_o ,$$

whose inverse is

$$S^{-1}(z) = \frac{z + s_o(r_{min} - z) + s}{s_o(r_{min} - 1) + s} .$$

Combining these results, we obtain

Theorem 10. If C is a canonic system in which all canons are either of the form $\vdash \omega$ or $x_1\underline{F}_1 \wedge \ldots \wedge x_n\underline{F}_n \vdash \omega_o x_{i_1} \omega_1 x_{i_2} \ldots \omega_{n-1} x_{i_n} \omega_n \underline{G}$, then

$$1 + \log_{r_{max}} \left(\frac{z(r_{max}-1) + \ell}{\ell_o} \right) \leq m'(C, z) \leq \frac{z + s_o(r_{min}-2) + s}{s_o(r_{min}-1) + s}$$

for sufficiently large z, $r_{max} > 1$, and s_o, s, ℓ_o, and ℓ as defined previously.

The class of canonic systems yielding exactly the context-free languages requires canons of the form either $\vdash a\underline{G}$ or $x\underline{A} \wedge y\underline{B} \vdash xy\underline{G}$, where a is a single symbol from an alphabet V [4]. Then $s_o = \ell_o = 1$, $s = \ell = 0$, and $r_{min} = r_{max} = 2$, and Theorem 10 reduces to

Corollary 2. If C defines a context-free language in standard form, then $1 + \log_2 z \leq m'(C, z) \leq z$.

Summary

We have defined four functions measuring the complexity of a derivation in a canonic system. These functions were shown to have certain properties, and relationships among them were derived. For canons in which only the identity function appears in the premises, we derived bounds on some of these functions in terms of the structure of the structure of the system.

There is a question of interest which we have not settled: Given a system C, is it possible to find a system C* such that C* is equivalent to C, but $m'(C^*, \ell) < m'(C, \ell)$ for all ℓ? In other words, is it possible to "speed up" a canonic system in the same way that a Turing machine can be speeded up? The answer to this question will determine the nature of a lower bound on our measure functions.

Chapter 3 - Inverse Systems and Compiling

Introduction

A canonic system C is a specification of the members of a set and, as such, gives rules for generating successive elements of that set. Because of its similarity to the parsing problem, the inverse process is more interesting: given a string ω, how could it have been generated by C? The operation is one of analysis rather than synthesis.

An algorithm which accepts a canonic-system description of a language and analyzes an input string based on this description exists [1]. We go one step further and create from C a new canonic system C^{-1} which exactly describes the recognition procedure for strings generated by C.

For example, consider the language $\mathcal{A} = \{a^n b a^n \mid n \geq 0\}$ defined by the system

$$\vdash b\underline{A}$$
$$x\underline{A} \vdash axa\underline{A} \; .$$

A proof in this system C that $aabaa\underline{A}$ is

$$b\underline{A},\ aba\underline{A},\ aabaa\underline{A} \; .$$

Consider the system

$$axa\underline{A} \vdash x\underline{A}$$
$$b\underline{A} \vdash I$$

where the conclusion I indicates an arbitrary terminating condition. When this system is presented with an axiom $\vdash \omega$ such that ω is provable in C, it proceeds to decompose that string. For example, if we include the axiom $\vdash aabaa\underline{A}$, the system becomes

$$\vdash aabaa\underline{A}$$
$$axa\underline{A} \vdash x\underline{A}$$
$$b\underline{A} \vdash I$$

and generates the proof

$$aabaa\underline{A},\ aba\underline{A},\ b\underline{A},\ I \; .$$

Apart from the end-of-proof indicator I, this is simply the proof in C written backwards. The system we have just presented is C^{-1}, the inverse of C.

Construction of C^{-1} From C

The relationship between the proofs in C^{-1} and C suggests a procedure for obtaining the canons of C^{-1} from those of C: interchange the premise and the conclusion. Thus if C has the canon

$$x\underline{A} \vdash axa\underline{A} ,$$

C^{-1} should have

$$axa\underline{A} \vdash x\underline{A} .$$

This procedure fails if the original canon had multiple premises, for the inversion of the general canon

$$f_1(x_1,\ldots,x_n)\underline{F}_1 \quad \wedge\ldots\wedge \quad f_m(x_1,\ldots,x_n)\underline{F}_m \vdash g(x_1,\ldots,x_n)\underline{G}$$

would produce

$$g(x_1,\ldots,x_n)\underline{G} \vdash f_1(x_1,\ldots,x_n)\underline{F}_1 \quad \wedge\ldots\wedge \quad f_m(x_1,\ldots,x_n)\underline{F}_m \quad ,$$

and we do not define the meaning of a canon with multiple conclusions. (There is no way of indicating in a proof that several conclusions must hold simultaneously.) Thus only systems C in which each canon has no more than one premise possess an inverse C^{-1} by this construction. It is the case, however, that any arbitrary system can be reduced to this form, and this we now demonstrate.

Normal-Form Systems

Given an arbitrary system C, it is possible to derive a system C_N such that

1. Each canon in C_N has a single premise.
2. A remark t is provable in C_N iff t is provable in C.

The general canon we must consider is

$$f_1(x_1,\ldots,x_n)\underline{F_1} \quad \wedge\ldots\wedge \quad f_m(x_1,\ldots,x_n)\underline{F_m} \vdash g(x_1,\ldots,x_n)\underline{G}$$

where

$$f_i\colon \ A^n \to A^{d_i}$$

since the canon is over the algebraic system $(A, *_1,\ldots,*_k)$. We shall call d_i the degree of f_i; the result of applying f_i is a d_i-tuple of elements from A, and F_i must be a predicate over d_i-tuples. For convenience, let us abbreviate this canon as simply

$$f_1\underline{F_1} \quad \wedge\ldots\wedge \quad f_m\underline{F_m} \vdash g\underline{G} \ .$$

The following algorithm produces C_N from C:

1. If C has either the axiom $\vdash \omega\underline{G}$ or a canon $f\underline{F} \vdash g\underline{G}$ with a single premise, then retain these rules in C_N.

2. If C has the canon

   $$f_1\underline{F_1} \quad \wedge\ldots\wedge \quad f_m\underline{F_m} \vdash g\underline{G} \ ,$$

   then let C_N have the canon

   $$\langle f_1,\ldots,f_m\rangle \underline{F_1\ldots F_m} \vdash g\underline{G}$$

   and the new predicate $F_1\ldots F_m$ of degree $(\sum_{i=1}^{m} d_i)$.

3. If C_N has the canons

   $$\langle r_1,\ldots,r_p\rangle \ \underline{R} \vdash u\underline{R'}$$
   $$\langle s_1,\ldots,s_q\rangle \ \underline{S} \vdash v\underline{S'} \ ,$$

   then let C_N also have the canon

   $$\langle r_1,\ldots,r_p,s_1,\ldots,s_q\rangle \ \underline{RS} \vdash \langle u,\ v\rangle \ \underline{R'S'}$$

   and the new predicates RS of degree (degree R + degree S) and R'S' of degree (degree R' + degree S'). Repeat step 3 for each pair R'S' describing a predicate created by rules 2 and 3. Since C has only a finite number of canons, this process will terminate.

4. No other canons are in C_N.

This algorithm reduces several premises to a function of higher degree. Rule 1 retains all those canons which already have only a single premise or none at all. Rule 2 reduces a canon with several premises to the desired form by combining the d_i results of each function f_i $(1 \le i \le m)$ into one $(\sum_{i=1}^{m} d_i)$-tuple. When an n-tuple is evaluated, all components must be shown true, so the effect of requiring the proof of all the original premise remarks is achieved. When canons are combined in this way, new predicates are created in the premises, and Rule 3 provides the canons to describe these. This rule provides for the interaction between objects described by several canons.

For example, let C be the system

$\vdash a\underline{A}$
$\vdash b\underline{B}$
$\vdash c\underline{C}$
$x\underline{A} \vdash ax\underline{A}$
$x\underline{B} \vdash bx\underline{B}$
$x\underline{C} \vdash cx\underline{C}$
$x\underline{A} \wedge y\underline{B} \vdash xy\underline{D}$
$x\underline{C} \wedge y\underline{D} \vdash yxy\underline{E}$

which generates $\mathcal{E} = \{a^n b^m c^k a^n b^m \mid n,m,k \ge 1\}$. By Rule 1 of the algorithm, C_N contains the canons

$\vdash a\underline{A}$
$\vdash b\underline{B}$
$\vdash c\underline{C}$
$x\underline{A} \vdash ax\underline{A}$
$x\underline{B} \vdash bx\underline{B}$
$x\underline{C} \vdash cx\underline{C}$.

By Rule 2, C_N contains

$<x, y> \underline{AB} \vdash xy\underline{D}$
$<x, y> \underline{CD} \vdash yxy\ \underline{E}$

corresponding to the two canons with multiple premises. Rule 3 provides definitions for the sets $\underline{AB}$ and $\underline{CD}$; applying it to the canons in C_n which define $\underline{A}$ and $\underline{B}$ separately produces the set of rules

$$\begin{aligned}
\langle x, y\rangle\underline{AB} &\vdash \langle ax, by\rangle\underline{AB}\\
x\underline{A} &\vdash \langle ax, b\rangle\underline{AB}\\
y\underline{B} &\vdash \langle a, by\rangle\underline{AB}\\
&\vdash \langle a, b\rangle\underline{AB}
\end{aligned}$$

The definition of $\underline{CD}$ generates the canons

$$\begin{aligned}
\langle x, y, z\rangle\underline{ABC} &\vdash \langle cz, xy\rangle\underline{CD}\\
\langle x, y\rangle\underline{AB} &\vdash \langle c, xy\rangle\underline{CD}
\end{aligned}$$

A new predicate, $\underline{ABC}$, has been generated, and it must be defined by applying Rule 3 again:

$$\begin{aligned}
\langle x, y, z\rangle\underline{ABC} &\vdash \langle ax, by, cz\rangle\underline{ABC}\\
\langle y, z\rangle\underline{BC} &\vdash \langle a, by, cz\rangle\underline{ABC}\\
\langle x, y\rangle\underline{AB} &\vdash \langle ax, by, c\rangle\underline{ABC}\\
\langle x, z\rangle\underline{AC} &\vdash \langle ax, b, cz\rangle\underline{ABC}\\
x\underline{A} &\vdash \langle ax, b, c\rangle\underline{ABC}\\
y\underline{B} &\vdash \langle a, by, c\rangle\underline{ABC}\\
z\underline{C} &\vdash \langle a, b, c\rangle\underline{ABC}
\end{aligned}$$

The predicates $\underline{BC}$ and $\underline{AC}$ must be defined, again by Rule 3:

$$\begin{aligned}
\langle x, y\rangle\underline{BC} &\vdash \langle xb, yc\rangle BC\\
x\underline{B} &\vdash \langle xb, c\rangle\underline{BC}\\
y\underline{C} &\vdash \langle b, yc\rangle\underline{BC}\\
&\quad \langle b, c\rangle\underline{BC}
\end{aligned}$$

$$\begin{aligned}
\langle x, y\rangle\underline{AC} &\vdash \langle xa, yc\rangle\underline{AC}\\
x\underline{A} &\vdash \langle xa, c\rangle\underline{AC}\\
y\underline{C} &\vdash \langle a, yc\rangle\underline{AC}\\
&\vdash \langle a, c\rangle\underline{AC}
\end{aligned}$$

No new predicates were created in this step, so the generation of C_N is complete.

Consider the proof of aabbbcaabbb$\underline{E}$ in the original system C:

$$b\underline{B},\ a\underline{A},\ bb\underline{B},\ aa\underline{A},\ bbb\underline{B},\ c\underline{C},\ aabbb\underline{D},\ aabbbcaabbb\underline{E}\ .$$

The system C_N simulates this derivation as:

$$b\underline{B},\ \langle a,\ bb\rangle\ \underline{AB},\ \langle aa,\ bbb\rangle\ \underline{AB},\ \langle c,\ aabbb\rangle\ \underline{CD},\ aabbbcaabbb\underline{E}\ .$$

We now show that C_N generates the same sets as C.

Theorem 1. A remark t is provable in C_N iff t is provable in C. Furthermore, the proof in C_N is effectively obtainable from that in C, and conversely.

Proof. To show that t provable in C implies t provable in C_N, we will use induction on the length of the proof in C.

Basis: If t is provable in one step in C, then t, being an axiom, is also provable in one step in C_N. Similarly, if t is provable in two steps in C, it is the result of applying a single-premise canon to an axiom in C. By Rule 1 forming C_N from C, these operations may also be performed in C_N.

Induction: Assume all theorems provable in less than or equal to k steps in C are also provable in C_N. A proof of length k+1 of a theorem t in C is a list of remarks $r_1,\dots,r_{k+1}$ such that

1. r_1 is an axiom.
2. r_i $(1 < i \le k+1)$ is either an axiom or a remark immediately derivable from some subset of $\{r_1,\dots,r_{i-1}\}$.
3. $r_{k+1} = t$.

t is not an axiom of C since it is assumed to be provable in no less than k+1 steps. Therefore r_{k+1} is immediately derived from some subset $\{\omega_1\underline{F}_1,\dots,\omega_m\underline{F}_m\}$ of $\{r_1,\dots,r_k\}$ by the application of a canon

$$f_1(x_1,\dots,x_n)\underline{F}_1 \wedge \dots \wedge f_m(x_1,\dots,x_n)\underline{F}_m \vdash g(x_1,\dots,x_n)\underline{G}$$

where there exist $z_1,\dots,z_n \in A$ such that $f_i(z_1,\dots,z_n) = \omega_i$ and $g(z_1,\dots,z_n)\underline{G} = r_{k+1}$. Furthermore, each $\omega_i\underline{F}_i$ is provable in less than or equal to k steps.

From these facts about the proof in C, it is possible to construct the proof in C_N. When C derives r_{k+1} by applying the canon

$$f_1(x_1,\ldots,x_n)\underline{F}_1 \wedge \ldots \wedge f_m(x_1,\ldots,x_n)\underline{F}_m \vdash g(x_1,\ldots,x_n)\underline{G} \ ,$$

C_N uses

$$<f_1,\ldots,f_m> \underline{F_1\ldots F_m} \vdash g(x_1,\ldots,x_n)\underline{G}$$

generated by Rule 2 forming C_N from C. Each component of the m-tuple is provable in less than or equal to k steps in C; by the inductive assumption, each is therefore also provable in C_N, as is the entire m-tuple. Then, as long as the predicate $F_1\ldots F_m$ appears in the conclusion of some canon, r_{k+1} may also be proven in C_N. But Rule 3 of the construction algorithm insures that all predicates appearing in premises appear on the right-hand side of _some_ canon, so r_{k+1} is indeed provable in C_N.

Thus the existence of a proof of arbitrary length of t in C implies the existence of a proof of t in C_N.

It is now necessary to show that t provable in C_N implies t provable in C. The proof of t in C_N is a list of remarks $s_1,\ldots,s_\ell$ such that

1. s_1 is an axiom.
2. s_i $(1 < i \leq \ell)$ is immediately derivable from s_{i-1}.
3. $s_\ell = t$.

The following procedure reconstructs the proof of t in C. Consider, with no loss of generality, each remark s_i to be of the form

$$<q_1,\ldots,q_p> \underline{Q_1\ldots Q_p} \quad \text{where } p \geqq 1.$$

In the construction of C_N from C, multiple premises in canons are reduced to components of one n-tuple. If we know the predicates in C, we can look at such a remark in the proof in C_N and decompose the one term into several. That is, the remark

$$<q_1,\ldots,q_p> \underline{Q_1 \ldots Q_p}$$

in C_N becomes the set of remarks

$$q_1 \underline{Q}_1, \ \ldots, \ q_p \underline{Q}_p$$

in C, where each Q_i may be a predicate of arbitrary degree. Continuing this procedure on each remark in the proof $s_1,\ldots,s_\ell$ of t in C_N produces the proof of t in C. Therefore t provable in C_N implies t provable in C. QED

Measure Functions of Normal Systems

Theorem 2. $n(C_N, t) = m(C_N, t)$ for all t provable in C_N, and $n'(C_N, \ell) = m'(C_N, \ell)$ for all ℓ.

Proof. These are trivially true since each step in a derivation in C_N requires the evaluation of exactly one premise. QED

Theorem 3. If ν is the maximum number of premises per canon in C, then $m(C_N, t) \le m(C, t) \le \dfrac{\nu^{m(C_N, t)} - 1}{\nu - 1}$ for all t provable in C.

Proof. Figures 1 and 2 illustrate the correspondence between derivations in C_N and C. We observe that each node in the C_N tree expands into at least one and at most ν nodes in the C tree. Both trees have the same number of levels (the number of nodes in the C_N tree is equal to the length of the longest path in the C tree), so that if the top level in C_N expands into ν nodes, the next lower level could contain as many as $\nu \cdot \nu = \nu^2$, and so on. Thus

$$m(C, t) \geqq m(C_N, t)$$

$$m(C, t) \leqq \sum_{i=0}^{m(C_N, t)-1} \nu^i$$

so that

$$m(C_N, t) \le m(C, t) \le \frac{\nu^{m(C_N, t)} - 1}{\nu - 1} .$$

QED

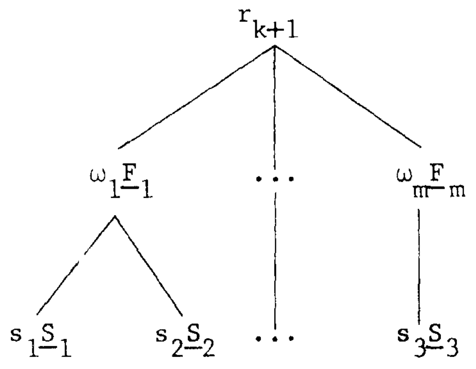

Figure 1 - Derivation in System C

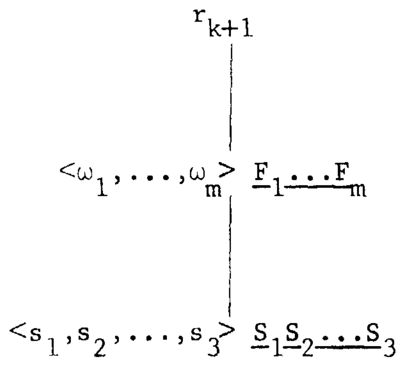
r_{k+1}
$<\omega_1,\ldots,\omega_m> \underline{F_1\ldots F_m}$
$<s_1,s_2,\ldots,s_3> \underline{S_1 S_2\ldots S_3}$

Figure 2 - Derivation in System C_N

Inverses of Arbitrary Systems

Recall that normal-form systems were introduced to show that any canonic system C can be reduced to another system C_N in which each canon has only one premise. The difficulty in constructing the inverse system C^{-1} from an arbitrary C arose exactly in the consideration of canons with multiple premises. The normal-form theorem resolves this problem by always providing an equivalent system of the required form. We may now formally define an inverse system.

C^{-1} is constructed from a system C in normal form by the following algorithm:

1. If C has the axiom $\vdash \omega \underline{G}$, then let C^{-1} have the canon $\omega \underline{G} \vdash I$, where I is an arbitrary end-of-proof indication.

2. If C has the canon $f(x_1,\ldots,x_n)\underline{F} \vdash g(x_1,\ldots,x_n)\underline{G}$, then let C^{-1} have the canon $g(x_1,\ldots,x_n)\underline{G} \vdash f(x_1,\ldots,x_n)\underline{F}$.

Let $C \cup \{\alpha\}$ denote the system C with the added canon α. The inverse system has the following property.

<u>Theorem 4</u>. t is provable in C iff I is provable in $C^{-1} \cup \{\vdash t\}$. Furthermore, the proof in C^{-1} is effectively obtainable from that in C, and conversely.

<u>Proof</u>. If t is provable in C, then there exists a proof $r_1,\ldots,r_k$ of t in C_N such that

1. r_1 is an axiom.
2. r_i $(1 < i \leq k)$ is immediately derivable from r_{i-1}.
3. $r_k = t$.

If the axiom $\vdash t$ is given to C^{-1}, the only proof possible must begin t,... since C^{-1} has no other axioms. In the proof of t is C_N, $t = r_k$ is derived from r_{k-1} by applying a canon

$$f(x_1,\ldots,x_n)\underline{F} \vdash g(x_1,\ldots,x_n)\underline{G}$$

where there exist $z_1,\ldots,z_n \in A$ such that $f(z_1,\ldots,z_n)\underline{F} = r_{k-1}$ and

$g(z_1,\ldots,z_n)\underline{G} = r_k$. By the construction of C^{-1} from C_N, the inverse canon

$$g(x_1,\ldots,x_n)\underline{G} \vdash f(x_1,\ldots,x_n)\underline{F}$$

is available in C^{-1}, so the proof in C^{-1} can continue: $t, r_{k-1},\ldots$. This process may be repeated until the axiom r_1 is reached: $t, r_{k-1},\ldots,r_1$. Then, since the canon $r_1 \vdash I$ exists in the inverse system if $\vdash r_1$ was an axiom in C_N, the proof terminates $t,r_{k-1},\ldots,r_1,I$, proving I.

If I is provable in $C^{-1} \cup \{\vdash t\}$, then there exists a proof $t,s_1,\ldots,s_{k-1},I$ in that system. The last canon applied must have been $s_{k-1} \vdash I$, since these are the only canons which generate I. By the construction of C^{-1} from C_N, $\vdash s_{k-1}$ must have been an axiom of C_N. At each step in the proof where s_{i+1} is deduced from s_i by the application of a canon

$$g(x_1,\ldots,x_n)\underline{G} \vdash f(x_1,\ldots,x_n)\underline{F} ,$$

the inverse rule

$$f(x_1,\ldots,x_n)\underline{F} \vdash g(x_1,\ldots,x_n)\underline{G}$$

exists in C_N and may be applied. After k-1 steps, we arrive at a proof $s_{k-1},\ldots,s_1$ in C_N. A final application of the inverse of the canon which produced t from s_1 produces the desired proof. QED

Measure Functions of Inverse Systems

The proof of Theorem 4 can be extended to indicate the relationship between the length of the proof in C^{-1} and that in C_N.

Theorem 5. $m(C^{-1} \cup \{\vdash t\}, I) = m(C_N, t) + 1$ for all t provable in C_N.

Proof. The result follows immediately from a consideration of the construction of Theorem 4. The proof in the inverse system is simply the reverse, remark by remark, of the proof in C_N, plus the end-of-proof signal I. QED

Corollary 1. $m'(C^{-1} \cup \{\vdash t\}, |I|) = m'(C_N, |t|) + 1$ for all t provable in C_N.

Proof. Immediate.

An interpretation of these results is beneficial at this point. The corollary states that if t is known to be provable in C_N in no more than $k = m'(C_N, |t|)$ steps, then the inverse system C^{-1}, given the single axiom $\vdash t$, can reproduce the proof in no more than k+1 steps.

Theorem 3 of this chapter states that $m'(C_N, \ell) \leq m'(C, \ell)$ for all ℓ. This fact and Corollary 1 produce

Corollary 2. $m'(C^{-1} \cup \{\vdash t\}, |I|) \leq m'(C, |t|) + 1$ for all t provable in C.

Proof. Immediate.

Inverse Systems As Recognizers

Suppose the canonic system C describes a programming language. The system C^{-1} behaves like an analyzer for that language since, given any string derivable in C, C^{-1} can produce a parse of that string. The set of all strings derivable in C is simply the set of all legal programs in the language described by C. Once the m' function is known for C, it is possible to characterize the recognition process of a legal program in that language by applying Corollary 2.

While $m'(C, \ell) + 1$ is an upper bound on the number of steps in the parse in C^{-1}, nothing we have said helps construct this parse. C^{-1}, like any canonic system, is a non-deterministic statement of transformations which may be applied at any step in a proof. In general, more than one canon may be applicable, and the correct rule to choose may not be immediately obvious.

Recursive Sets And Programming Languages

Canonic systems in their most general form define recursively enumerable sets. Given a particular string ω and system C, it is not possible to determine in general whether ω could eventually be generated by C. A recognizer based on a recursively enumerable language may never halt for some input strings, and this situation is undesirable for practical computing systems.

One solution is to allow only language definitions which define recursive sets. It is undecidable, however, whether an arbitrary canonic system has that property (Appendix I). Another approach is to restrict the form of the system so that its language is known to be recursive. The correspondence with Chomsky's grammars does this, but the forms produced do not resemble the definitions of programming languages.

Let us examine the question from a different point of view: we are trying to insure that a compiler presented with an invalid program will not loop endlessly. In theory, a machine capable of examining _any_ arbitrary string and deciding acceptance or rejection must be based on a recursive language. In practice, compilers are not usually asked to process _completely_ arbitrary inputs. A source program may be correct except for the duplication of a statement label or a missing declaration.

It is possible then to write the language description in such a way that common errors are detected. In effect, we are defining a new language, similar to the desired one except for well-defined errors. We still cannot guarantee our compiler will halt for arbitrary inputs, but "almost correct" programs will not cause infinite looping.

For example, consider generating DO loops in Fortran of the simplified form

```
          DO <label> <integer variable> = <constant>, <constant>
          .
          .
          .
<label>   CONTINUE
```

Most Fortran systems impose the restriction that the value of the DO index may not be changed within the loop; that is,

```
   DO 5 J = 1, 10
   J = 20
 5 CONTINUE
```

is illegal. Assume we have defined in a canonic system the predicate not in such that <x, y> not in is true iff the element x is not contained in the set y, and <x, y> in if it is. A specification for a legal DO loop is:

$$\begin{array}{l}\langle x, y\rangle\ \underline{\text{set of Fortran statements with assigned variables}}\\ \wedge\ z\ \underline{\text{integer variable}} \wedge \langle z, y\rangle\ \underline{\text{not in}} \wedge\ w\ \underline{\text{statement label}}\\ \wedge\ i\ \underline{\text{integer constant}} \wedge\ j\ \underline{\text{integer constant}}\end{array}$$

$$\vdash \begin{array}{l}\quad \text{DO } w\ z = i, j\\ \quad x\\ w \text{ CONTINUE}\end{array} \qquad \underline{\text{legal DO loop}}\ .$$

To detect the case of a program redefining the value of the index, we might write:

$$\begin{array}{l}\langle x, y\rangle\ \underline{\text{set of Fortran statements with assigned variables}}\\ \wedge\ z\ \underline{\text{integer variable}} \wedge \langle z, y\rangle\ \underline{\text{in}} \wedge\ w\ \underline{\text{statement label}}\\ \wedge\ i\ \underline{\text{integer constant}} \wedge\ j\ \underline{\text{integer constant}}\end{array}$$

$$\vdash \begin{array}{l}\quad \text{DO } w\ z = i, j\\ \quad x\\ w \text{ CONTINUE}\end{array} \qquad \underline{\text{program attempting to redefine DO index}}\ .$$

Thus a program with a syntactic error is recognized as such, and we may be assured that the compiler will not loop on receiving an incorrect DO loop. The language defined by the entire canonic system may not be recursive in the formal sense, yet we may be able to determine membership in some set for most input strings we are likely to encounter. The language description might be called "almost recursive," and a recognizer based on that description would halt for "almost all" inputs.

Summary

This chapter has presented two main results. First, we showed how an arbitrary canonic system C could be reduced to an equivalent system C_N which contained only single-premise canons. From this normal form, we developed the idea of an inverse system C^{-1} which characterized the recognition of strings generated by C. In addition, we computed the measure functions of both C_N and C^{-1} and showed the relationship between them. Finally, we developed a technique for developing "almost recursive" language descriptions so that recognizers would not be likely to loop on most inputs.

Inverse systems as we have defined them do not specify parsing algorithmns for the languages they recognize. An open question is whether inverse systems can be so extended, and under what conditions.

Appendix I - Systems Describing Recursive Sets

Introduction

A canonic system can specify the elements of any recursively enumerable set. However, from at least a theoretical point of view, it is desirable to know under which conditions the set generated is also recursive. Doyle [4] demonstrated equivalences between canonic systems generating sets of strings and the formal languages of Chomsky [2]. Since type 1, 2, and 3 languages are recursive sets, the related canonic systems also generate recursive sets.

In this section we consider canonic systems generating set of natural numbers. It is difficult to describe functions of strings, but any operation on natural numbers can be characterized by some partial recursive function. First, recall three definitions.

A set Q is recursively enumerable if and only if it is the range of some total recursive function. Equivalently, there exists some Turing machine which will generate all members of Q in some order.

The characteristic function $C_Q(x)$ of a set Q is defined to be

$$C_Q(x) = \begin{cases} 1 & \text{if } x \in Q \\ 0 & \text{if } x \notin Q. \end{cases}$$

A set Q is recursive if and only if its characteristic function is total recursive. In other words, there exists a Turing machine which, given any x, will decide either $x \in Q$ or $x \notin Q$. A set is recursive if and only if both it and its complement are recursively enumerable.

Canonic Systems and Recursive Sets

Let N be the set of natural numbers, and consider canons over the algebraic system (N, +). We now prove two theorems relating canonic systems and recursive sets of natural numbers.

<u>Theorem 1</u>. If a set $Q \subseteq N$ is recursive, then it can be generated by a canonic system in which the only canon describing Q is of the form

$$f(x)\underline{P} \vdash x\underline{Q}$$

where $f(x)$ is a total recursive function.

<u>Proof</u>. Since Q is recursive, its characteristic function

$$C_Q(x) = \begin{cases} 1 & \text{if } x \in Q \\ 0 & \text{if } x \notin Q \end{cases}$$

is total recursive. Let $f(x) = C_Q(x)$, and include the axiom $\vdash 1\underline{P}$. Then the conclusion $x\underline{Q}$ in the canon

$$C_Q(x)\underline{P} \vdash x\underline{Q}$$

is provable if and only if $x \in N$ and $C_Q(x) = 1$, since only then is the second premise $C_Q(x)\underline{P}$ satisfied. Therefore $x\underline{Q}$ is provable if and only if $x \in Q$, as desired. The entire canonic system generating A is then

$$\vdash 1\underline{P}$$
$$C_Q(x)\underline{P} \vdash x\underline{Q}$$

<u>QED</u>

The converse of this theorem is true, but a stronger result is possible.

<u>Theorem 2</u>. Let the set $\mathfrak{G}$ be defined only by canons of the form

$$f_1(x_1,\ldots,x_n)\underline{F}_1 \wedge \ldots \wedge f_m(x_1,\ldots,x_n)\underline{F}_m \vdash g(x_1,\ldots,x_n)\underline{G}.$$

Then $\mathfrak{G}$ is recursive if

1. g^{-1} exists and is a total function
2. $f_1,\ldots,f_m$ are total
3. $\mathfrak{F}_1,\ldots,\mathfrak{F}_m$ are recursive sets
4. $g(x_1,\ldots,x_n)$ depends on each variable x_i in a non-trivial way.

Proof. Assume, with no loss of generality, that the range of g is the set of k-tuples $z = <z_1,\ldots,z_k>$. To show that $\mathfrak{G}$ is recursive, we show how to determine whether $z \in \mathfrak{G}$ or $z \notin \mathfrak{G}$ for an arbitrary z.

If g^{-1} exists and is total, for an arbitrary k-tuple $z = <z_1,\ldots,z_k>$ we can compute $g^{-1}(z_1,\ldots,z_k) = <x_1,\ldots,x_n>$. Every variable that appears in the premises is some x_j in $g^{-1}(z) = <x_1,\ldots,x_n>$ since g is assumed to depend upon all the variables non-trivially. Then, for each function $f_1,\ldots,f_m$, we can compute $f_i(x_1,\ldots,x_n)$ since each f_i is total. Then, knowing the values of $f_i(x_1,\ldots,x_n)$, we can determine membership in $\mathfrak{F}_1,\ldots,\mathfrak{F}_m$ since these are recursive sets.

Formally, the characteristic function of $\mathfrak{G}$ is computable for all z as

$$C_{\mathfrak{G}}(z) = C_{\mathfrak{F}_1}(f_1(x_1,\ldots,x_n)) \cdot \ldots \cdot C_{\mathfrak{F}_m}(f_m(x_1,\ldots,x_n))$$

where $g^{-1}(z) = <x_1,\ldots,x_n>$ and $C_{\mathfrak{F}_1},\ldots,C_{\mathfrak{F}_m}$ are the characteristic functions of the recursive sets $\mathfrak{F}_1,\ldots,\mathfrak{F}_m$. Since $C_{\mathfrak{G}}(z)$ is total recursive, $\mathfrak{G}$ is a recursive set. QED

The General Decision Problem

Theorems 1 and 2 relate certain canonic systems and recursive sets. This is not to say that recursive sets can only be described by systems of those forms, nor that any system with canons of another form can not generate a recursive set. We wish to sketch two results of Mandl [5] which explain the apparent inadequacy of our structure theorems.

Theorem 3. It is undecidable whether an arbitrary canonic system generates a recursive set.

Proof. A canonic system defines, in general, a recursively enumerable set, and it is undecidable whether an arbitrary recursively enumerable set is also recursive. QED

Theorem 4. There is no class of canonic systems such that

1. Every system in the class generates a recursive set.
2. Every recursive set is generated by some member of the class.

Proof. The proof requires a diagonalization argument which we do not wish to reproduce here.

Summary

For canonic systems generating sets of natural numbers, we have derived restricted forms of canons which guarantee that the system will generate a recursive set. The general question of deciding whether an arbitrary system generates a recursive set has already been shown undecidable.

Appendix II - Equivalent Systems

Introduction

In previous papers on canonic systems, only canons describing functions of strings have been considered. That is, the algebraic system has been assumed to be the monoid $(V^*, \cdot)$ of all finite strings over an alphabet V under the operation of concatenation. Under these conditions, the functions involved in the definition of a canon take on a particularly simple form.

We desire the most general function $f(x_1,\ldots,x_n)$ from n-tuples of strings into k-tuples. Since concatenation of variables and constants is the only operation available, that function is

$$f(x_1,\ldots,x_n) = \langle \omega_{01} z_{11} \omega_{11} \ldots z_{n_1 1} \omega_{n_1 1}, \ \ldots \ , \omega_{0k} z_{1k} \omega_{1k} \ldots z_{n_k k} \omega_{n_k k} \rangle$$

where $z_{ij} \in \{x_1,\ldots,x_n\}$ and $\omega_{ij} \in V^*$. For example, a function over $\{a, b, c\}^*$ from ordered pairs into ordered pairs might be

$$f(x, y) = \langle abcxy, axbycx \rangle.$$

Other authors have defined restricted forms of this general function, such as one in which each variable $x_1,\ldots,x_n$ is used exactly once. We desire to investigate some of these forms and show the relationship between them. The effect of this is to allow us to rewrite a system in an equivalent form which may be easier to manipulate in a particular instance.

Predicates of Degree k to Indicated Context

Definition 1. A canon with indicated context is a canon in which at least one function $f_i(x_1,\ldots,x_n)$ in the premises contains a concatenation of a constant string and a variable. That is,

$$f_i(x_1,\ldots,x_n) = \langle \omega_{01} z_{11} \omega_{11} \ldots z_{n_1 1} \omega_{n_1 1}, \ \ldots \ , \omega_{0k} z_{1k} \omega_{1k} \ldots z_{n_k k} \omega_{n_k k} \rangle$$

where $z_{ij} \in \{x_1,\ldots,x_n\}$, $\omega_{ij} \in V^*$, and at least one ω_{ij} is not the empty string.

This designation arises because the canon may be applied only if the context is matched. For example, the canon $x a \underline{A} \wedge y \underline{B} \vdash \langle x, y \rangle \underline{C}$ requires that a string ending in "a" be in $\tilde{A}$.

<u>Theorem 1</u>. Any predicate of degree $k > 1$ can be reduced to a predicate of degree 1 and indicated context.

<u>Proof</u>. A remark involving a predicate of degree $k > 1$ is of the form $\langle s_1, s_2, \ldots, s_k \rangle \underline{P}$, where the range of each $s_i: (V^*)^n \rightarrow V^*$ is a single string. Choose a symbol X not in V, and create a predicate Q of degree 1 which contains the information in P in the form $s_1 X s_2 \ldots X s_k \underline{Q}$. Wherever the quantity $\langle s_1, s_2, \ldots, s_k \rangle \underline{P}$ appears, replace it by $s_1 X s_2 \ldots X s_k \underline{Q}$, which is of degree 1 and uses indicated context. <u>QED</u>

<u>Corollary 1</u>. Any canonic system can be reduced to one in which no predicate is of degree greater than one, if indicated context is allowed.

<u>Proof</u>. Apply the theorem to all predicates of degree greater than one.

The converse of Theorem 1 is true, but the resulting predicate of degree k may still require indicated context. (A trivial proof just changes every predicate of degree one into degree two and makes the information in each coordinate indentical.) The question to ask is whether the indicated context can be replaced by a simpler operation. This does not appear to be possible, as an example will show. Consider the canon $x a y \underline{A} \vdash x a y \underline{B}$, which has the effect of placing in B all those strings in A which contain an "a" anywhere. If we attempt to remove the indicated context and write something like $\langle x, a, y \rangle \underline{A}$, we have lost the ability to search for the "a" within the string xay. As a matter of fact, it becomes necessary to allow terms of the form xy, where the division of a string into its x- and its y-component becomes completely arbitrary. For example, to simulate a premise remark of $x a y \underline{A} \wedge \ldots$, we must write

$$x\underline{A} \vdash \langle x, \lambda, \lambda \rangle \underline{B}$$
$$\langle \omega x, y, z \rangle \underline{B} \vdash \langle \omega, xy, z \rangle \underline{B}$$

$$<\omega, xy, z>\underline{B} \vdash <\omega, x, yz>\underline{B}$$
$$<x, a, y>\underline{B} \wedge \ldots$$

where B is a temporary predicate introduced only to allow the searching from left to right. Since we feel such a canonic system is no simpler than one allowing indicated context, we shall not attempt to state a converse to the theorem.

Relation to Post Systems

At this point, we wish to demonstrate the equivalence between canonic systems and Post systems. Although canonic systems evolved from Post systems, an exposition of the process has never been made.

A Post system is a set of rewriting rules of the form

$$\alpha_1 x_1 \alpha_2 x_2 \ldots \alpha_n x_n \alpha_{n+1} \rightarrow \beta_1 y_1 \beta_2 y_2 \ldots \beta_m y_m \beta_{m+1}$$

where α_i and β_i are constant strings over a finite alphabet V, and $y_i \varepsilon \{x_1, \ldots, x_n\}$. For example, the rule

$$x11y \rightarrow x22yy$$

would change the string 333114 to 3332244, if x = 333 and y = 4. The left-hand part of a rule is called the antecedent, and the right the consequent.

The distinguishing feature between Post and canonic systems is that a Post system allows only one workspace. All its rules refer to the same string, while a canonic system allows many such workspaces - one corresponding to each predicate. For example, the Post system generating all strings of $\underline{a}$'s of even length is:

axiom : aa
rule : x → aax

The analagous canonic system becomes

$\vdash aa\underline{A}$

$x\underline{A} \vdash aax\underline{A}$

No more computing power can be added to a Post system. It is simply often more convenient to specify a process using the notation of canonic systems.

Theorem 2. Any Post system can be simulated by a canonic system using predicates of degree one.

Proof. If the Post system has the axiom ω, let the canonic system have the rule $\vdash \omega\underline{W}$. If the Post system has the production

$$\alpha_1 x_1 \alpha_2 x_2 \ldots \alpha_n x_n \alpha_{n+1} \rightarrow \beta_1 y_1 \beta_2 y_2 \ldots \beta_m y_m \beta_{m+1} ,$$

let the canonic system have the rule

$$\alpha_1 x_1 \alpha_2 x_2 \ldots \alpha_n x_n \alpha_{n+1} \underline{W} \vdash \beta_1 y_1 \beta_2 y_2 \ldots \beta_m y_m \beta_{m+1} \underline{W}.$$

Then the set $\widetilde{W}$ describes exactly the strings the Post system generates. QED

Observe that, in general, canons with indicated context will be required. The equivalence in the opposite direction is of interest only to show the construction required.

Theorem 3. Any canonic system can be simulated by a Post system.

Proof. We will only indicate the procedure involved in the simulation, since the details are tedious. We let the Post system build up a proof in much the same way that higher level systems do (Chapter 1). Markers must be left in the workspace to separate each remark in the proof, and non-determinism must be allowed so that any canon can "reach" arbitrarily far back for remarks proven previously.

Reduction of Indicated Context to Cross Referencing

Definition 2. A canon

$$f_1(x_1,\ldots,x_n)\underline{F}_1 \wedge \ldots \wedge f_m(x_1,\ldots,x_n)\underline{F}_m \vdash g(x_1,\ldots,x_n)\underline{G}$$

where

$$f_i(x_1,\ldots,x_n) = <\omega_{01}z_{11}\omega_{11}\ldots z_{n_1}\omega_{n_1}, \ldots, \omega_{0k}z_{1k}\omega_{1k}\ldots z_{n_k k}\omega_{n_k k}>$$

and $z_{ij} \in \{x_1,\ldots,x_n\}$, $\omega_{ij} \in V^*$ is said to contain cross referencing if not all the z_{ij} are distinct variables; that is, $z_p = z_q$ does not imply that $p = q$ for all p and q.

Theorem 3. Any canon using indicated context may be reduced to a rule without indicated context if cross referencing is allowed.

Proof. Consider a premise term of the general form

$$\omega_0 z_1 \omega_1 z_2 \ldots \omega_{r-1} z_r \omega_r \underline{G}$$

where the ω_i are constant strings indicating contest, and the z_i are variables. Form the r+1 new canons

$$\vdash \omega_0 \underline{A}_0$$

$$\vdash \omega_1 \underline{A}_1$$

$$\vdots$$

$$\vdash \omega_r \underline{A}_r$$

where the A_i are new, distinct predicates. The intent of the original premise term can be simulated by

$$x_0\underline{A}_0 \wedge x_1\underline{A}_1 \wedge \ldots \wedge x_r\underline{A}_r \wedge x_0 z_1 x_1 z_2 \ldots x_{r-1} z_r x_r \underline{G} .$$

Since each $\widetilde{A}_i$ contains exactly one element by construction, the effect of evaluating these premises is identical to that of applying the original. QED

Summary

We have shown that successive reductions from predicates of degree k to indicated context and from indicated context to cross referencing are possible. These manipulations indicate some of the relationships between canons of various types.

References

1. Alsop, J. W. A canonic translator. MAC-TR-46 (THESIS), Project MAC, Massachusetts Institute of Technology, November 1967.

2. Chomsky, N. On certain properties of grammars. Information and Control 2, September 1960, 137-167.

3. Donovan, J. J. Investigations in simulation and simulation languages. Doctoral dissertation, Yale University, January 1967.

4. Doyle, J. T. Issues of undecidability in canonic systems. S.M. thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, January 1968.

5. Mandl, R. Canonic systems and recursive sets. Project MAC, Massachusetts Institute of Technology, November 1968 (unpublished).

The following paper, not referenced in the text, also concerns canonic systems.

6. Donovan, J. J. and H. F. Ledgard. Canonic systems and their application to programming languages. MAC-M-347, Project MAC, Massachusetts Institute of Technology, April 1967.

CS-TR Scanning Project
Document Control Form

Date : 3 / 15 / 96

Report # LCS-TR-77

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
[✓ Laboratory for Computer Science (LCS)]

Document Type:

[✓ Technical Report (TR)] ☐ Technical Memo (TM)
☐ Other:

Document Information Number of pages: 46(51-images)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
[✓ Single-sided or]
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
[✓ Double-sided]

Print type:
[✓ Typewriter] ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:

Check each if included with document:

[✓ DOD Form] ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other:

Page Data:

Blank Pages(by page number):

Photographs/Tonal Material (by page number):

Other (note description/page number):

Description : Page Number:

(A) IMAGE MAP: (1-46) UNH'ED TITLE PAGE, 2-46
(47-51) SCANCONTROL, DOD, TRGT'S(3).
(B) SOME STAINS ON CERTAIN PAGES.

Scanning Agent Signoff:
Date Received: 3 / 15 / 96 Date Scanned: 3 / 21 / 96 Date Returned: 3 / 21 / 96

Scanning Agent Signature: [illegible]

Rev 9/94 DS/LCS Document Control Form cstrform.vsd

UNCLASSIFIED
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) Massachusetts Institute of Technology Project MAC	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP None

3. REPORT TITLE

Complexity Measures for Language Recognition by Canonic Systems

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

M.S. Thesis, Department of Electrical Engineering, January 1969

5. AUTHOR(S) (Last name, first name, initial)

Haggerty, Joseph P.

6. REPORT DATE October 1970	7a. TOTAL NO. OF PAGES 48	7b. NO. OF REFS 6
8a. CONTRACT OR GRANT NO. Nonr-4102(01)	9a. ORIGINATOR'S REPORT NUMBER(S) TR-77 (THESIS)	
b. PROJECT NO.		
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.		

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency 3D-200 Pentagon Washington, D.C. 20301

13. ABSTRACT

A canonic system C is a specification of a recursively enumerable set, such as a set of strings over a finite alphabet. From this description C, it is possible to generate a system C, called a proof measure function, which is an indication of the complexity of the language defined. For certain simple but important classes of canonic systems, algebraic bounds on these functions can be derived from the structure of the system. Another transformation on C produces a system C^{-1} which characterizes the recognition of strings generated by C. A relationship exists between the measure functions of C and C^{-1}, thus relating the complexity of the recognition procedure to that of the language description.

14. KEY WORDS

Formal Systems
Logical Systems
Complexity Measures
Computation
Canonic Systems

DD FORM 1 NOV 65 1473 (M.I.T.)

UNCLASSIFIED
Security Classification

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 3/24/1996

M.I.T. Libraries
Document Services

darptrgt.wpw Rev. 9/94