DOTSYS III: A PORTABLE BRAILLE TRANSLATOR

by

Joseph E. Sullivan

Background

The subject of Braille translation is not a widely familiar one, so that a brief introduction to that topic would seem to be in order before taking up DOTSYS III. Most people know what Braille is in general - a coding system employing raised dots so that the sense of touch alone suffices to read. However, contrary to the impression one gets from those little cards, the most widely used codes are not "substitution ciphers" - that is, the Braille equivalent of a given "inkprint" text is generally not a simple transliteration but rather a kind of translation. This is because the rules for transcription involve not only the spelling of words, but also their syllabification and pronunciation. And since these occasionally vary with the meaning of the word (e.g., the verb "do" and the musical note "do") the transcriber must understand what he is transcribing, at least at a superficial semantic level. This is characteristic of translation, although of course the process is not nearly as difficult as the translation of one natural language to another.

The actual rules of Braille will not be treated in any great depth here; the various codes differ from language to language and, even within a language, from situation to situation. In this country there are three standard codes: one for general, "literary" use[1], one for scientific notation[2], and another for musical notation[3]. A fourth standard[4] covers formatting rules. In addition to these, a number of more-or-less widely used codes exist, ranging from simple transliteration codes such as "Grade 1" and "computer Braille" to a Braille shorthand. Unless otherwise indicated, we shall be referring to the literary code, sometimes called Grade 2 Braille. This code requires some sixty pages of rules to define.

The basic unit of Braille is a "cell" comprising six dot positions, three positions high and two wide. These are labeled 1-6, counting downward first on the left and then on the right. Each position may have a dot or not; thus there are $2^6=64$ possible combinations. In a given piece of text, there will typically be a number of cells that stand one-for-one for characters in the inkprint counterpart. However, the general rule is that a group of one or more Braille cells will correspond to a group of one or more inkprint characters. In order to reduce the space required by Braille and correspondingly improve the reading rate, the rules are so contrived that the Braille group is almost always shorter. The chief device for doing this is the contraction. One-hundred-eighty-nine commonly-occurring letter groups have a short Braille form that substitutes for full spelling in some circumstances. Generally, these circumstances are dictated by the fact that a given Braille sign may have different meanings depending on context - e.g., the sign for a period and the sign for the letter-group "dis" are one and the same, so that a contraction may be made when the letters appear at the beginning of a word (e.g., "dislike") but for the obvious reason full spelling must be used when they occur at the end ("Charybdis"). Braille is riddled with such context dependencies, including "case" and "escape" codes to permit representation of inkprint effects such as italics and to shift interpretation modes - even the digits use the same signs as the letters a - j. However intricate rules covering these cases may be, they are still mechanical and thus not the worst complication in Braille translation. The difficulty lies in a group of rules that forbid use of contractions where syllabification or pronunciation would likely be thrown off; for example, the "the" contraction should not be used in "sweetheart" because the letter-group straddles a strong syllable boundary. And, as previously noted, syllabification and pronunciation cannot always be determined without ascending to the level of meaning.

General Function of DOTSYS III

DOTSYS III is a computer program to translate inkprint into Braille as automatically as possible. But since the determination and representation of deep semantics is beyond the current state of the programming art - at least for natural languages used in unrestricted context - it is clear that a program to produce perfect Braille automatically is not technically feasible. DOTSYS III represents a compromise between the perfect and the possible. The occasional flaws in its translation can be removed by human intervention or, in some situations, simply left in as an acceptable level of "misprints".

The program proper does not embody the rules of English Braille, but rather a generalized process appropriate to the larger inkprint-to-Braille problem. A set of tables specify the process for a particular code or related group of codes. A single set of tables (with many versions!) presently implements English Braille, with intermixed Grade 1 and computer Braille permitted. It appears that a table for Grade 2 Spanish Braille can be implemented quite easily. This flexibility of changing translation rules by altering tables is motivated not only by the possibility of switching codes but also by the expectation that, even for a particular code, a long process of refinement (with occasional special-purpose temporary modifications) will take place.

Thus the program was designed with two primary goals: fidelity to the standard Braille rules and flexibility to adapt to the various codes, their interpretations, and application contexts. Two additional goals were: "portability" of the system from environment to environment, and comprehensibility of the system so that it would be a known quantity to its users, and to facilitate local adaptation. It was chiefly with portability in mind that COBOL was chosen as the system programming language[11], even though languages more capable in other respects were available.

The Algorithm

The basic algorithm is a modified finite-state process devised by Dr. Jonathan Millen[5]. It is perhaps easiest to understand the algorithm by observing it in action on a particular case, using the English Braille tables.

First, let us assume that some text has already been translated. The input, seen through a 10-character "sliding window" (underlined), looks like

. . . THIS <u>DISEASE IS</u> NOT SERIOUS . . .

The first step is to locate a qualifying entry in the main translation table, which in this case contains about 1,000 entries. In principle, though not in actuality, the table is searched serially. The entry in question is:

DIS| L 6 3 50 - - -

In order for an entry to qualify, three distinct tests must be passed. First, the left end of the window must match the entry as far as the "|". Secondly, the next character beyond this point must fall into the "right-context class" designated by the second field of the entry. In this case, "L" has been defined (by another table) to mean "letter". Thus it is clear at a glance that the first two qualifications are met. The third qualification is determined by the "input class" - in this case, class 6. This class selects an entry in a decision table and in turn this entry imposes a boolean test on a set of conjunctions. The state variables themselves are simply true-false indicators deriving "meaning" only from the logic of a particular set of tables. For the literary tables, the set of state variables and their approximate meaning is as follows:

1) after the start of a number

2) after the start of a word

3) Grade 1 translation

4) inside a quotation

5) inside italicized text

6) Not at the start of a prefix or stem

7) part-way through a word or phrase too long for one entry

8) just after a space or A-J, following a number

Getting back to our example, the decision table entry for class 6 will demand that states 2, 3, and 7 all be "off" - i.e., we must be at the beginning of a word, in grade 2 translation, and not looking for the second part of a double entry. All these will apply in this case, so the entry qualifies. Had it not qualified for any reason, then the translation table search would have continued; and if no entry qualified, the single leftmost character would be translated according to an assumed (default) entry.

Once an entry is accepted, three actions take place. First, up to four Braille signs and/or program control codes are issued. In this case "50" selects the dots 2-5-6 cell representing the contraction "dis". Second, the window is shifted a certain number of places, in this case, 3:

. . . THIS DISEASE IS NOT SERIOUS . . .

Finally the input class selects an entry in the transition table. This entry determines the new setting of each state variable: set, reset, changed or unaltered. In this case state variable l, 7, and 8 will be reset, variables 2 and 6 will be set, and the rest will be left unchanged.

The input class is thus the key to the state setting and testing logic. It tends to be the same for most contractions of a given type - class 6, for example, is characteristic of those contractions used only at the beginning of a word.

It may be noted at this point that "disease" is one of those odd words whose contraction depends on meaning. When used in the usual sense, the next two letters (ea) should be contracted as a dot 2 cell. However, when used in the archaic sense of "lack of ease", the following rules combine to forbid the use of the contraction:

Rule X Par. 42: "The lower-sign contractions for ea and (certain others) . . . must never begin or end a word." Hence "ease" is spelled out.

Rule X Par. 34 b(1): ". . . a contraction must not be used where the usual Braille form of the base word would be altered by the addition of a prefix or suffix."

Naturally, the standard table will contract the ea. If the input preparer knew that the archaic meaning was intended, he could flag the input to inhibit contraction (as will be discussed later under "Special Features") or, if a large work used the term so often that this approach would be irksome, another entry - perhaps

```
DISEASE    -    6    6    50170114
```

inserted before the DIS entry, would suffice to alter the logic for just this word and its derivatives.

Organization of the Program

Although the process just described is the heart of the matter, there are a good many other details for the program to attend to: input handling, output control for a variety of devices, line and page formatting including running titles, pagination and special formats, and provision for special rules, one of the most troublesome is the requirement that a numeric quantity followed by a unit of measure must be represented in a standard form with the measure first following by the quantity. The order reversal is controlled by the tables but provision to perform this operation must be supplied by the program.

The program comprises five modules, or, if you prefer, "levels of abstraction". The processor at each level is organized around some characteristic resource, and operates as a logical coroutine, sharing narrow interfaces with the other levels. At least, that is the basic idea - or rather, the idea that crystallized after too much of the program was running to consider a strict restructuring along these lines. Nevertheless, these structural modularity concepts are present in sufficient degree to form a model for explanation.

The first module is the primary input processor. Its main job is to accept the inkprint text stream, discarding certain extraneous control markings (such as record boundaries) and filler blanks, and produce the "window" for the translator. When "self-checking" is in effect (see the next section), the input processor also computes the "correct translation" of each inkprint word and places it on a ring buffer for later comparison.

The second module is the translator itself. The main logic has already been discussed. The output of this processor is simply the stream of Braille signs and special control codes exemplified by the "50" in the sample shown.

An important submodule of the translator is the main table look-up, which implements a logically serial search but capitalizes on the fact that whole portions of the table can often be bypassed when failure to match the window contents occurs on a particular entry. There is an input ordering restriction: entries whose first n letters are alike must be grouped together. However, this does not reduce the generality of logic supported by the table. Using at most one extra pointer per entry, "chains" are set up so that all first entries for the characteristic second-letter groups are linked and so forth. A significant reduction in search time, at minimal cost in space, is effected by this technique.

A third module, the stacker, operates on the codes issued by the translator. In the simplest case, these codes are merely accumulated until the code for the blank Braille sign is received - i.e., a Braille word is finished. This word, constituting one entry in the stack, is normally then removed from the stack and issued as output to the line composer module. In some circumstances, two or more words may be held in a stack before release. This may be to effect order reversal, or to group words for running titles or centered headings. If "self-checking" is in effect, it is the stacker that compares the actual translation with the correct translation left on the ring buffer by the input processor. The ring buffer allows for the logical lag in the various processing stages, and permits recovery from synchronization problems that occur when, as occasionally called for in the rules, several inkprint words are run together into one Braille "word".

The line composer, the fourth module, operates on the Braille words produced by the stacker. It concerns itself with arranging the words on a Braille line, starting a new line or new page when necessary, page numbering and titles, and so forth.

The fifth module, the output writer, is actually a collection of processors, one for each distinct mode of output, as listed in the next section.

SPECIAL FEATURES

Formatting

The program incorporates a number of special formatting options, including automatic pagination and numbering, centered titles, and a "poetry" (hanging indent) format. An extensive system of "typed" tabs permit easy alignment of columnar material on the left, right or decimal point within the entries. In general, the selection of options, the setting of modes and tab values, and the issuing of direct format controls (e.g., "skip to new page") is accomplished by commands embedded in the input text.

Output Devices

At present, the program can drive five output devices, or any combination. Two of these are "printer Braille": a form of embossing accomplished by printing periods on a standard line printer with a resiliant backing inserted between the paper and the platen. The periods dimple the paper so that the reverse is a crude form of Braille. With a printer set for normal spacing the size of the Braille cell is far off the standard, but a special IBM modification, in conjunction with slightly different program output logic (hence the two "devices"), can bring the spacing back into acceptable range. This is the technique used by the Atlanta Public Schools.

A high-speed embosser developed at MIT[6] can also be driven directly by the program. In a typical configuration, the program is running under a time-sharing system such as Interactive Data Corporation's virtual machine system, with the embosser slaved to the user's terminal teletype. The teletype can be used for entry and editing of text as well as an intermediary for the embosser output.

One form of output, the "proof", is intended for the use of a sighted editor. The proof contains an echo of the input and the resulting Braille printed as dots.

Finally, a card output, containing numeric codes for the Braille signs, is provided for general interface purposes.

Direct Translation Control

Although not strictly a function of the program, the standard tables provide a means for inhibiting a contraction that would otherwise be made, or for forcing a contraction that would normally not occur. In most cases of mistranslated words, this is the easiest way to correct the Braille.

Self-Checking

Early in the development of DOTSYS III, it became apparent that some automatic means was necessary for checking on the quality of Braille translation achieved by a given table - and especially after a series of changes, to see that no unwanted side effects were introduced. This need was met by preparing an input text containing 5,808 "problem" words used in training human transcribers[7], with special markings to indicate the correct translation. The program compares the normal output with the correct Braille, flagging misses with a code in Braille and a large TSK on the proof output.

HISTORICAL NOTES AND WORK TO BE DONE

DOTSYS III is by no means the only Braille translation program, although it is one of very few that attempts to produce Braille in accord with the existing standards. The programs at the American Printing House for the Blind[9, 12, 17] were in use for some years prior to DOTSYS, but unfortunately they are in machine language and hence not portable. A later PL/I program by Dr. Lois Leffler[13] also adheres to the standard.

Those programs that deviate from the standard usually do so for eminently practical reasons, for as we have seen the Braille rules are not designed for easy automation, and in many situations quantity and speed is simply an overriding consideration. Beyond this, there are those who argue that the rules are not even well designed from the human standpoint, and that a major redesign friendly to man and machine alike is called for.[14] Despite much technical merit in such proposals, it is clear that the investment already sunk into training and literature inventory (and - yes - computer programs!) will weigh heavily in favor of the existing standards, with perhaps some local improvements, for some time to come.

The development of DOTSYS III was carried out at MITRE under the direction of Robert A. J. Gildea, presently chairman of SIGCAPH, under contract with the Massachusetts Institute of Technology Sensory Aids Center and the Atlanta Public Schools Computer Braille Project. Professor Robert Mann of MIT and Dr. Marion Boyles of Atlanta directed the project at their respective establishments.[16] Messrs Vito Proscia and George Dalrymple of MIT, and Robert Lagrone and Donald Bell of IBM and Atlanta, were also instrumental in various phases of the project, especially in helping to improve the translation quality while integrating the program into their operating environments.

The original technical development was Dr. Jonathan Millen's DOTSYS II program.[8] This pilot program implemented the basic algorithm, and even with a relatively simple set of tables produced Braille good enough that further development into a production capability was clearly in order. Mr. W. Reid Gerhart and the author thereupon set out to refine the table and to retrofit the program with most of its present formatting capability, including units-of-measure order reversal. In the process, the table search was improved to its present form and self-checking was introduced. Some modularity was introduced into the output section, so that new device types could be introduced easily. In fact, the MIT Braillemboss module was not added until after the first delivery of the program to Atlanta in August 1970. That brought the program to essentially its present form, although a number of minor fixes and improvements have been made since, many by the users themselves. In this regard Dr. Judy Gunnerman and Messrs Philip Bagley and John Covici of Information Engineering deserve special mention for considerable work with the program, including the preparation of some additional program logic documentation.

As expected, evaluation of the tables is a continuing process. Most of the entries are determined in the light of the hindsight arising from a particular mistranslation. A table of contexts for contraction use[15] is typically used to aid in making the table entry specific to the desired cases, and the self-checker is used to verify that no bugs have been introduced.

It might be well at this point to examine how well DOTSYS III has met its goals of fidelity, portability, flexibility and comprehensibility. As for fidelity, there seems to be general agreement that a consistently high quality Braille is achieved. Even in the 5,808 "problem" word text, for example, fewer than 200 mistranslations were made - most of them on farfetched concoctions such as "abecedarian" and the archaic "disease". In more typical text, the error rate is fewer than one in ten pages. Portability also seems to have been successfully achieved, thanks largely to the ubiquitous COBOL, although the size of the program (59K bytes, with the tables, on a 360/370) is more restrictive on minimum size than we had hoped would be the case.

Flexibility of the tables is a built-in fact, but flexibility of the program itself to add new formatting capabilities, or new output devices appears to work out only for the original programmers, as in the case of the MIT embosser. This speaks unfavorably of the program's comprehensibility, which seems to be the chief user complaint despite extensive functional documentation.[10] Some of the difficulty may no doubt be chalked up to the mutual interference of three "cooks" working on one program, or the fact that none of the three had programmed in COBOL before (and they haven't since), or the verbosity and lack of block structure in COBOL, or simply to the fact that perhaps 80 percent of the present 1,600 lines of code is retrofitted. In the opinion of the author, however, the basic problem was a very familiar phenomenon: the proper structure of the program was not and could not, as a practical matter be recognized until the point was past where a total restructuring was thinkable. Deadlines, a tight budget and a bird-in-hand in the form of running code can conspire in a sort of tyranny over one's options despite what is obviously desirable technically.

In any event, this issue has been revived in the form of an effort, under Air Force auspices, to rewrite DOTSYS III as a case study in the principles of structured programming.

Finally, plans are being made to expand the scope of application from the English literary code to other languages and possibly the scientific (Nemeth) code. The former will probably require no more than a table change in most cases. For the latter, a preprocessor will probably be necessary to convert from a reasonable input language, yet to be designed, into a text that defines complicated formats unambiguously and is acceptable to DOTSYS III and some set of tables.

References

1) English Braille, American Edition 1959. Revised 1968. American Printing House for the Blind, Louisville, Kentucky, 1969.

2) The Nemeth Code of Braille Mathematics and Scientific Notation. American Printing House for the Blind, Louisville, Kentucky 1965. (Revision to be published in 1973)

3) Spanner, H. V. (Compiler). Revised International Manual of Braille Music Notation, 1956. American Printing House for the Blind, Louisville, Kentucky 1961.

4) Code of Braille Textbook Formats and Techniques. Revised 1970. American Printing House for the Blind, Louisville, Kentucky 1970.

5) Millen, Jonathan K., Finite-State Syntax-Directed Braille Translation. Technical Report MTR-1829, MITRE Corporation, Bedford, Massachusetts July 2, 1970.

6) Development of a High-Speed Brailler System for More Rapid and Extensive Production of Informational Material for the Blind. (Final Report) Sensory Aids Evaluation and Development Center, Massachusetts Institute of Technology, Cambridge, Massachusetts September 29, 1970.

7) Krebs, B. M., Transcribers' Guide to English Braille. The Jewish Guild for the Blind, New York, New York 1967.

8) Millen, Jonathan K., DOTSYS II User's Guide and Transfer and Maintenance Manual. Technical Report MTR-1853, MITRE Corporation, Bedford, Massachusetts July 2, 1970.

9) Schack, Ann S. and Mertz, R. T., Braille Translation System for the IBM 704. Mathematics and Applications Department, IBM Corporation, New York, New York 1961.

10) Gerhart, W. Reid; Millen, Jonathan K., and Sullivan, Joseph E., DOTSYS III: A Portable Program for Grade 2 Braille Translation. Technical Report MTR-2119, MITRE Corporation, Bedford, Massachusetts May 14, 1971.

11) Millen, Jonathan K., Choice of COBOL for Braille Translation. Technical Report MTR-1743, MITRE Corporation, Bedford, Massachusetts, December 1969.

12) Siems, John R., Report of New Braille Translation Program at APH. Proc. Conf. on New Processes for Braille Manufacture. American Printing House for the Blind, Louisville, Kentucky 1968.

13) Leffler, Lois C., The Development of a Computerized Grade II Braille Translation Algorithm. Wescon Technical Papers 1971, Session 30. August 24, 1971.

14) Allen, Jonathan and Borroz, W. Terry, Recent Improvements in Braille Transcription. Proc. ACM Conference, 1972.

15) Key Braille Contraction Contexts. American Printing House for the Blind, Louisville, Kentucky 1969.

16) Boyles, Marion P., and LaGrone, Robert E., Computer Braille Translation of the Atlanta School System. Wescon Technical Papers 1971, Session 30. August 24, 1971.

17) Haynes, Robert L., An Automated Braille Translation System. Wescon Technical Papers 1971, Session 30. August 24, 1971.