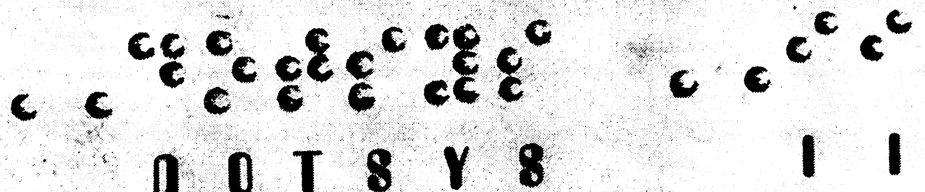
cl-45
rep

[illegible]

MTR-1829



finite-state syntax-directed braille translation

by j. k. millen

july 1970

the MITRE corporation

THE MITRE CORPORATION
BEDFORD, MASSACHUSETTS

MITRE Technical Report

MTR-	1829					
	No.	Vol.	Series	Rev.	Supp.	Corr.

Subject:	Finite-State Syntax-Directed Braille Translation
Author:	Dr. J. K. Millen
Dept.:	D-73
Date:	2 July 1970
Contract No.:	SR 21697
Contract Sponsor:	Sensory Aids Evaluation and Development Center, Massachusetts Institute of Technology
Project:	1248
Issued at:	Bedford, Massachusetts
Department Approval:	[illegible] E. L. Lafferty
MITRE Project Approval:	[illegible] R. A. J. Gildea

Page 1 of 48 Pages

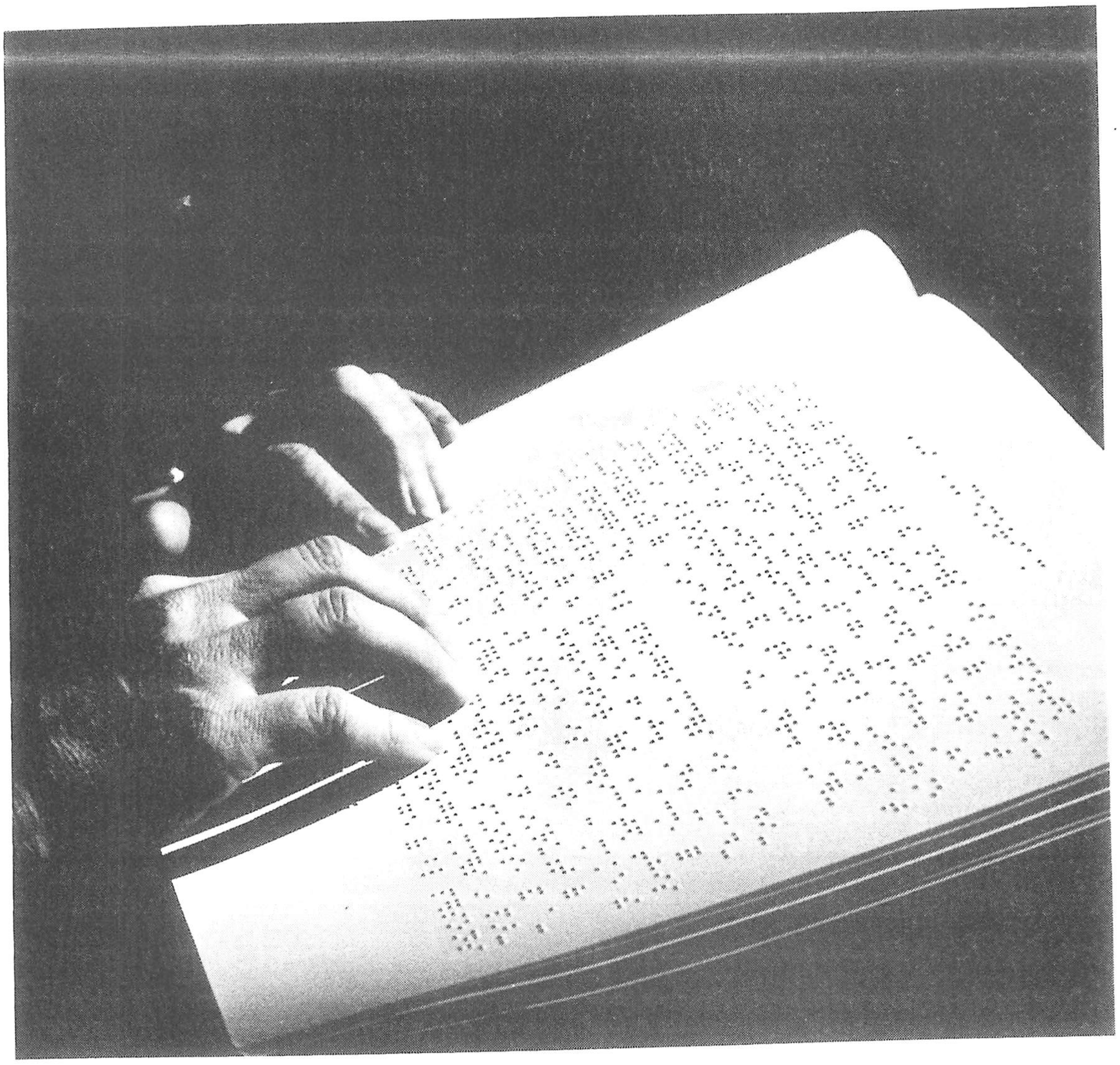


TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

DOTSYS II

FINITE-STATE SYNTAX-DIRECTED BRAILLE TRANSLATION

I. INTRODUCTION

This document describes the structure of DOTSYS II, a COBOL program to translate natural language text into braille. No knowledge of DOTSYS II or of COBOL is presupposed, but some acquaintance with computer programming and the problems of braille translation is expected.

In its present form, DOTSYS II is a COBOL source program which may be run on any computer having a COBOL compiler and a modest amount of core storage. For input, it requires the initial values of some tables, and some text to be translated. Its output is essentially the sequence of braille signs equivalent to the input text, encoded in a form specified as part of the tabular input. An output routine and input tables have been provided which equip DOTSYS II to translate English text into Grade 2 braille, and provide output which can be embossed on a modified high-speed line printer on which the paper is backed with a resilient material.

With modified tables,* DOTSYS II would be capable of processing different kinds of text, such as poetry, languages other than English, and text containing mathematical or other technical notation, or alternate symbols for paragraph starts and other format controls.

At this writing, DOTSYS II has been run on an IBM 360/50 and an IBM 360/65, translating text at the rate of about 1300 words per minute on the former and 3300 words per minute on the latter. The storage space required is about 9000 8-bit bytes for the program itself plus 8000 to 12,000 bytes for tables.

DOTSYS II owes the fundamentals of its translation algorithm both to previous work in the field of braille translation by computer and to elementary concepts in the theory of automata, logic design, and formal languages.

The use of a dictionary of braille contractions, and a buffer or filter to allow the inspection of several characters at a time, were

*Instructions for modifying the tabular input for DOTSYS II are given in MTR-1853, "DOTSYS II: User's Guide and Transfer and Maintenance Manual."

proposed by J. P. Cleave (1,2),* who also described a technique for implementing certain rules through entries in the dictionary besides the basic set of contractions.

The Schack Translation Subroutine (3) placed dictionary entries into classes, and indicated, for each entry, its illegal precedents--that is, those classes of entry which, if just used, prevent the use of the current entry immediately afterward.

In the program described in his dissertation, A. Nemeth (4) assigned individual characters, rather than dictionary entries, into numbered classes. To provide a means for letting past characters affect the translation of present ones, he kept a cumulative information code, which retained information not just about the last characters translated but about as many previous characters as were pertinent. Prefix-stem decomposition was also performed, by keeping a list of prefixes and stem-beginnings. A major division of a word was made between a prefix and a stem-beginning or between a prefix and another prefix. Contractions were not made across such divisions. A useful by-product of this scheme for identifying major word divisions was the ability to hyphenate.

The Schack Translation Subroutine was used in DOTSYS, the translator initiated by three Harvard undergraduates under the direction of the late John K. Dupress. DOTSYS introduced the use in braille translation of the concept of co-routines (5,6), a means of linking the translation routine and input, output, and code conversion routines in such a way as to minimize the buffer storage required for their intercommunication. Co-routine linkage is not a part of the translation process itself and therefore will not be discussed here.

The translation techniques mentioned above have equivalents in DOTSYS II. Despite the apparent diversity of those techniques, however, recent work in syntax-directed transduction (7), a topic in the theory of formal grammars, has shown that those historically-developed techniques are actually pieces of a unified algorithm, and can work together within a common framework in an essentially simple way. Furthermore, the close connection between syntax-directed transduction and automata theory permits, in this case, an application of sequential (finite-state) machine theory to the program design.

In Section II, the applicable parts of the study of syntax-directed transduction and finite-state machines are highlighted,

---

*The numbers in parentheses refer to the like-numbered citations in the list of References.

beginning with the presentation of a few of the basic concepts of the subject, and arriving at a fairly explicit description of the program. Some presently unanswered questions will be raised after this presentation, concerning this and other approaches to braille translation. The translation algorithm is presented in Section III, including a description of the contents and uses of the six tables of DOTSYS II.

II. FINITE-STATE SYNTAX-DIRECTED BRAILLE TRANSLATION

A. Basic Concepts

1. Right-Linear Context-Free Grammars

The first basic definition* is that of a right-linear context-free grammar (8,9). A right-linear context-free grammar is a set of rules, each of the form $A \rightarrow sB$ or of the form $A \rightarrow s$, called productions, where s represents a string (or sequence) of symbols called terminals, and A and B represent symbols belonging to a set of symbols called non-terminals (one of which, usually S, is designated as a start symbol). No symbol is both a terminal and a non-terminal. The modifier "right-linear" means that there is at most one non-terminal on the right side of the arrow, and it is to the right of the string of terminals.

Using the notational convention that non-terminals are capital letters and terminals are words or punctuation, then Figure 1 shows three productions.

S → a rose is A

A → a rose is A

A → a rose,

Figure 1. Three productions

With S designated as the start symbol, the set of the three productions in Figure 1 is a right-linear context-free grammar. The grammar in Figure 1 can be used to generate a set of strings by writing down the start symbol, S, and then interpreting each production as indicating a possible substitution of the symbols on the right side of the arrow for the non-terminal on the left side, wherever and whenever it occurs. Since no terminal symbol can appear on the left side of a production, no further substitution is permitted in a string of terminals; hence terminals terminate substitution, thus the name. In particular, the first production, S → a rose is A, indicates that the start symbol may be replaced by the string "a rose is A". Then, according to the other two productions, the A may be replaced by either "a rose is A" or "a rose,". In the former case, a further substitution may be made for A, and so on. For example, we could generate the string "a rose is a rose is a rose," with three substitutions, as follows:

*Definitions will be given here only informally; formal definitions can be found in the references.

S

a rose is A

a rose is a rose is A

a rose is a rose is a rose,

A sequence of substitutions beginning with the start symbol and ending with a string of terminals is called a derivation of the string of terminals.

The same grammar could be used to generate other strings of terminals--in fact, an infinite number of them. Two examples are shown below.

a rose is a rose,

a rose is a rose is a rose is a rose,

The set of all strings of terminals that can be generated with a grammar is called the <u>language</u> (or <u>formal language</u>) generated by that grammar. A language that can be generated with a context-free grammar is called a context-free language. A language that can be generated with a right-linear context-free grammar is called a <u>finite-state</u> language.

2. Transduction Grammars

We are concerned here not with generating languages, but with translating them. Therefore, we shall now show how to elaborate a context-free grammar so as to specify a translation of the language it generates. This is done by adding to each production an extra right-hand side (enclosed in brackets), containing the same non-terminals as the old right side, but with new (translated) terminals. Here is an example in which the new terminals are mathematical symbols.

S → a rose is A {99 A}

A → a rose is A {-1 A}

A → a rose, {< 100}

Figure 2. A transduction grammar

The above is called a transduction grammar, and the extra right sides transduction elements. Note that there is no direct correspondence between the original terminals and the terminals in the transduction elements. The kind of translation we shall do is more subtle than simple word-for-word translation.

To translate the string "a rose is a rose is a rose," we perform the substitutions which generated it, but using the transduction elements instead of the old right sides. Figure 3 shows the parallel between the old and the new derivations.

S	S
a rose is A	99 A
a rose is a rose is A	99 - 1 A
a rose is a rose is a rose,	99 - 1 < 100

Figure 3. Translation using parallel derivations

In practice, when a translation of a string is to be performed using a transduction grammar, a derivation of the string is not given. Consequently, part of the translation job is to find one, or at least enough information about one to perform the translation. There is a well-known scheme for finding a derivation of a string belonging to a finite-state language, by making use of a finite-state machine.

3. Finite-State Machines

A finite-state machine [10] is a device that changes state only in response to an input, with both the number of states and the number of inputs being finite. One example is an electrical light switch operated by a push-button. Such a switch has two states--"on" and "off", and one input--a push on the button.

A change of state, or a repetition of the current state, occurring in response to an input, is called a transition. Given a right-linear context-free grammar, one can construct (by simulating with a computer program, say) a finite-state machine whose inputs are the strings of terminals occurring in productions and whose possible state changes in response to inputs are determined by the productions. The states of this machine correspond to the non-terminals of the grammar.

If one is also given a right-linear transduction element with each production, that is, a transduction grammar, the machine can be made to produce the terminal strings occurring in the transduction elements as output during a transition. A translation based in this way on a transduction grammar is called <u>syntax-directed</u>.

Initially, the machine is in the state corresponding to the start symbol S, and is presented with a string belonging to the language generated by the grammar from which the machine was constructed. Thereafter the machine operates as follows.

The machine looks for a production of the form $A \to sB$ or $A \to s$ such that the current state of the machine corresponds to A and the input string (or remainder thereof) <u>begins</u> with s.

If a production of the for $A \to sB$ is found, with transduction element s'B , its response is to produce s' as output, change its state to B, remove s from the beginning of the input string, and continue as from the beginning.

If a production of the form $A \to s$ is found, with transduction element s' , its response is to produce s' as output, and stop.

The operation of a finite-state machine can be represented graphically by joining circles (representing states) with arrows (labelled with inputs, showing transitions). In this context, outputs are associated with transitions and are shown on each arrow below or after the input, separated from it by a slash (/). Stopping is indicated by a transition to a new terminal state, drawn with a double circle. The resulting figure is called a <u>transition graph</u>*; a transition graph for the finite-state machine constructed from the transduction grammar in Figure 2 is shown below in Figure 4

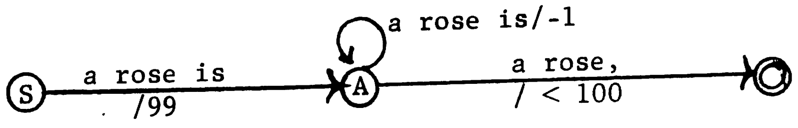


Figure 4. Transition Graph

* Transition graphs are more general than conventional state graphs, because not all inputs necessarily apply at each state, and they might be nondeterministic.

Let us trace the operation of the finite-state machine depicted in Figure 4 as it translates the string "a rose is a rose is a rose is a rose,". Figure 5 lists the state, input string, production used, and output at each step.

STATE	INPUT STRING	PRODUCTION	OUTPUT
S	a rose is a rose is a rose is a rose,	S → a rose is A	99
A	a rose is a rose is a rose,	A → a rose is A	-1
A	a rose is a rose,	A → a rose is A	-1
A	a rose,	A → a rose,	<100

Figure 5. Steps in the translation of an input string

Stringing together all of the outputs gives us "99 - 1 - 1 < 100", the translation of the input string. Note that this could have been done by tracing out the input string on the transition graph alone, without reference to the productions.

B. Theoretical Characterization of DOTSYS II

1. The Transduction Grammar

Abstractly, DOTSYS II simulates a finite-state machine based on a transduction grammar whose old terminals are characters occurring in natural language text and whose new terminals are braille signs. A typical production, with its transduction element, is shown in Figure 6.

A → hered A {⠓⠻⠫ A}

Figure 6. A production with its transduction element, for English-to-braille translation

The transduction grammar used will not be given explicitly because it is too large and repetitive to be readable in that form; it is given implicitly, however, in the tables that drive the program.

2. Other Kinds of Translation

It is reasonable to ask if the program could be used with other transduction grammars to perform other kinds of translation. The answer is a qualified yes. There are two qualifications: First, the transduction grammar must be based on a right-linear context-free grammar, and have right-linear transduction elements; that is, the translation must be finite-state. Second, the tables might have to be modified in format to accommodate different kinds of terminals. On the whole, for purposes other than braille translation, the general scheme used here is likely to be of more use than this particular implementation of it.

3. The Tables in DOTSYS II

Several tables contain the information needed to simulate the finite-state machine. The contraction table, together with the alphabet table, is a list of the input/output pairs occurring in transitions. The decision table decides whether a given input/output pair is associated with a transition from a given current state. The transition table determines the new state from a given input and current state. Details of how these tables work are given in the following chapters. The actual format and use of the tables are more complicated than this abstract discussion suggests, partly due to the use of techniques designed to improve the speed of the translation, and partly due to the provision of variable-shift and right-context fields in the contraction table entries, giving tremendous power to specify translation rules in a way possible, but not presently practical, with an ordinary transduction grammar. A detailed description of these fields may be found in the Contraction Table section.

While a theoretical perspective on the above provision is beyond the scope of this report, there is an important practical consequence of it, having to do with an improvement in the program that would be possible, given greater resources.

4. Possible Improvement

There is presently an inefficiency in the program, resulting from the fact that the beginning of the input string must be matched repeatedly against several similar strings to find the correct transition. For example, the string "here and now" would be matched unsuccessfully with "here," "herence," "hered," and others before "here " was found. Thus, the four letters h, e, r, e were compared at least three extra times. In many text-processing systems*, as well as theoretical discussions on finite-state machines, a technique of introducing new states to prevent superfluous comparisons is used. Each character becomes an input, so that the finite-state machine goes through four states to recognize the string "here"--but never has to backtrack. Figure 7 shows how such a machine discriminates among the strings beginning with "here".

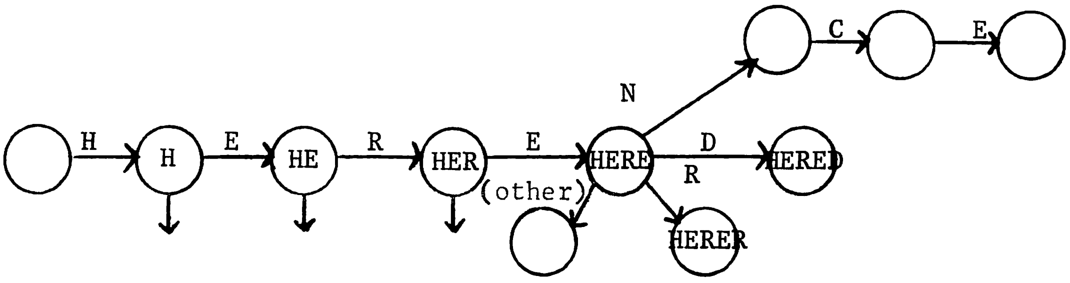


Figure 7. Recognition of "herence", "herer", "hered", and "here" with single-letter inputs (part of a transition graph)

The catch is that one must also assign outputs, reserving them until each particular string is recognized. This is not so bad in the example above, but becomes difficult when right context is introduced (either explicitly, or implicitly with a small shift**), since there is no backtracking on that either, and it may generate more output.

These remarks are meant only to indicate that there _is_ a problem, and that it is compounded by introducing variable shift and right context. Even without the extensions, the kind of operation described above needs tables more complicated than the present ones, because they must be decomposed on a letter-by-letter basis. Because of the requirement that the tabular information is to be changed easily, it is mandatory that conversion of the tables from an initial simple form to the required new form be done automatically by the program. This kind of problem has been solved before, (See 11, 12, 13), but not in a way that applies to the problem at hand.

An investigation of the extensions, and specification of a single-letter-input syntax-directed translator in their presence, appears to be a worthwhile research problem, both from a theoretical and practical point of view.

* See (11), p. 79.

** Cf. Section III.C.2.d.

5. Suitability of a Finite-State Machine for Braille Translation

Translation of English into braille is an open-ended problem for which finite-state methods are not the whole answer. An example of a Braille rule which exceeds finite-state translation capability is the following:

> When in inkprint a number or letter is preceeded or followed by a symbol or abbreviation for coinage, weight, measure, or other special sign, including the individual terms of a sequence, the corresponding braille symbol or abbreviation, without the period or plural "s", should always be placed immediately before the number or letter to which it refers.*

For example, an input of "3 yds." should produce as output the braille signs for "YD3". Turning "yds." into "YD" is trivial, but reversing the order is not possible with a finite-state machine. In order to handle this rule, DOTSYS II must put out 3YD instead, followed by another symbol which will instruct the output routine, as a special case, to move the letter signs before the immediately preceding number.

Another rule beyond the capability of a finite-state machine is this:

> There should be no space between the lower-sign contractions to, into, by, and the word which follows, if there is no natural pause between them.

A natural pause occurs between "to," "into," or "by" and the next word if they are parts of independent grammatical constructions. Although there are programs which perform syntactic analysis of English deeply enough to handle this problem, they are relatively slow, on the order of one word per second. Calling in such a program for every sentence in which "to," "into," or "by" appears would significantly reduce the speed of the translation program. Instead, DOTSYS II invokes the next sentence of the same braille rule:

> "If in doubt about the pause, they should be joined."

With the current tables, DOTSYS II always joins "to", "into", or "by" with the next word.

* (14), p. 30

III. THE TRANSLATION ALGORITHM

A. Introduction

1. Basic Approach

The following paragraphs stress the conceptual foundations of the translation algorithm. The level of detail is gradually deepened until the names and roles of the most important tables and dynamic storage areas have surfaced. A mnemonic summary of the chronological steps in the algorithm is given in Table 1. It may be helpful to refer to Table 1 again at the end of this introduction.

1.	Index	alphabet table
2.	Search	contraction table, decision table, right-context table
3.	Output	contraction table or alphabet table, sign table
4.	Shift	contraction table
5.	Change State	transition table

Table 1. Chronological form of the translation algorithm as a loop of five steps, showing the tables used

DOTSYS II translates the input text from left to right. At a given moment, ten consecutive characters are inspected. Those ten characters occupy the buffer, a set of ten storage locations (See Figure 8.).

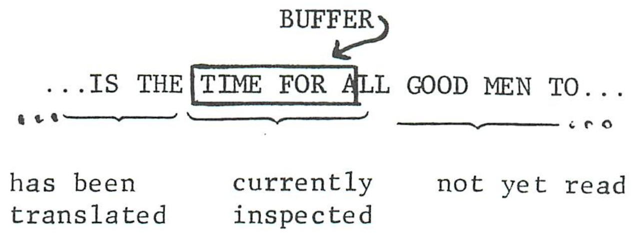


Figure 8. The ten-character buffer

If ten or fewer characters beginning at the left end of the buffer are to be translated as a group in braille, i.e., contracted, then DOTSYS II puts out the braille sign or signs and shifts the contents of the buffer left to move the contracted characters out of the buffer. The same number of new characters are read into the vacancies at the right end of the buffer.

In the example shown in Figure 8, the word TIME occurs at the left end of the buffer. Since TIME can be contracted, DOTSYS II puts out the two braille signs ⠐⠞ which are its braille translation and then shifts the buffer contents left by four characters. As shown in Figure 9, the first character after TIME, a space, is then the leftmost character in the buffer.

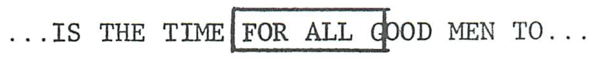


Figure 9. Contents of the buffer after translating TIME

If no character string which can be contracted begins at the left end of the buffer, DOTSYS II puts out the braille equivalent of the leftmost character in the buffer and then shifts the buffer contents left by one character. In the example shown in Figure 9, DOTSYS II puts out a space and then shifts the buffer contents left by one character, resulting in the situation shown in Figure 10.

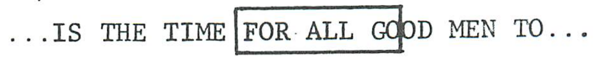


Figure 10. Contents of the buffer after translating the blank after TIME

2. Conditions for Contraction

Most of the program is dedicated to determining whether or not there is a character string beginning at the left end of the buffer which can be contracted. DOTSYS II uses a table containing all the character strings for which there are standard braille contractions: the contraction table. Each entry in the contraction table consists of a character string, its braille sign or signs, and other information to be discussed below.

The first step in testing for the possibility of a contraction is to search the contraction table for a match with the buffer. In the example shown in Figure 8, there was an entry in the contraction table whose character string was TIME. Since this matched the first four characters of the buffer, TIME was a possible contraction.

Finding an entry in the contraction table which matches the initial characters in the buffer is a necessary condition to contract those characters, but not a sufficient condition. For example, the letter group "ING" is contracted when it occurs in the middle or at the end of a word, such as in LINGER and BEING, but is not contracted at the beginning of a word, as in INGRATE. (A list of the letter groups for which there are standard English braille contractions, and the rules which determine under what circumstances each letter group is to be contracted, are found in English Braille (14).)

3. The Three Mechanisms for Implementing Braille Rules

There are basically three mechanisms in DOTSYS II for implementing the braille rules:

(a) Contraction Table Additions

Some of the rules are implemented merely by additions to the contraction table. For example, when the contracted letter groups EA and AR overlap, as in the word NEAR, the AR contraction is preferred. This rule is implemented by putting the letter group EAR in the contraction table and specifying its translation to be the two braille signs for E and AR.

(b) Right Context

Some contractions are made only when the character string to be contracted is followed immediately by another letter--or, alternatively, only when followed by a punctuation mark or other non-alphabetic character. Either of these conditions may be indicated by means of a third part of the contraction table entry: the right-context character. This character is either blank, L, or P, meaning, respectively: no condition, follow by a letter, or follow by a character other than a letter. For example, the contraction table entry with the character string EA will have a right-context of L, since EA is never contracted when it occurs at the end of a word, as in SEA, or when followed by a hyphen or apostrophe; hence, it must be followed by a letter.

The choice and meanings of right-context characters are given in the right-context table. By changing the entries in this table, new right-context conditions can be supplied or old ones altered.

(c) Finite-State Memory

DOTSYS II uses a finite-state memory to implement rules involving characters to the left of, i.e., preceding, the characters currently under inspection. Unlike the right-context mechanism, which tests only an immediately adjacent character, the finite-state memory allows a contraction to be affected by characters which may have occured remotely (in the past, hence the term "memory"). The finite-state memory appears in DOTSYS II in the form of a state vector, and it is used with the help of a transition table and a decision table.

The state vector is a number of consecutive storage locations called state variables. Each state variable contains the character Y (for Yes) or N (for No). The occurrence of some characters in the input text can change specific state variables from N to Y, or vice versa, as they are shifted out of the buffer. The change, if any, is specified in the transition table. For example, one of the state variables is set to Y by any letter, and reset to N by a punctuation mark other than the apostrophe and hyphen. Thus, if ING is at the left end of the buffer, the contents of this state variable can then be used to decide whether that ING occured after the beginning of a word: yes if Y, no if N. The decision table contains the information that ING (and other final letter contractions) are to be contracted if the relevant state variable is Y and not otherwise.

4. Summary

To summarize, DOTSYS II translates input characters from left to right while passing them leftward through a ten-character buffer. A character string beginning at the left end of the buffer is contracted if three conditions are satisfied:

(1) it matches an entry in the contraction table;

(2) it has an acceptable right-context character, if required;

(3) contraction if permitted by the decision table, as a function of the state variables.

In this case the contraction table entry is said to apply to the character group in the buffer.

DOTSYS II then puts out one or more braille signs, as appropriate, and shifts the characters just translated out of the buffer. As the characters leave the buffer, they may change some state variables in accordance with the transition table.

In addition to the contraction table, transition table, and decision table, there are two more tables: the alphabet table and the sign table. Conceptually, the alphabet table is part of the contraction table; it contains the braille signs used for individual characters when they are not translated as part of a contraction. It also contains indexing entries for efficient search of the contraction table. The sign table merely sets up a correspondence between the numbers from 0 to 63 and the 64 braille signs (counting the space). This allows the numerical equivalent of the signs to be used in the program for indexing and for storage in tables.

In the following sections, where tables are described, their formats are presented as COBOL data items, which describe storage allocation in a simple, logical way. The example below, showing a typical COBOL (15,16) data item and its corresponding storage layout with sample contents, is sufficient explanation for our purposes.

```
01   A.
     02   B           OCCURS 2 TIMES.
          03   C          PICTURE S99, USAGE COMPUTATIONAL.
          03   D          PICTURE XXX.
```

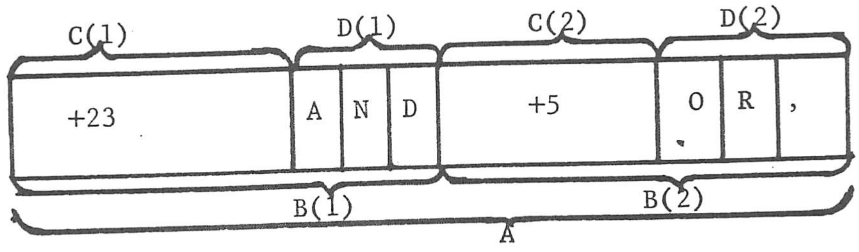


Figure 11. A typical COBOL data item and its corresponding storage layout, with sample contents

B. Finite-State Logic

1. The State Vector

This section describes the finite-state memory in DOTSYS II. As noted in the introduction, the finite-state memory implements braille translation rules whose application is affected by characters preceding the characters to be translated.

DOTSYS II remembers past information with the state vector, which is described by the following COBOL data item:

```
01    STATE-VECTOR
      02    STATE-VARIABLE OCCURS 10 TIMES, PICTURE X.
```

Thus, the state vector consists of at most ten state variables, each one a single character.

Each state variable has value of Y or N; all are given the value N initially. The value of each state variable is recomputed immediately after the characters just translated are shifted left out of the buffer. The new value of a state variable depends on its old value and on the character or group of characters shifted out of the buffer--that is, on the alphabet or contraction table entry just found.

2. Input Classes

Because many different characters or contractions may have the same effect on the state vector, they are grouped into numbered input classes. This way, the new value of each state variable depends on its old value and on the input class of the alphabet or contraction table entry. The input class number is one field of the table entry.

In order to be grouped into the same input class, two contractions must also be made or not upon the same condition of the state vector--i.e., they must be able to share a common decision table entry as well as a common transition table entry.

Contractions in the same input class are similar enough to share a common brief description. The descriptions of the input classes, as currently provided, are given in the appendix, for the sake of concreteness. The appendix also shows the information retained by state variables. See Figure 22 for the associated logic tables.

3. The Transition Table

The transition table specifies the recomputation of the state vector. It is described by this COBOL data item:

```
01   TRANSITION-TABLE.
     02   TRANSITION TABLE COLUMN OCCURS 20 TIMES.
          03   TRANSITION OCCURS 10 TIMES, PICTURE X.
```

The above data item describes a table which looks like Figure 12. NIC and NSV are, respectively, the number of input classes and the number of state variables.

		INPUT CLASS			
		1	2	...	(NIC)
	1	R	S	...	R
STATE VARIABLE	2	-	T		-
	⋮				
	(NSV)	-	R	...	R

Figure 12. The transition table, with typical entries

Each row is associated with a state variable, and each column is associated with an input class. Each entry, which is either R, S, T, or -, specifies the effect of an input class on a state variable, as shown in Figure 13.

Transition Table Entry	R		S		T		-	
Old Value of State Variable	Y	N	Y	N	Y	N	Y	N
New Value of State Variable	N	N	Y	Y	N	Y	Y	N

Figure 13. Effect of input classes on state variables, as specified by transition table entries

rows for $i_{v=Y}$ and $i_{v=N}$, allowing the new value of u to be chosen according to the value of v. An input class can be further subdivided if the values of two or more other state variables are taken into account, and the whole procedure can be performed for as many input classes as necessary.

While the logical capability of the program is not limited, the above procedure entails a great deal of work. In the environment of braille translation, however, the state variables can be chosen so that their transitions are largely or entirely independent of one another.

In the terminology of switching theory, our finite-state logic is essentially a realization with a reset-set-toggle flip-flop memory of a sequential machine.

C. The Contraction Table

1. Introduction

The largest table in DOTSYS II, the contraction table,contains the 189 English braille contractions, plus many additional entries to implement some rules and indicate exceptions to others. Discussed in this section are: the format of contraction table entries, the table search algorithm, and the ordering of table entries.

2. Contraction Table Entry Format

This is the COBOL data item describing the contraction table:

```
01    CONTRACTION-TABLE.
      02    TABLE-ENTRY              OCCURS 300 TIMES.
            03    STRING.
                  04    TABLE-CHAR        OCCURS 9 TIMES, PICTURE X.
            03    RIGHT-CONTEXT           PICTURE X.
            03    INPUT-CLASS             PICTURE S99, USAGE COMPUTATIONAL.
            03    SHIFT                   PICTURE S99, USAGE COMPUTATIONAL.
            03    SIGNS.
                  04    SIGN              OCCURS 4 TIMES, PICTURE S99,
                                            USAGE COMPUTATIONAL.
```

Thus, each contraction table entry has five fields, containing: all but the first character of a character string to be represented, a right-context character, the input class number, the number of characters to shift out of the buffer, and the numerical codes of the braille signs to be put out. In Figure 15 is an example of the contraction table entry for EA.

STRING	RIGHT-CONTEXT	INPUT-CLASS	SHIFT	SIGNS
A$	L	4	2	2 99 99 99

Figure 15. Contraction table entry for EA

The absence of the E, the dollar sign in the string, and the numbers in the SIGNS field will be explained in the paragraphs below.

a. The STRING Field

The first field of the contraction table entry, the STRING field, comprises nine characters, and represents a character string recognized as a whole when encountered in the buffer. The first character in the string is encoded by the position of the entry in the table, and is not included in the STRING field. The STRING field consists of the second through last characters in the character string; if there are fewer than nine, they are followed by a dollar sign, and enough trailing blanks to make a total of nine characters. For example, the STRING field for the character string THROUGH looks like this: HROUGH$bb. The entry is placed in the table in a section where all strings are prefixed by T. The limits of that section are identified in the alphabet table.

b. The RIGHT-CONTEXT Field

The second field of the contraction table entry, the RIGHT-CONTEXT field, contains one character: either a blank or a character such as L or P. Currently, an L indicates that a <u>letter</u> must be found in the buffer immediately following the character group to which the entry would apply, in order for this contraction table entry to be applicable. A P indicates that any character other than a letter must be found there, instead. A blank indicates the absence of a right-context condition.

For example, the contraction table entry shown in Figure 15 is a match for the buffer contents shown in Figure 16a but not in Figure 16b.

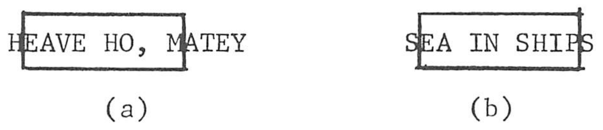


Figure 16. Buffer contents which (a) do, (b) do not match the contraction table entry for EA shown in Figure 15

c. The INPUT-CLASS Field

The third field of the contraction table entry, the INPUT-CLASS field, is a positive integer which assigns this entry to a group of entries having the same effect on the state vector (as specified in the transition table) and the same requirements on the state vector for the contraction to be made (as specified in the decision table).

As indicated in Figure 15, the EA entry is in input class 4. All entries in this class have an effect on the state vector specified in column 4 of the transition table, and none will be used unless the state variables satisfy conditions given in the lower part of some column of the decision table whose upper part has Y opposite this input class.

d. The SHIFT Field

The fourth field of the contraction table entry, the SHIFT field, is a positive integer giving the number of characters to be shifted out of the buffer after this entry is applied. This field might appear redundant, because the length of the character string just translated is implied in the STRING field. But it is sometimes convenient to translate and shift out of the buffer only a part of the character string to which the entry applies.

The best example of a situation in which it is convenient to translate and shift out only a part of the character string is in the implementation of the rule which eliminates spaces between the braille signs for A, AND, FOR, OF, THE, and WITH. Assume that there are two contraction table entries as show in Figure 17, and suppose that the buffer contents are as shown in Figure 18.

	STRING	RIGHT-CONTEXT	INPUT-CLASS	SHIFT	SIGNS
(a)	ND FOR $		7	4	47 99 99 99
(b)	OR THE $		7	4	63 99 99 99

Figure 17. Contraction table entries for AND FOR and FOR THE

$\boxed{\text{AND FOR T}}$HE OTHERS,

Figure 18. Buffer contents for text example

DOTSYS II will match AND FOR, translate it as ⠯ (AND, coded by 47), and shift four characters, resulting in the buffer contents shown in Figure 19.

$\boxed{\text{FOR THE OT}}$HERS,

Figure 19. Buffer contents after shifting

Now FOR THE will be matched and translated as ⠿ (FOR, coded by 63), and the buffer contents shifted by four again. Note that ⠯ and ⠿ have just been put out with no intervening space. Similarly, there will be no space between ⠿ (FOR) and ⠮ (THE).

e. The SIGNS Field

The fifth and last field of the contraction table entry, the SIGNS field, contains four numbers representing the braille translation of that part of the character strings which will be shifted out of the buffer.

The correspondence between the numbers and braille signs is given in the sign table. If the translation is fewer than four braille signs, the extra numbers are specified as 99.

3. The Ordering of the Contraction Table

The position of an entry in the contraction table depends on the character string to which it applies, according to three rules:

a. An entry is placed with other entries applying to character strings with the same first character. For example, the entries for EA, ED, EN, EAR, EVER, etc. are together in one section of the table. Thus, the contraction table is a sequence of sections; each section is associated with the character which begins all of the character strings to which the entries in that section apply.

<u>The order of the sections is the same as the order, in the alphabet table, of the associated characters</u>.

b. Within a section, entries are grouped together if they apply to character strings having the same second character (i.e., having STRING fields with the same first character). For example, the ordering ALSO, AND, ALWAYS would violate this rule, since the ALSO and ALWAYS must not be separated by an entry applying to a character string whose second letter is not L. In this example, the entry for AND could occur either before or after the entries for storage beginning with AL.

c. Among entries applying to strings having the same first two characters, if there are situations in which both contraction table entries would apply, place the preferred entry first in the table. For example, the entry for EAR must precede the entry for EA, since, when they both apply, as in NEAR, the EAR contraction is used.

Once the three rules cited above have been satisfied, any remaining questions of order may be dictated arbitrarily, but they are best settled with an eye to the efficiency of the contraction table search. For best results, entries and groups of entries with higher frequency of use ought to be placed ahead of others less frequently used, whenever possible under the restrictions stated above.

4. <u>The Contraction Table Search</u>

In order to translate the characters in the buffer, DOTSYS II attempts to find a matching contraction table entry in the following manner. First, the subscript (or index) of the first entry in the section associated with the first character in the buffer is obtained from the EXTENT field of its alphabet table entry.

DOTSYS II compares the second through last characters in the buffer with the characters in the STRING field of entries in the proper section of the table, stopping when a contraction is made, or when the end of the section or the end of a succession of entries matching the second character of the buffer is reached. If no contraction in the table is made--that is, if the contraction table search fails--then the first character in the buffer is translated by itself as specified in its entry in the alphabet table.

Comparison of the STRING field of an entry with the characters after the first in the buffer is character by character, stopping (unsuccessfully) on a mismatch or (successfully) when the dollar sign in the string field is found or all nine characters match.

Even after a successful comparison, of course, the right-context character and decision table must still be checked; if they do not permit this entry to be used, the search continues to the next entry.

D. The Alphabet Table

The alphabet table is used in two ways: first, to identify each character which moves into the initial position in the buffer; second, to provide the last-resort translation of that character as a single braille sign, if it is not part of a group which can be contracted at that time.

It is described by this COBOL data item:

```
01   ALPHABET.
     02   ALPHABETIC-ENTRY      OCCURS 64 TIMES.
          03   SYMBOL           PICTURE X.
          03   CHAR-CLASS       PICTURE X99, USAGE COMPUTATIONAL.
          03   SINGLE-SIGN      PICTURE S99, USAGE COMPUTATIONAL.
          03   EXTENT           PICTURE S999, USAGE COMPUTATIONAL.
```

The SYMBOL field contains the characters; the CHAR-CLASS field, its input class; the SINGLE SIGN field, the number representing its braille sign; and the EXTENT field,an index into the contraction table. Aside from the EXTENT field, the alphabet table entry might have been incorporated into the contraction table (with a SHIFT field of 1).

Here is an example to show what the EXTENT field is and how it is used to initiate the contraction table search. Suppose the buffer contains the word TIME, as shown in Figure 20.

THE $\boxed{\text{TIME FOR A}}$LL

Figure 20. TIME in the buffer

DOTSYS II searches the alphabet table, starting at the top, for an entry whose SYMBOL field is equal to the character occupying the first position in the buffer--T. This brings it to the point in the alphabet table illustrated in Figure 21, which shows the entry for T and also the next one, which happens to be for A.

SYMBOL	CHAR-CLASS	SINGLE-SIGN	EXTENT
T	1	30	20
A	1	1	43

Figure 21. Two successive entries in the alphabet table

The EXTENT field contains the subscript of the first of the entries in the contraction table which apply to character strings beginning with that character.

The EXTENT field for T is 20; this means that the twentieth entry in the contraction table is the first one applying to strings beginning with T. Furthermore, since the order of the sections in the contraction table is the same as the order of the associated characters in the alphabet table, the last contraction table entry for T is the one just before the first entry for A, since A appears next in the alphabet table. Thus, the contraction table search starts at the twentieth entry and ends, at worst, at the forty-second.

Every character which can appear in the input text must have an alphabet table entry, even those which begin no contractions. The EXTENT field of an entry for a character which begins no contractions should be the number of entries in the contraction table plus one.

Note that, because DOTSYS II must search the alphabet table for practically every character in the input text, it is more efficient to arrange the alphabet table so that the entries for the more frequently occurring characters are closer to the top of the table.

E. The Right-Context Table

The right-context table lists the possible characters that may be found in the RIGHT-CONTEXT field of a contraction table entry, and the associated input classes of characters which must occur immediately following the character group in the buffer if the entry is to apply.

It is described by the following COBOL data item:

```
01    RIGHT-CONTEXT-TABLE.
      02    RIGHT-CONTEXT-TABLE-ENTRY OCCURS 2 TIMES, DEPENDING ON NNT.
            03    NON-TERMINAL          PICTURE X.
            03    RIGHT-CONTEXT-CLASSES.
                  04    RIGHT-CONTEXT-CLASS OCCURS 4 TIMES, PICTURE S99,
                                            USAGE COMPUTATIONAL.
```

Thus, each right-context table entry pairs a right-context character symbol, or non-terminal, in the NON-TERMINAL field, with four input class numbers; characters in those input classes are to be acceptable as right context.

The name NON-TERMINAL, borrowed from the theory of context-free languages, is used instead of RIGHT-CONTEXT-CHARACTER in order to distinguish the symbol representing a class of characters from the actual character in the buffer which may or may not be acceptable as right context.

Where fewer input class numbers are required than the number of RIGHT-CONTEXT-CLASS fields, input class numbers may be repeated to fill the entry.

F. Output

1. Intermediate Output

The output section of DOTSYS II is not really part of the translation program, but it is necessary in order to put the braille output in the desired printed, punched, and tactile forms. The translation section produces as intermediate output a sequence of two-digit numbers taken from the SIGN-field of contraction table entries or the SINGLE-SIGN field of alphabet table entries. Numbers from 0 to 63 represent braille signs according to a correspondence defined below. Number 64 is not used, but numbers from 65 through 99 (possibly not all used) represent format controls for tabulation, line-skipping, paragraphing, and the like. The output section turns numbers from 0 through 63 into braille signs in whatever form is requested (and implemented) with the help of the sign table, and acts upon numbers from 65 through 99 according to various built-in capabilities such as inserting spaces or skipping to the beginning of the next line or page.

Automatic line breaking is presently also a function of the output section, which begins a new line after the last space occurring normally before the end of the current line. Hyphenation, however, would have to be added to the translation section, because of the existence of braille rules concerning the use of contractions on either side of a hyphenated line break.

2. Kinds of Output Implemented

The kinds of output presently implemented are:

(1) printed proof output, showing each braille sign (drawn with periods), up to three proof characters to aid in identifying the character, and the number of the sign.

If proof output is requested, DOTSYS II prints displays showing the current contents of the right-context table, decision table, and transition table before the braille output begins. Figure 22 below shows these displays and some braille proof output.

(2) braille output; braille signs are printed backwards and read from right to left, so that when the dots push through the paper (if there is a resilient backing) the resulting embossed dots on the other side read normally.

(3) punched output; the two-digit numbers for braille signs are punched on Hollerith cards, two columns per number, giving a maximum of forty signs per card.

```
                    DECISION TABLE

COLUMN                  01  02  03  04  05  06  07  08

INPUT CLASS      01     Y   -   -   -   -   -   -   -
INPUT CLASS      02     -   -   Y   -   -   -   -   -
INPUT CLASS      03     G   -   -   -   -   -   -   -
INPUT CLASS      04     -   Y   -   -   -   -   -   -
INPUT CLASS      05     G   -   -   -   -   -   -   -
INPUT CLASS      06     -   -   -   Y   -   -   -   -
INPUT CLASS      07     -   -   -   Y   -   -   -   -
INPUT CLASS      08     -   -   -   -   Y   -   -   -
INPUT CLASS      09     -   -   -   -   -   Y   -   -
INPUT CLASS      10     -   -   -   -   -   -   Y   -
INPUT CLASS      11     -   -   -   -   -   -   -   Y
INPUT CLASS      12     G   -   -   -   -   -   -   -
INPUT CLASS      13     G   -   -   -   -   -   -   -
INPUT CLASS      14     G   -   -   -   -   -   -   -
INPUT CLASS      15     -   -   N   G   -   -   -   -

STATE-VARIABLE   01     -   -   N   -   -   -   -   -
STATE-VARIABLE   02     -   Y   -   N   -   -   -   -
STATE-VARIABLE   03     N   N   -   N   -   -   -   -
STATE-VARIABLE   04     -   -   -   -   Y   N   N   Y
STATE-VARIABLE   05     -   -   -   -   N   Y   -   -
```

```
                    TRANSITION TABLE

STATE VARIABLE          01  02  03  04  05

INPUT CLASS      01     R   S   -   -   -
INPUT CLASS      02     S   R   -   -   -
INPUT CLASS      03     -   R   -   -   -
INPUT CLASS      04     R   S   -   -   -
INPUT CLASS      05     -   -   T   -   -
INPUT CLASS      06     R   S   -   -   -
INPUT CLASS      07     R   R   -   -   -
INPUT CLASS      08     R   R   -   -   -
INPUT CLASS      09     R   R   -   -   -
INPUT CLASS      10     R   R   -   S   -
INPUT CLASS      11     R   R   -   R   -
INPUT CLASS      12     -   R   -   -   S
INPUT CLASS      13     -   R   -   -   R
INPUT CLASS      14     R   R   -   -   -
INPUT CLASS      15     R   S   -   -   -
```

```
              RIGHT CONTEXT TABLE

NON-TERMINAL      INPUT CLASSES

     P          14  03  02  02

     L          01  13  13  13
```

Figure 22. Logic table displays and proof output

```
D  A  T  E  :        #  F  I  _J F  _J B     ¬V EN E  Z  U  E  L  A     ¬R O  C  K  E  F  E  L  L  ER
25 01 30 17 18 00 00 60 11 10 40 26 11 40 26 03 00 32 39 34 17 53 37 17 07 01 00 32 23 21 09 05 17 11 17 07 07 59 00 00

P  O  ST P  O  N  ED    ?     ¬V EN E  Z  U  E  L  A  N     V  I  S  I  T     ;  C     OF    P  O  S  S  I  #
15 21 12 15 21 29 43 00 38 00 32 39 34 17 53 37 17 07 01 29 00 39 10 14 10 30 00 06 09 00 55 00 15 21 14 14 10 60 00 00

V  I  O  L  =  E  .
39 10 21 07 48 17 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

      ¬THE    ¬N E  W     ¬Y O  R  K     G  O  V  ER N  O  R  ,     O  N     A     ¬L A  T  IN
00 00 32 46 00 32 29 17 58 00 32 61 21 23 05 00 27 21 39 59 29 21 23 02 00 21 29 00 01 00 32 07 01 30 20 00 00 00 00 00

¬A M  ER I  C  A  N     T  OU R     FOR   ¬P R  E  S  I  D  ENT      ¬N I  X  O  N  ,     S  D     H  E
32 01 13 59 10 09 01 29 00 30 51 23 00 63 00 32 15 23 17 14 10 25 34 30 00 32 29 10 45 21 29 02 00 14 25 00 19 17 00 00

A  G  R  E  ED    WTH   ?  R  F  G  R  E  T  "     TO THE   R  E  Q  U  E  ST    "  THE   ¬V EN E  Z  U  E  L  A  N
01 27 23 17 43 00 62 00 38 23 17 27 23 17 30 52 00 22 46 00 23 17 31 37 17 12 00 52 46 00 32 39 34 17 53 37 17 07 01 29

G  O  V  ER N  =  T  .        ¬H E     S  P  O  K  E     IN    ¬T R  IN I  D  A  D  ,     WH    H  E     456H
27 21 39 59 29 48 30 50 00 00 32 19 17 00 14 15 21 05 17 00 20 00 32 30 23 20 10 25 01 25 02 00 49 00 19 17 00 56 19 00

B  E  EN    S  CH ED U  L  ED    TO    L  ,  V  E     T  D     FOR   ¬C AR A  C  A  S  .
03 17 34 00 14 33 43 37 07 43 00 22 00 07 02 39 17 00 30 25 00 63 00 32 09 28 01 09 01 14 50 00 00 00 00 00 00 00 00 00

¬R O  C  K  E  F  E  L  L  ER    T  O  L  D     N  E  W  S  M  EN    H  E     H  O  P  ED    TO V  I  S  I  T
32 23 21 09 05 17 11 17 07 07 59 00 30 21 07 25 00 29 17 58 14 13 34 00 19 17 00 19 21 15 43 00 22 39 10 14 10 30 00 00

¬V EN E  Z  U  E  L  A     ?  5  S     5  T     IN    THE   F  U  T  U  R  E  .  "
32 39 34 17 53 37 17 07 01 00 38 16 14 00 16 30 00 20 00 46 00 11 37 30 37 23 17 50 52 00 00 00 00 00 00 00 00 00 00 00
```

Figure 22. Logic table displays and proof output (continued)

3. The Numerical Codes for the Braille Signs

The correspondence between numbers from 0 through 63 and the sixty-four braille signs, recorded in the sign table, is based on the standard numbering of the six dots in a braille cell, as shown:

```
1 • • 4
2 • • 5
3 • • 6
```

Figure 23. The standard numbering of the dots in a braille cell

Given a braille sign having the dots numbered k_1, k_2, ..., then the corresponding number is $2^{k_1-1} + 2^{k_2-1} + \ldots$ For example, the braille sign for A, which uses only dot 1, is given the number $2^{1-1} = 2^0 = 1$. The braille blank is given the number zero by this system.

Another way of thinking of this numerical encoding of braille signs is through a correspondence between dots and the digit positions of a binary number. If six bits are numbered 1 to 6 from right to left, the number representing a given braille sign is the binary number with ones in the bit positions corresponding to the dots in the sign. For example, the braille sign for B has dots in positions 1 and 2. Thus, it is represented by the binary number 000011, or, in decimal, 3.

4. The Sign Table

The COBOL data item below describes the sign table

```
01   SIGN-TABLE.
     02   SIGN-TABLE-ENTRY              OCCURS 64 TIMES.
          03   DOTS-1-4                 PICTURE XX.
          03   DOTS-2-5                 PICTURE XX.
          03   DOTS-3-6                 PICTURE XX.
          03   PROOF-CHARACTERS         PICTURE XXX.
```

The first three fields of a sign table entry are the three rows of the braille cell represented by that entry. Spaces and periods are placed in those fields to make a picture of the character.

The PROOF-CHARACTERS field contains a combination of up to three characters which uniquely identify the sign. DOTSYS II prints them, in addition to the braille sign, when in proof mode, to help a programmer test the output.

The correspondence between braille signs and numbers is set up simply by the order of the entries; the first (representing the braille sign for A) is given the number 1, and so on. Thus, the number of a sign is a subscript which can be used by the output program to obtain its picture and proof characters. (The blank sign is placed in the table as the 64th entry because there can be no zeroth entry.)

For example, the sign table entries for "A" and THE are shown in Figure 24.

	DOTS-1-4		DOTS-2-5		DOTS-3-6		PROOF-CHARACTERS			
(a)	●						A			
(b)		●	●		●	●	T	H	E	

Figure 24. Sign table entries for (a) A and (b) THE

Jonathan K Millen
Dr. J. K. Millen

JKM:ces

APPENDIX

STATE VARIABLES AND INPUT CLASSES

State Variable	1	after the start of a number
	2	after the start of a word
	3	grade 1 translation
	4	in a quotation
	5	in italicized text
Input Class	1	contractions always used in grade 2
	2	digits
	3	most punctuation
	4	contractions used after the start of a word
	5	$G (grade switch)
	6	contractions used only at the start of a word
	7	isolated full-word contractions
	8	$P" (start paragraph in quotation)
	9	$P (start paragraph in italics)
	10	" (left quote)
	11	" (right quote)
	12	_ _ (begin italics)
	13	_ (last word of italics)
	14	(space)
	15	A to J occurring in a number

REFERENCES

1. J. P. Cleave, 'Braille Transcription,' Mechanical Translation, Vol. 2., No. 3 (1955), pp. 50-54.

2. ____________, 'The Mechanical Transcription of Braille,' Mechanical Resolution of Linguistic Problems, A. Booth, L. Brandwood, J. P. Cleave, Academic Press, N.Y., 1958, pp.97-109.

3. A. S. Schack and R. T. Mertz, Braille Translation System for the IBM 704, M&A-10, (1961), International Business Machines Corp., N.Y.

4. A. Nemeth, Digital Enciphering of English into Braille, Wayne State University, (1963) (Project No. 433).

5. FINAL REPORT to the Vocational Rehabilitation Administration, Department of Health, Education and Welfare, by the Sensory Aids Evaluation and Development Center, M. I. T.

6. D. E. Knuth, The Art of Computer Programming, Addison-Wesley (1968), pp. 190-196.

7. P. M. Lewis and R. E. Stearns, 'Syntax-Directed Transduction,' J. ACM. (15) No. 3, (July 1968).

8. N. Chomsky and M. P. Shutzenberger, 'The Algebraic Theory of Context Free Languages,' Computer Programming and Formal Systems, P. Braffort and D. Hirschberg (Eds.), North Holland Publishing Co., Amsterdam (1963), pp. 118-161.

9. S. Ginsburg, The Mathematical Theory of Context-Free Languages, McGraw-Hill (1966).

10. M. L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall (1967).

11. C. Salton, Automatic Information Organization and Retrieval, McGraw-Hill (1968).

12. P. J. H. King, 'Conversion of Decision Tables to Computer Programs by Rule Mask Techniques,' Comm. ACM (9) No. 11 (Nov. 1966).

13. K. Thompson, 'Regular Expression Search Algorithm,' Comm. ACM (11) 6 p. 419.

14. English Braille, American Edition, American Printing House for the Blind, Louisville, Ky. (1966).

15. COBOL General Information, F28-8053, International Business Machines Corp., N.Y.

16. United States Standard COBOL, American National Standards Institute, ANSI X 3.23 - 1968.

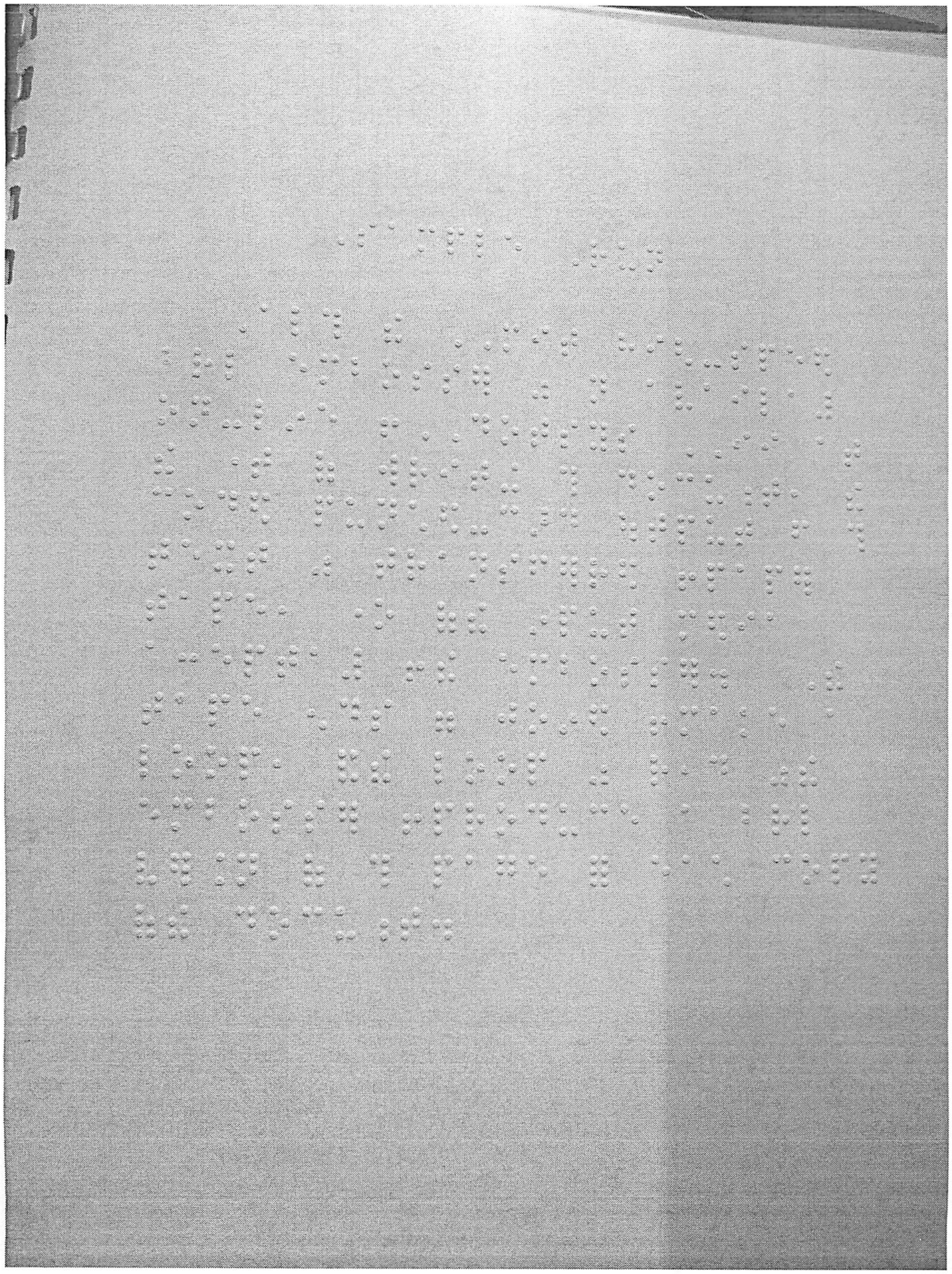
ATTACHMENT

A PAGE OF BRAILLE

The braille page reads:

Sample Run

Although the M.I.T. high-speed braille embosser was not available on-line to DOTSYS II at the time of writing this document, the coded punched-card output from the text was transferred to paper tape, one of the input media acceptable to the embosser. The tape ends were joined to make a loop, and the loop was read by the embosser to produce a braille version of this page for each copy of the document.



[illegible]

DISTRIBUTION LIST

INTERNAL

C-01

C. W. Farr

D-06

P. R. Vance

D-07

J. H. Burrows
A. J. Roberts

D-11

J. J. Croke
C. E. Duke
J. F. Jacobs

D-12

C. A. Zraket

D-53

F. Engel

D-63

R. S. Nielsen

D-73

W. Amory
N. A. Anschuetz
E. H. Bensley
J. A. Clapp
T. L. Connors
C. G. Crothers
M. T. Gattozzi
W. R. Gerhart
R. A. J. Gildea (25)
J. B. Glore
O. R. Kinney
E. L. Lafferty
Dr. J. K. Millen (25)

D-73 (cont.)

J. Mitchell
C. M. Sheehan
J. E. Sullivan
N. B. Sutherland
L. M. Thomas
Dr. D. E. Walker
E. W. Williamson

PROJECT

G. Dalrymple, M.I.T.
M. Leonard, M.I.T.
Prof. R. W. Mann, M.I.T.
V. A. Proscia, M.I.T. (50)

EXTERNAL

H. Bassler
Colonial Penn Insurance Co.
112 South 16th Street
Philadelphia, Pennsylvania 19102

Dr. M. P. Boyles (3)
Director, Computer-Braille Project
Instructional Services Center
Atlanta Public Schools
2930 Forrest Hill Drive, S.W.
Atlanta, Georgia 30315

L. L. Clark (2)
Director, IRIS
American Foundation for the Blind
15 West 16th Street
New York, New York 10011

E. L. Glaser
Computation Center
Case Western Reserve University
University Circle
Cleveland, Ohio 44106

DISTRIBUTION LIST (cont.)

EXTERNAL (cont.)

Dr. C. E. Hallenbeck
Department of Psychology
University of Kansas
Lawrence, Kansas 66044

R. Haynes
American Printing House for
the Blind
1839 Frankfort Avenue
Louisville, Kentucky 40206

Dr. K. R. Ingham
Room 20-B-207
Massachusetts Institute
of Technology
77 Massachusetts Avenue
Cambridge, Massachusetts
02139

R. E. LaGrone
IBM Corporation
Federal Systems Division
Department PC4, Room 2P25
18100 Frederick Pike
Gaithersburg, Maryland 20760

Dr. L. Leffler
Applied Mathematics Division
Argonne National Laboratories
9700 South Cass Avenue
Argonne, Illinois 60440

R. J. McNaughton
RCA Aerospece Systems Division
Burlington, Massachusetts
01801

Prof. A. Nemeth
Mathematics Department
University of Detroit
4001 W. McNichols
Detroit, Michigan 48821

Dr. B. Perella
Department of Defense
Fort George Meade, Maryland

A. Schack
Schack Associates
127 West 12th Street
New York, New York 10011

J. Siems
American Printing House for the
Blind
1839 Frankfort Avenue
Louisville, Kentucky 40206

Dr. E. J. Waterhouse
Director
Perkins School for the Blind
175 North Beacon Street
Watertown, Massachusetts 02172

V. Zickle
American Printing House for the
Blind
1839 Frankfort Avenue
Louisville, Kentucky 40206

FOREIGN

P. W. F. Coleman
4 George Street
Eastleight, Hants
S05 4 BU, United Kingdom.

C. W. Garland
Royal National Institute for
the Blind
224-6 Great Portland Street
London, W. 1, England

DISTRIBUTION LIST (conc.)

FOREIGN

R. House
Research Group for Quantitative
  Linguistics
Sodermalmstorg 8, Stockholm
Sweden

Dr. G. Lamprecht
Wistfalische Wilhelms Universitat
44 Munster, Germany
Roxeler Strasse 64

B. Lindqvist
De Blindas Forening
Utvechlingsavdelningen
Gotlandsgatan 46
116 65 Stockholm
Sweden

J. Lindstrom
Handikappinstitutet
Pa Iach, 161 03 Bromma 3
Sweden

V. Mokleby
Husby OFF Skole for Blinde
Hovseterveien 3
Oslo 7
Norway

W. Sorke
Deutsche Blindenstudienanstalt
355 Marburg/Lahn
Am Schlag 8
West Germany

J. Vinding
Statens Institut for Blinde
  og Svagsynede
Rymarksvej 1
2900 Hellerup
Denmark

Prof. H. Werner
Institut fur Numerische und
  Instrumentelle Mathematik
Universitat Munster
Schlossplatz 5
44 Munster, West Germany