triroff, an adaptation of the device-independent troff for formatting tri-directional text

ZEEV BECKER (זאב בקר ゼエブ・ベケル 則富北克爾) AND DANIEL BERRY (דניאל ברי ダニエル・ベリ 丹尼儿北利)
Computer Science Department
Technion
Haifa 32000
Israel

SUMMARY

This paper describes a system for formatting documents consisting of text written in languages printed in three different directions, left-to-right, right-to-left, and top-to-bottom. For example, this paper is such a document because it contains text written in English, Hebrew, Japanese, and Chinese. The system assumes that the input is in the order in which the text is read aloud, and it produces output in which each language is printed in its own correct direction, but for which a human cognizant of the reading conventions will reproduce the input order. The system consists of three major pieces of software: Ossana and Kernighan's ditroff for formatting text consisting of only left-to-right or unidirectional text, Buchman and Berry's ffortid for rearranging right-to-left language text buried in ditroff output to be printed from right to left, and a new program bditroff for arranging that top-to-bottom text buried in ditroff output is printed from top to bottom.

Below are translations of this English language abstract, except for this paragraph, into Hebrew, Japanese, and Chinese. The latter two are each printed twice, once in a modern left-to-right style, and once in a more traditional top-to-bottom style. The software described in this paper was used to format and typeset this paper.

תקציר

מאמר זה מתאר מערכת לעריכת מסמכים המכילים שפות הנכתבות בשלושה כוונים שונים, משמאל לימין, מימין לשמאל ומלמעלה למטה. לדוגמא, מאמר זה הנו מסמך כזה, מכיוון שהוא מכיל טכסט הכתוב באנגלית, עברית, יפנית וסינית. המערכת מניחה כי הקלט נתון בסדר בו הוא נקרא בקול רם, והיא מפיקה פלט שבו כל שפה מודפסת בכוון הנכון, אך אדם המודע למוסכמות הקריאה ייצר מחדש את סדר הקלט. המערכת מורכבת משלושה חלקים עיקרים: ditroff של Ossana ו־Kernighan לעיבוד טכסט הכתוב רק משמאל לימין או בכוון אחד, ffortid של Buchman ו־Berry לסידור הטכסט הכתוב מימין לשמאל הטמון בפלט של ה־ditroff כך שיודפס מימין לשמאל, ותכנית חדשה bditroff, לסידור הטכסט מלמעלה למטה הטמון בפלט של ה־ditroff כך שידפס מלמעלה למטה.

アブストラクト

左から右、右から左、上から下、の異なる３つの方向に印字される複数の言語によって記述されたドキュメントをフォーマットするシステムを紹介する。例えば本論文は、英語、ヘブライ語、日本語、中国語で記述されており、そのようなドキュメントの１つである。本システムは、テキストの読まれる順に入力が与えられたものとして、それぞれの言語が正しい方向に印字されるように出力を生成するものであるが、読み方を知っている人間なら、出力から入力順序を再構成することも可能である。本システムは、左から右または単一方向のテキストのみからなる文書をフォーマットするための、Ossanaと Kernighanの ditroff、ditroffの出力に埋め込まれた右から左のテキストを正しく印字するための、Buchmanと Berryの ffortid、そして、ditroffの出力に埋め込まれた上から下のテキストを正しく印字するための、新しいプログラム bditroff、の３つの主要ソフトウェアから構成されている。

0894–3982/89/030003–24$12.00
© 1989 by John Wiley & Sons, Ltd.

Received 22 August 1988
Revised 22 November 1989

アブストラクト

左から右、右から左、上から下、の異なる３つの方向に印字される複数の言語によって記述されたドキュメントをフォーマットするシステムを紹介する。例えば、本論文は、英語、ヘブライ語、日本語、中国語で記述されており、そのようなドキュメントの１つである。本システムは、テキストの読まれる順に入力が与えられたものとして、それぞれの言語が正しい方向に印字されるように出力を生成するものであるが、読み方を知っている人間なら、出力から入力順序を再構成することも可能である。本システムは、左から右または単一方向のテキストのみからなる文書をフォーマットするための、Ossanaと Kernighanのditroff、ditroffの出力に埋め込まれた右から左のテキストを正しく印字するための、BuchmanとBerryのffortid、そして、ditroffの出力に埋め込まれた上から下のテキストを正しく印字するための、新しいプログラムbditroff、の３つの主要ソフトウェアから構成されている。

擇要：

在　些情況下，一　文件可能會用上幾種不同語文，而本章就是介紹如何替此等正文輸入正當的格式，不同語文有不同的編排規則，即如此文便可作一例，因文中同時套用了英文，　伯　文，日文及中文，故此同一文件之中，便有三種編排格式，分別爲：由右至左、由左至右、及由上至下，此類系統的排列，蓋以誦讀時的排序爲原則，從而輸入合乎各語文的編排方向，但操作人員必須熟識各語文的傳統格式，此類編制格式系統，主要有三部　軟件，分別爲：Ossana and Kernighan's ditroff（用作　理由左至右或單一方向排列的正文），Buchman and Berry's ffortid（用作　理貯於），bditroff（的排列爲上至下的正文）。

擇要：

在　些情況下，一　文件可能會用上幾種不同語文，而本章就是介紹如何替此等正文輸入正當的格式，不同語文有不同的編排規則，即如此文便可作一例，因文中同時套用了英文，　伯　文，日文及中文，故此同一文件之中，便有三種編排格式，分別爲：由右至左、由左至右、及由上至下，此類系統的排列，蓋以誦讀時的排序爲原則，從而輸入合乎各語文的編排方向，但操作人員必須熟識各語文的傳統格式，此類編制格式系統，主要有三部　軟件，分別爲：Ossana and Kernighan's ditroff（用作　理由左至右或單一方向排列的正文），Buchman and Berry's ffortid（用作　理貯於），bditroff（的的排列爲上至下的正文）。

KEY WORDS Document Preparation Multi-lingual Multi-directional Troff Typesetting

BACKGROUND

With variants of the UNIX™ operating system spreading throughout the world[27], there is a concern to adapt the various UNIX facilities to be useable in a variety of languages and in mixed language environments[13]. There has already been much work done in multi-lingual formatting. For example, Becker describes one such what-you-see-is-what-you-get (WYSISYG) formatting system[4]. This paper describes new software whose general goal is to help adapt the facilities of the UNIX device-independent troff, known as ditroff[23, 16], to the multi-lingual environment.

The ditroff system is composed of a basic formatter called ditroff[16, 23] plus a number of preprocessors, postprocessors, and macro packages. Among the preprocessors are

1. refer, for handling bibliographical citations[20]
2. ideal, for drawing pictures[30],
3. pic, for drawing pictures[15],
4. grap, for plotting graphs[6],
5. drag for drawing directed graphs[25],
6. flo for drawing flowcharts[29],
7. psfig for including figures drawn in POSTSCRIPT™[3],
8. alg and its derivatives for formatting included program code[7]
9. tbl for laying out tables[21], and
10. eqn for laying out mathematical formulae[14].

Among the postprocessors are

1. all the various device drivers for translating ditroff output into the instructions needed to print the formatted documents on the various printing devices, ranging from line printers, dot matrix printers, laser printers, through to phototypesetters,
2. all the various software which allows preview, on a high resolution screen, of how a document will appear when printed on some printing device, and
3. indx[1], for preparing a back-of-document index.

Finally, the macro packages include mm, man, ms, me, and mX, which are described in the various versions of the *UNIX Programmer's Manual*[26]. The primary advantage of the ditroff system over other more monolithic systems is precisely that it is not monolithic and is composed of many programs, which may be combined for application to a document to get their combined functionality. It is relatively easy to add new programs with new functionality as evidenced by the ever growing collection of pre- and postprocessors cited above. Thus, it is relatively easy to experiment with new functionality in the context of a full-function system.

These formatting programs were developed in a primarily English-speaking environment. However, in principle, these programs can be used in conjunction with any language written from left-to-right with lines flowing top-to-bottom for which fonts are mounted on the printing device.

The goal of the authors and their colleagues has been to adapt the ditroff collection to the multi-lingual setting. A ditroff postprocessor, ffortid™[2], has been developed to make the collection useable as a bi-directional formatting system, in conjunction with Arabic, Farsi, and Hebrew fonts. In addition, the ability to handle very large character sets, such as those used in Japan, Korea, and the People's Republic of China, requiring two bytes for encoding characters, has been added[12]. Interestingly, the three character sets from these countries, the JIS, KS C 5601, and GB-2312 standards are all arranged as 94×94 matrices. Given that all characters in all of these character sets are exactly the same square size, the width tables and the processing for these character sets are nearly identical.

For the purpose of identifying groups of languages with similar formatting problems, the following group names are used in this paper.

1. The group of languages, including English, whose members are printed with alphabets of size less than 256 in the left-to-right direction is called the *Latin* languages, even though it includes many languages, such as Greek, and Russian, not written with the Latin alphabet.
2. The group of languages, including Arabic, Farsi, and Hebrew, whose members are printed with alphabets of size less than 256 in the right-to-left direction is called the *Middle East* languages, even though it includes languages, such as Urdu, whose locale is not really in the Middle East.
3. The group of languages including Chinese, Japanese, and Korean whose members are printed with alphabets of size greater than 256, traditionally in the top-to-bottom direction, is called the *Far East* languages.

There are a number of writing directions dealt with in this paper, they are identified as follows.

1. The direction of writing in which the characters flow from left to right on a line and the lines flow from top to bottom is called the *left-to-right* direction.
2. The direction of writing in which the characters flow from right to left on a line and the lines flow from top to bottom is called the *right-to-left* direction.
3. The direction of writing in which the characters flow from top to bottom on a line and the (vertical) lines flow from right to left is called the *top-to-bottom* direction.
4. Together the left-to-right and right-to-left directions are called the *horizontal* directions while the top-to-bottom direction is a *vertical* direction.

THE NEED FOR BI- AND TRI-DIRECTIONAL FORMATTING

Throughout the Far East, documents are written containing a mixture of text in Far East and Latin languages. The Latin language text may include mathematical formulae. It is often desired to print the Far East language text in the traditional top-to-bottom direction. While it is possible to print the Latin language text letter-by-letter in the same direction, it is preferable to print the Latin language text in its traditional left-to-right direction.

Moreover, in Hong Kong and Japan, newspapers and magazines have their main-line Far East language written in the top-to-bottom direction and their headlines written in the right-to-left direction. These same newspapers and magazines have advertisements using Latin language text written in the left-to-right direction. Thus, in one document, text is written in three directions, left-to-right, right-to-left, and top-to-bottom.

In the Xinjinang Uighur autonomous region of the People's Republic of China, inhabitants speak the languages, Uighur, Kazak, and Kirgiz[28], which are written in the right-to-left direction, as are their linguistic cousins Arabic, Farsi, and Urdu. When combined with the general use of Chinese and Latin languages in the country, the need arises in the region for tri-directional formatting. For example, at universities in the region, a technical paper might very well be written in a local language, use English for technical words, have formulae, and be required to have a Chinese abstract.

Finally, any business contract for high technology work done jointly by companies from a Far East country and a Mid East country could require tri-directional formatting.

There is an additional language used in the Xinjinang Uighur region, whose correct traditional printing direction was learned only after the software was written and the referees' comments on the first draft of this paper had been received. Mongolian is the language, and in its traditional writing direction, the characters flow from top to bottom on a line, and the lines flow from *left to right*! How this direction can be handled is discussed later in the section on weaknesses; implementing this solution is left for future work.

Note that if the tri-directional formatting problem is solved, then any bi-directional sub-problem, e.g., left-to-right and right-to-left, left-to-right and top-to-bottom, and right-to-left and top-to-bottom, is also solved.

THIS PAPER

This paper describes a pair of programs that enhance the ditroff collection to be tri-directional. One of these, ffortid, which provides the right-to-left formatting capability, was described in detail in an earlier paper[2]. The second of these, bditroff, which provides the top-to-bottom formatting capability, is the focus of this paper. The sequential, piped composition of ditroff with these two programs is called triroff. By enhancing an existing full-function formatting system, it is intended to be able to use the existing system's preprocessors and macros with no change. Indeed, this very paper was typeset camera-ready for this journal on a Linotronic™ 300 at 1250 dpi with the help of refer, pic, eqn, ditroff, ffortid, bditroff, and variants of the ms and mX macro packages that were developed for this journal.

The plan for the rest of the paper is as follows. Existing software is surveyed in order to be able to determine desirable properties of a tri-directional formatting system. Then it is possible to identify a basic structure that allows these properties to be met. The basic structure proposed works because of certain observable invariants in tri-directional text. An algorithm that exploits these invariants is given. The algorithm allows a tri-directional formatter to be built on the existing full-function ditroff system. It is explained how the algorithm and the underlying ditroff system can be exploited by the layout designer to achieve most desired effects. Some of these effects are illustrated by examples. However, not all desired effects are achievable; the weaknesses of the present system are identified and solutions are proposed for them. Implementation of these solutions are left to future work.

EXISTING SOFTWARE

There are a variety of systems for formatting Japanese or Chinese printed from left to right mixed with other left-to-right languages, e.g., English. These include Kameyama and Hasebe's jtroff[17, 11], Nagashima and Kawabata's early adaptation of TEX™[19] for Japanese[22], Saito's jTEX[24], and Berry's and Chow's adaptations of ditroff[12, 8]. These can also print Japanese and Chinese from top to bottom simply by printing from left to right in landscape mode with a font consisting of the characters rotated 90° counterclockwise. All of these systems use the standard 94×94 matrix arrangement of the

Japanese or Chinese characters, as the case may be. The first four of these formatters have modified the base program, troff or TEX, so that two-byte character codes are acceptable as input. The two-byte codes are generally distinguished from ASCII characters by having the eighth bit turned on in each of the two bytes. The latter two formatters avoid having to modify the base program, ditroff, by considering each row of the matrix to be a separate font and addressing the *cd*th character of the *ab*th row as `\f(ab\(cd`. This, by the way, is the scheme adopted by the system presented herein, as it too is based on ditroff.

Among the products for Japanese and Chinese wordprocessing on the Macintosh™ are EgWord[9] and FeiMa[10]. Both EgWord, a Japanese word processor, and FeiMa, a Chinese word processor, seem not to be able to deal with bi-directional text. All of their examples are strictly left-to-right. However, since they run on a Macintosh computer with its standard user interface, there is nothing to stop the user from rearranging some text to be written from top to bottom, and cutting this text out and pasting it into another document whose text is printed from left to right. In addition, at least FeiMa gives the user a choice of printing direction when requesting the printing of a document; however, this direction applies to the whole document.

One paper from the Xinjinang Uighur autonomous region[28] describes UKKMC-DOS, a version of MS-DOS™ which is capable of accepting input in English, Uighur, Kazak, Kirgiz, Mongolian, and Chinese. The text of each language is displayed on the screen printed in its own proper direction as it is being entered. The system is a WYSIWYG system in which the screen appearance reflects exactly what is in the file. The text in the file is broken into pages. Each page is treated as a two-dimensional array of characters. When a page is seen on the screen, the array is displayed directly on the screen with each row of the array being displayed on a separate row of the screen. Entry of a Latin character causes cursor movement one position to the right both on the screen and in the array, entry of a Uighur, Kazak, or Kirgiz character causes cursor movement one position to the left, and entry of a Chinese or Mongolian character causes cursor movement one position downward. In the editor, one moves around this array to directly address each character in its own position.

Working with this system is rather straightforward precisely because the visual image is a very accurate model of the internal structure. One is formatting the text as it is entered, with the software taking care of most of the global formatting details, such as keeping to the line length and to the page length. It is easy to address each character for direct manipulation of the character.

However, certain formatting operations are difficult with this system, most notably changing the page sizing characteristics, e.g., the line length, and the page length. Because the formatting is done during entry, based on the current setting of line and page lengths, and the file is stored as it appears, changing either of these lengths means massive manual editing. If one could recover the original entry order of the characters from the text, the original entry algorithms could be applied to the stream of characters relative to the new line and page lengths. However, with raw printable characters, in which punctuation is not distinguished by language, complete recovery is impossible because of punctuational ambiguities. Thus a system in which the text is stored in the original input order with the original language information is more general. This same problem occurs with many, but not all, WYSIWYG systems, because many of them store the characters in the visual order.

DESIRABLE PROPERTIES OF A TRI-DIRECTIONAL FORMATTING SYSTEM

There appear to be a number of desirable properties that should be satisfied by a tri-directional formatting system that make it more general, more functional, easy to use, and easy to program.

In order to allow maximum formatting flexibility with easily changed sizing characteristics, it is necessary to store the characters of the document in the order entered (with backspacing corrections normalized out), and to let the formatting algorithm be applied at the time of printing. This suggests a batch formatting system, but it does not preclude a WYSIWYG system, so long as the formatting algorithm is fast enough to be applied on the fly, with little observable delay to the user. This order of input is called *time order*; it is the order in which the text is thought of as it is being written. It is the order in which the properly formatted text is read out loud by a human reader cognizant of the multi-directional text reading conventions. It is also the order in which the letters would appear on paper if all languages were written in the same, say left-to-right, direction. This time order is the input order that is assumed by a variety of multi-lingual systems, specifically those implemented by Joseph Becker[4, 5], by Pierre MacKay and Donald Knuth[18], and by Cary Buchman, Daniel Berry, and Jakob Gonczarowksi[2]. The groups doing these projects seem to have arrived, independently, at the conclusion that time order is best. That is, each group had written at least a draft of its paper and code before any of the others' papers had appeared.

Thus, from the input, shown in stylized form:

```
.ft R \"Roman
English
.in .5i \" effective ...
.ll 4.5i \" line length 4 inches
.br
.PR \" predominantly right-to-left
.ft HB \"Hebrew
עברית
.br
.PL \" predominantly left-to-right
.ft KT \"Katakana
カタカナ
.br
.BT \" begin top-to-bottom
.ft HR \"Hiragana
ひらがな
.sp
.ft CH \"Chinese
漢字
.br
.ET \" end top-to-bottom
```

assuming that English and Katakana are printed from left to right, Hebrew is printed from right to left, and Hiragana and Chinese are printed from top to bottom with the columns laid out from right to left, one gets as output something like:

English

עברית

カ タ カ ナ

ひらがな

漢字

To minimize both programming effort and user learning effort, it is useful that the formatting software be an upward compatible extension of an existing system. For maximum functionality, it is useful that the underlying existing system be a stabilized system capable of dealing with pictures, tables, graphs, equations, indices, tables of contents, bibliographical citations, program code formatting, indexing, etc. One example of such a system is the UNIX documentor's workbench (DWB™) or ditroff collection. Another is the TEX collection.

In this respect, it is best of all if the underlying system's software can be used *unchanged*. Then, only the new capabilities need to be programmed. Full functionality is obtained with no additional programming effort. Finally, the user community can rely on extant behavior being reproduced, even down to the bugs that have become features.

BASIC STRUCTURE OF SYSTEM

This paper describes a system for tri-directional formatting based on the UNIX DWB or ditroff collection. The system assumes input

1. in time order, and
2. with line breaks before and after each contiguous stream of constant width characters—in Far East language fonts, all characters are the same width—to be printed from top to bottom,[1] and
3. with the current language and direction being identified by the current font.

The system uses the standard, essentially unchanged underlying ditroff formatting program to make all the formatting decisions. Fonts for the standard character sets from Korea, Japan, and the People's Republic of China are arranged as a 94×94 matrix. For the purpose of ditroff formatting of Far East language text, the martix is treated as 94 fonts each containing 94 characters, all exactly the same size. Thus, the only changes to the ditroff program as distributed by AT&T are the changes of the constant defining the number of fonts mounted to a number large enough to accommodate the 94 row fonts plus whatever else is mounted on the local printing devices and of the constant defining the size of the character set to something large enough to accommodate the 94 characters

[1] To make sure that requiring line breaks before and after each block of top-to-bottom text is reasonable, we went to the neighborhood bookstore and bought a Japanese magazine with both left-to-right and top-to-bottom printing of Japanese. In all cases of switch of direction, there was an accompanying line break. That is there was no case of beginning top-to-bottom printing on the same horizontal line that contains the preceding left-to-right text, and there was no case of beginning left-to-right printing on the same horizontal line serving as the bottom line of the rectangle of the preceding top-to-bottom text.

per row in addition to whatever else is available for the device. Because the standard ditroff program is used, all existing preprocessors and macro packages still work. The system generates output in exactly the same format as is generated by the existing standard ditroff. Thus all existing postprocessors still work, and the formatted output can be printed on any existing device for which both a device driver and all the required fonts are available. The ability to format right-to-left and top-to-bottom text in addition to left-to-right text is created by the addition of two programs between the basic ditroff program and the device drivers. These two programs each accept as input the output of ditroff and produce output in the same form as ditroff output. Thus these two programs can accept as input each other's output as well and can send their outputs to the same postprocessors to which ditroff can send its output. Because the input to the underlying ditroff is in time order, ditroff's output reflects formatting decisions made as if all the text were written from left to right.

The first of the additional programs is ffortid, which, on a line-by-line basis, reorganizes the line of text so that each contiguous sub-line of text in left-to-right fonts is printed from left-to-right and each contiguous sub-line of text in right-to-left fonts is mirrored about its own center in its current position so that it is printed from right to left. To this program, top-to-bottom text is treated as left-to-right text. The second of the additional programs is bditroff.[2] On a page-by-page basis, bditroff reorganizes the text on a page so that each contiguous $n \times m$ rectangle scanned left to right, top to bottom, of text in top-to-bottom fonts is permuted to become an $m \times n$ rectangle scanned top to bottom, right to left. Because the scanning directions of the two rectangles are perpendicular to each other *and* the characters are all the same size, the $n \times m$ and the $m \times n$ rectangles actually occupy the same area on the paper. Thus the structure of the system is as given in Figure 1.

INVARIANTS THAT ALLOW THIS SYSTEM TO WORK

The simple, modular structure described in the previous section works because of a number of invariants that apply both to the text and its printing.

1. A given horizontal line on the page consists either of left-to-right and right-to-left text or of top-to-bottom text. This is the case because of the line breaks that are required at each point of changing from horizontal to vertical text or vice-versa.
2. While reading, within each contiguous rectangle of horizontal text on a page, one does not move from a line l to the next until one is finished reading all the text on line l. Within line l, one may in fact bounce around reading in alternating directions; however, no text is read more than once.
3. Within any such horizontal line, any permutation of the characters in the line will have exactly the same length.[3]
4. Within any contiguous rectangle of top-to-bottom Far East language text

[2] The origin of the name 'bditroff' is that to get 'ditroff' written from top to bottom in the unenhanced ditroff, one says `\b'ditroff'`; this utterance appears to the shell as 'bditroff'.

[3] For the length to be totally independent of the order, it is required that any kerning algorithm have the current font's direction as a parameter in order to know which pairs of letters must be kerned. If the algorithm is table-driven, then the kerning distance of the pair (X,Y) must be adjusted to look good when Y is printed to the left of X.

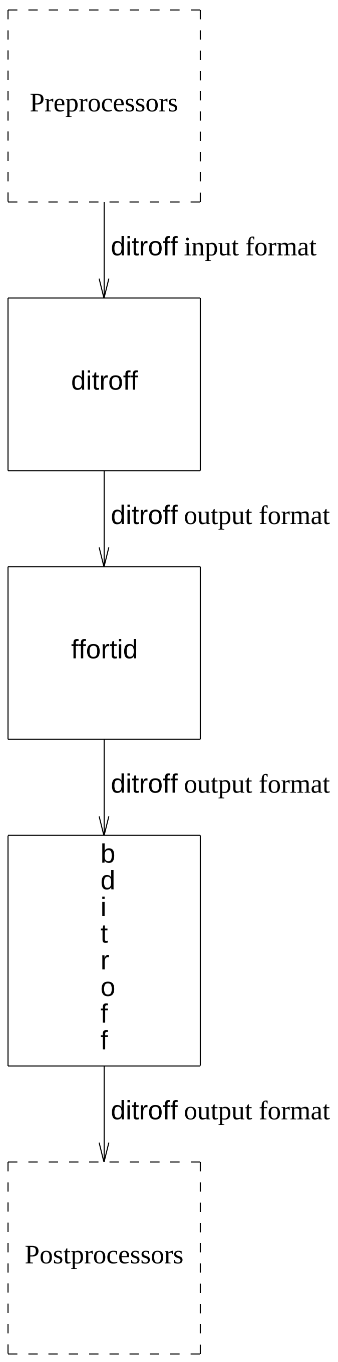

Figure 1. Data flow of triroff.

within a page, any permutation of the text within the rectangle will exactly fill the rectangle. For the purpose of this statement, trailing blanks are considered to be characters. This invariant works because the widths and the heights of all Far East language characters are the same.

Observe that application of ffortid to the output of ditroff should not affect the holding of the invariants. ffortid merely permutes the characters of horizontal lines; these lines remain horizontal. In fact, whether or not ffortid has been applied is irrelevant to bditroff because it leaves horizontal lines intact.

The description of the algorithm in the next subsection appeals to these invariants to demonstrate that the algorithm does work.

THE bditroff ALGORITHM AND ITS USE

The following discussion assumes the following input, in which lower case letters represent characters of a variable-width font to be printed from left to right and in which upper case letters represent characters of a constant width font to be printed top-to-bottom. Note that the algorithm can be used to print any text from top to bottom so long as the text is composed of characters that have the same constant width, either naturally or via the `.cs` command. Consider the input:

```
.ft R
\s9ada is a trademark of the u.s. dept. of defense.
ms-dos is a trademark of microsoft, inc.\s(10
.br
\!x TS
.ft C
A B C D E F
G H I J K L
M N O P Q R
S T U V
W X Y Z
.br
\!x TE
.ft R
\s9ffortid is a trademark of berry computer scientists, ltd.
unix is a trademark of at&t bell laboratories.\s(10
```

Note the `.br` commands before and after what is considered top-to-bottom text. Note also the transparent outputs, `\!x TS` and `\!x TE`, signalling the start and end of what is to be printed from top to bottom. Assuming a line length that allows 5 constant width characters and intervening blanks per line, and ignoring page breaks that might occur in the midst of the example, the output of ditroff follows, shown schematically, i.e., after passing it through the device driver:

```
ada is a trade-
mark   of   the
u.s.  dept.  of
defense.    ms-
```

```
dos is a trade-
mark         of
microsoft, inc.
A B C D E
F G H I J
K L M N O
P Q R S T
U V W X Y
Z
ffortid  is   a
trademark    of
berry      com-
puter    scien-
tists, ltd. unix
is a trademark
of  at&t   bell
laboratories.
```

The rectangular region to be reorganized by bditroff is the 5×6 region containing the typewriter font characters. This region's last four characters are blanks of exactly the same size as the letters. Suppose that all of the text fits on one page. Then, bditroff reads the characters in the region in a left-to-right, top-to-bottom sweep, as A, B, C, · · · blank, blank, blank, blank. It then lays them out in a top-to-bottom, right-to-left sweep in the same order to fill the same region. After this reorganization, the text is, schematically:

```
ada is a trade-
mark  of   the
u.s.  dept.  of
defense.    ms-
dos is a trade-
mark         of
microsoft, inc.
Y S M G A
Z T N H B
  U O I C
  V P J D
  W Q K E
  X R L F
ffortid  is   a
trademark    of
berry      com-
puter    scien-
tists, ltd. unix
is a trademark
of  at&t   bell
laboratories.
```

The reader should note that the algorithm is being applied by the formatting software on these examples. If however, the page break were to come, relative to the original ditroff output, after the third line of the constant width text, i.e., between the O and the P, the output would be, schematically:

```
ada is a trade-
mark  of   the
```

```
u.s.  dept.  of
defense.    ms-
dos is a trade-
mark         of
microsoft, inc.
M J G D A
N K H E B
O L I F C

_______________

  Y V S P
  Z W T Q
    X U R
ffortid  is   a
trademark    of
berry      com-
puter    scien-
tists, ltd.  unix
is  a trademark
of   at&t  bell
laboratories.
```

Here, the horizontal rule represents the page boundary.

If one is using a ditroff macro package in which page headers and footers are generated, even just page numbering, then additional measures must be taken lest the header and footer be included in the regions that are to be rearranged into top-to-bottom printing. The macro that is invoked at the page bottom trap must issue the same commands that are used to end a top-to-bottom region *before* it emits any of the regular page footer text. Moreover, it must arrange that the very next invocation of the page header macro issue the same commands that are used to begin a top-to-bottom region *after* it has emitted any of the regular page header text. This arranging is done by setting a register to a value which is interrogated by the page header macro.

Consider any region in which bditroff has been asked to rearrange the text to be printed from top to bottom. The beginning of the region may have been requested explicitly by the user or it may be the top of a page. The end of the region may have been requested explicitly by the user or it may be the bottom of a page. In any case, the region cannot be larger than one page. Any such region is a rectangle bounded by

1. the beginning,
2. the end,
3. the left margin of the page, and
4. the right margin of the page.

The rearrangement algorithm makes a column of text as long as necessary to fill the region. All the extra blanks end up in the left most columns. If the user does not desire this sort of filling, then it is straightforward for the user to adjust the page offset, line length, page length, line spacing, etc. to obtain the desired physical appearance. The authors examined Japanese magazines and found that the spacing between successive characters in a column is about .1 times the character size, but that the space between columns is about 1.2 times the character size. To achieve this appearance with the algorithm, it suffices for ditroff to be told that the vertical spacing is 1.1 times the current point size (as opposed to the more usual 1.2 times), and that the spacewidth is 1.2 ems (as opposed to the more usual approximately .333 ems for variable width fonts or 1 em for fixed width fonts). Of course, it is necessary to reset these upon leaving a region of top-to-bottom text.

Observe that with this algorithm, text of length *n* residing within one line in ditroff, e.g.,

```
A B C D
```

will end up being in what appears to be a right-to-left order, e.g.,

```
D C B A
```

as the algorithm fills a 1×*n* region top-to-bottom, with the columns of length one being filled in from the right to the left. If it is desired to obtain an *n*×1 space, with all the text down one column, one must trick the formatter a bit; ditroff can be forced to format the text in a line length equal to the width of one character. Then the text gets printed correctly, top-down, without application of bditroff, and applying bditroff reorganizes the rectangle of width one character into itself. Thus, one can have the `A B C D` printed downward, in a right justified column, by giving the input:

```
.\" set the page indentation to the current line length
.\" minus the new line length defined below
.nr XX \n(.lu-\w'\fCA'u
.in +\n(XXu
.\" set line length to the indentation plus the width of a
.\" standard character
.nr XX +\w'\fCA'u
.ll \n(XXu
\!x TS \" additional \ if invoked from diverted text
.br
A B C D
.br
\!x TE \" additional \ if invoked from diverted text
.br
.in \" reset indentation to previous value
.ll \" reset line length to previous value
```

Doing so yields the output:

```
                                                                   A
                                                                   B
                                                                   C
                                                                   D
```

One can obtain a centered column by giving the input:

```
.\" set the indentation to half of (the current line length
.\" minus the new line length defined below)
.nr XX (\n(.lu-\w’\fCA’u)/2u
.in +\n(XXu
.\" set line length to indentation plus the width of a
.\" standard character
.nr XX +\w’\fCA’u
.ll \n(XXu
\!x TS \" additional \ if invoked from diverted text
.br
A B C D
\!x TE \" additional \ if invoked from diverted text
.br
.in \" reset indentation to previous value
.ll \" reset line length to previous value
```

The output obtained is:

```
A
B
C
D
```

In other words, the layout of the page is unchanged by the application of bditroff. Thus, the page prepared only by ditroff and a device driver can be used as a guide to the ultimate appearance of the page after application of bditroff.

Finally observe that the algorithm is page preserving. That is, the page on which a given occurrence of a character appears does not change, although the character’s location on that page might very well change. This fact means that the algorithm needs only to consider one page at a time, that the maximum storage required for the program is that to store one page, and that the output of the algorithm can be conveniently passed to any device driver which works page by page. That is, it can assume that once it has built the description of page *n* and has seen the beginning of the next, it may print page *n* with the assurance that no more information for page *n* can arrive later.[4] A POSTSCRIPT device driver behaves under this assumption, as the POSTSCRIPT language is a page description language.

REQUIREMENTS ON THE INPUT TO bditroff

In order for the algorithm described in the previous section to work it must be possible for the program

1. to determine the exact position of each character on the page, and
2. to independently determine the line and page boundaries in the input.

[4] A major source of problems with tbl is that it can violate this single pass page construction property. If a table with more lines than can fit on the current page also has vertical lines, which are normally drawn after finishing the last row of the table, these lines get drawn on the second page from the projection of the start of the table onto this page to the end of the table on this page.

The ditroff output consists of a preamble describing the device, followed by a sequence of page descriptions each beginning with a page command of the form p*n* signalling the beginning of page number *n*. The description of a page consists logically of a sequence of (position, character) pairs, each describing exactly where on the page to print a character. The actual form of the position information is as occasional absolute coordinates with intervening horizontal and vertical movements. Thus a program reading this output must keep a position state and follow the relative movements in order to calculate the exact position of each character. Embedded among these (position, character) pairs, and actually independent of them, are end-of-line markers, of the form n*b a*; the important thing here is the n; the *b* and the *a* give the amount of space before and after the line in the device's units. These markers are necessary and cannot be calculated from the movements. There is no guarantee that all large movements to the left with small movements downward are ends of lines. One finds such movements in equations, graphs, pictures, tables, etc.

Because there are no end-of-line markers in TEX's DVI output format, the system structure adopted in this paper cannot be applied to make a tri-directional version of TEX. Instead, one must make modifications to TEX either to have it do the reorganization or to have it emit end-of-line markers[18]. In either case, one cannot use the standard distributed TEX and one faces the problem of maintaining more than one version of the program.

ACTUAL PROGRAM

In the input, one must signal the beginning of the text to be printed vertically by use of the transparent output `\!x TS` and signal its ending by use of `\!x TE`. If the text to be printed vertically appears in a diversion, the signals must be preceded by one '\' per level of diversion. In addition, if the signals occur in macro definitions, each '\' must be doubled. These signals must be preceded by breaking commands such as `.br`. To assist the user in dealing with the top-to-bottom text, macros `.BT` and `.ET` are defined which do these activities and which also adjust the line and word spacing to produce nicely spaced columns. Their definitions are:

```
.de BT \" begin top-to-bottom processing
.\" The user is presumed to have properly set the .ps .ll
.\" and .in for desired printing
.br \" break
.\" Make sure that the appearance of mono-spacing is not
.\" destroyed by spreading characters to fill the line
.na
.\" signal beginning of vertical text
\\!x TS \" additional \\ if invoked from diverted text
.vs \\n(VV \"set vertical spacing
.ss \\n(VM*12 \" set space width to be the distance between
.\" columns
..
.nr VM 5
.\" should be 4 or 5 to get spacewidth 4 or 5 times normal
.nr VV \n(.s
.\" should be 1 or 1.1 times current .ps
.de ET
.br \"break
.\" signal ending of vertical text
\\!x TE \" additional \\ if invoked from diverted text
.ad \" go back to normal spreading of lines
.vs \" reset vertical spacing what it was
.ss 12 \" set space width back to normal
..
```

The `.BT` macro uses the values of the registers `VM` and `VV` to adjust the inter-word (horizontal) space and the inter-line (vertical) space to help make it clearer to the human eye that the text is to be read from top-to-bottom rather than horizontally. It is recommended that `VM` be set to 4 or 5 and that `VV` be set to 1 or 1.1 times the current point size.

To assist the user in in forcing the location of the columns, the macros `.RA`, `.BC`, and `.EC` are defined which force a given number of right adjusted columns, force a given number of centered columns, and reset normal page margins, respectively. Their definitions are

```
.de RA \"force right adjusted \$1 columns
.\" set the page offset to the current line length minus the
.\" new line length defined below
.nr XX \\n(.lu-(\\$1*\\w’\\f(a1\\(a1’u)
.nr XX -((\\$1-1)*\\w’\\f(a1\\ ’u)
.in +\\n(XXu
.\" set line length to (\$1 times the width of a standard
.\" character) plus ((\$1 minus 1) times the width of the
.\" inter-word space)
.nr XX +(\\$1*\\w’\\f(a1\\(a1’u)
.nr XX +((\\$1-1)*\\w’\\f(a1\\ ’u)
.ll \\n(XXu
..
.de EC \" end columns or centering
.in \" reset left margin to previous value
.ll \" reset right margin to previous value
..
.de CE \" centering \$1 columns
.\" set the page offset to half of (the current line length
.\" minus the new line length defined below)
.nr XX (\\n(.lu-(\\$1*\\w’\\f(a1\\(a1’u)
.nr XX -((\\$1-1)*\\w’\\f(a1\\ ’u))/2u
.in +\\n(XXu \" if invoked in diverted text, use \\!.po
.\" set line length to (\$1 times the width of a standard
.\" character) plus ((\$1 minus 1) times the width of the
.\" inter-word space)
.nr XX +(\\$1*\\w’\\f(a1\\(a1’u)
.nr XX +((\\$1-1)*\\w’\\f(a1\\ ’u)
.ll \\n(XXu
..
```

In order that `.RA` and `.CE` work properly with respect to the spacewidth established for the top-to-bottom text in the `.BT` macro, it is necessary that the `.RA` or `.CE` come after the `.BT`.

Obviously, these and other macro definitions given in this section must be modified to use different names for macros and registers if any of them conflict with those of the base macro set used, as indeed happened when these were used with the macro package supplied by the editors of this journal for preparing the camera-ready copy. Moreover, these macros assume that their invocations are not inside diverted text. If they are invoked in diverted text, the transparent output commands beginning with `\\!x` must be changed to begin with `\\\\!x`, i.e., an extra pair of '\'s must be added to delay the output until the surrounding text is printed.

In order to allow proper control of horizontal spacing in a horizontal printing of the Far East language fonts, the following special characters have been provided:

1. The interword space has been set at .125 em so that the proper horizontal inter-character spacing can be obtained just by making each Far East language character a word.
2. The blank character `\f(a1\(a1` in the upper left hand corner of the character matrix has the same 1 em width that all other characters have.
3. The `\|` character, which is normally 1/6 of an em space, has been set to have the width of a full character so it can be used to force a full character width without forcing a font switch to font a1, the font of the blank character and without forcing emission of a character; i.e, ditroff treats use of `\|` as a movement.
4. The `\^` character, which is normally 1/12 of an em space, has been set to have half the width of the `\|` character. It too is treated by ditroff as a movement with no character emission.
5. The `\&` character is still the zero width character.
6. Finally the `\(XX` character has been provided in the S font as printing nothing but having a width equal to that of the blank and all other characters. It is a true character, so ditroff emits a character. It is on the first special font, so it can be used regardless of the current font without having to request a new font.

EXAMPLES

This section shows a sample of input and of printing it both from left to right and from top to bottom. Unlike in[12], there is no need for cutting and pasting to get both of the outputs on the same page!

The text is a famous Chinese poem, composed around 700 A.D. by the most renowned poet in China, Li Bai. The input is:

```
\f(e0\(ae \f(c1\(b0 \f(cc\(c0 \f(b7\(ee \f(b8\(f7 \f(a1\(a4
\f(b5\(bf \f(c0\(a7 \f(c3\(cf \f(be\(e5 \f(c1\(fa \f(a1\(a3
.br
\f(da\(aa \f(c6\(ac \f(cb\(be \f(cc\(c0 \f(b7\(ee \f(a1\(a4
\f(c4\(e3 \f(c6\(ac \f(bb\(d7 \f(b8\(ce \f(b6\(bf \f(a1\(a3
```

Its output in left-to-right mode is:

牀 前 明 月 光 ， 疑 是 地 上 霜 。
舉 頭 望 明 月 ， 低 頭 思 故 鄉 。

Its output in top-to-bottom mode is:

牀前明月光，疑是地上霜。
舉頭望明月，低頭思故鄉。

This printing was done using the `.BT`, `.ET`, `.RA`, and `.EC` macros defined above.

WEAKNESSES

The enthusiasm of the authors notwithstanding, the tri-directional formatting system described herein has some weaknesses. Some are easily repaired, and others are not. The problems and possible solutions are presented one by one in this section.

Orientation of punctuation characters

Readers who know a Far East language will have noticed that the punctuation symbols in the examples are oriented incorrectly for top-to-bottom printing! They are oriented correctly for horizontal printing. Specifically, the stand-alone punctuation symbols, the period, the comma, etc., are in the lower left hand corner of their bounding boxes, and the bracketing punctuation symbols, the parentheses, the braces, etc., are oriented in their bounding boxes to wrap around the ends of enclosed *horizontal* text. A stand-alone symbol needs to be in the center of gravity of its bounding box, and a bracketing symbol needs to be rotated 90° counter-clockwise in its bounding box so that it can wrap around an end of *top-to-bottom* text.

Probably the simplest solution is to add the missing alternative forms to the character sets in unused positions. Then, the bditroff program can be modified so that when it is working with a region of top-to-bottom text, it simply replaces the codes for the horizontal versions of these characters by those for the alternative, vertical versions of the same characters.

Inclusion of proportional spaced Latin characters in top-to-bottom text

triroff supports inclusion of Latin language text among top-to-bottom text, but only in an unrotated-advancing-downward form using the constant width Latin characters found in row three of the JIS, the GB-2312, and the KS C 5601 character sets. It is also common these days, to rotate the Latin language text so that its natural right-to-left flow matches the top-to-bottom flow of the surrounding Far East language text. That is, the Latin text is printed in a variable width font sideways with its base line coinciding with the line running down the left edge of the vertical column containing the text. This printing is achieved by having a Far East language font with its letters rotated 90° counter-clockwise, and printing this Far East font together with the available Latin fonts from left to right. If such a page is then rotated 90° counter-clockwise, it appears to the reader that the Far East language characters are printed right side up top-to-bottom and the Latin letters are printed sideways. Probably, this style arose simply because it is so easy to implement with modern printing devices.

Figures 2 and 3, found in the appendix, show the Japanese and Chinese abstracts of the paper printed in that style. The reason these figures are in the appendix is to preserve the truth of the claim, made at the beginning of the paper, that the *entire* paper is printed as a single document with the software describe herein. The appendix figures cannot be printed in the same run of ditroff that prints the rest of the paper, *even* with a rotated Latin font, because the Latin font does not meet the constant width requirement for using bditroff. It is typeset as a separate document using the trick of the rotated Far East language font.

In order to be able to print the appendix in the same run of ditroff that prints the paper, it is necessary to have a ditroff device driver that can change from portrait to landscape mode and vice versa at any arbitrary point in the document. The particular device driver used to print this paper, psdit, from Adobe's TRANSCRIPT™ package, does not have this capability. There exists a device driver, namely Pipeline Associate's devps™ that has facilities for rotating arbitrary text at any angle. Thus, it should be possible to put the needed capability into any device driver.

In top-to-bottom Far East language text, a short multi-digit numeral in a Latin text font is occasionally printed as a unit unrotated, with its base line perpendicular to the vertical axis of the column that contains it. This works nicely when the numeral is short enough so that it does not stick out to far from the width of the column. This cannot be done in the current version of the software. However, there are a number of ways that this feature can be implemented as easy extension of the current software.

1. Add to the Far East language font all possible short multi-digit numerals as single characters. If the fonts are POSTSCRIPT fonts, then the added characters can, in fact, be POSTSCRIPT programs that build and show the numerals using digits from one of the available Latin fonts. Since the length of these numerals cannot be too much longer than the width of a normal Far East language character, the number of these is limited. Given the fonts used in this paper, it appears that the maximum length of such numerals is two digits; thus only 99 numeral characters would have to be added to the Far East font.
2. Use the above mentioned facility, such as available in devps, that can rotate any text any angle to build a macro that rotates its argument about its center and fools ditroff into believing that the size of the argument is the same as that

of the normal Far East font character.

As one of the referees pointed out, in the Xinjinang Uighur region of the PRC, there is another language spoken which is not written in any of the directions covered so far in this paper. The language is Mongolian, and it is written from top to bottom on lines that flow from *left to right*! Even if a font were available for the language, the current version of the software cannot handle its printing direction. However, it would not be difficult to make printing in the Mongolian direction another direction supported by the bditroff program. It would mean making the direction in which the reconstructed columns flow determined by a variable. The variable would be set to mean 'right-to-left' when the result of saying '`\!x TB`' is found and would be set to mean 'left-to-right' when the result of saying '`\!x TM`' is found. The result of saying '`\!x TE`' can be used to end the top-to-bottom region for either.

CONCLUSIONS

As can be seen from this paper, triroff works mostly as desired. That is, the three programs ditroff, ffortid, and bditroff combine to produce an effective tri-directional formatter that accepts any input accepted by ditroff, including that produced by any of ditroff's preprocessors, works with any set of ditroff macros, and generates output indistinguishable from ditroff's. This output is then acceptable to any ditroff device driver. The use of this software to typeset this paper is a demonstration of this claim.

The main strength of the triroff approach is its modularity. This modularity allows each new direction of printing to be attacked as a separate problem uncluttered by concerns with other directions and other formatting problems. This modularity allows the use of an *unmodified* ditroff, which in turn allows the use of all of ditroff's preprocessors and macro packages.

There are a number of minor problems both in appearance and in function. However, their solutions are straightforward because of this modularity. For example, changing the orientation of the punctuation symbols and moving them in the bounding box involves no change to any of the programs composing triroff; it requires only the use of a different Far East language font containing the reoriented and repositioned punctuation symbols as added characters. The solution to these problems are left for future work.

Of course, the ultimate judge of the quality of the software is the user. Accordingly, the bditroff software described herein is available from the second author for a nominal fee and under a non-disclosure agreement.

ACKNOWLEDGEMENTS

The Japanese and Chinese translations of the abstract were provided by Taiichi Yuasa (湯 浅 太 一) and Kam Pui Chow (鄒 錦 沛) respectively. Low Hwee Boon (劉 斐 文) provided samples of Hong Kong and Singaporean magazines and newspapers.

The authors thank one particular enthusiastic, but critical, anonymous referee, whose hard questions resulted in a greatly improved paper.

`devps` is a trademark of Pipeline Associates, Inc. DWB is a trademark of AT&T Bell Laboratories. ffortid is a trademark of Berry Computer Scientists. Linotronic is a trademark of Linotronic, Inc. Macintosh is a trademark of Apple Computers, Inc. MS-

DOS is a trademark of Microsoft, Inc. POSTSRIPT is a trademark of Adobe Computer Systems. TEX is a trademark of the American Mathematical Society. TRANSCRIPT is a trademark of Adobe Computer Systems. UNIX is a trademark of AT&T Bell Laboratories.

REFERENCES

1. K.K. Abe and D.M. Berry, ‘indx and findphrases, A System for Generating Indexes for Ditroff Documents’, *Software—Practice and Experience*, **19** (1), 1-34 (1989).
2. C. Buchman, D.M. Berry, and J. Gonczarowski, ‘*DITROFF/FFORTID*, An Adaptation of the UNIX *DITROFF* for Formatting Bi-Directional Text’, *ACM Transactions on Office Information Systems*, **3** (4), (1985).
3. N. Batchelder and T. Darrell, ‘Psfig — A DITROFF Preprocessor for POSTSCRIPT Figures’, Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA 19104.
4. J.D. Becker, ‘Multilingual Word Processing’, *Scientific American*, **251** (1), 96-107 (1984).
5. J.D. Becker, ‘Arabic Word Processing’, *Communications of the ACM*, **30** (7), 600-611 (1987).
6. J.L. Bentley and B.W. Kernighan, ‘GRAP — A Language for Typesetting Graphs, Tutorial and User Manual’, Computing Science Technical Report No. 114, AT&T Bell Laboratories, Murray Hill, NJ 07974 (December, 1984).
7. S. Carson and D.M. Berry, ‘Alg* — Filters for Typesetting Algorithms’, News, Usenet (1985).
8. K.P. Chow and C.T. Hung, ‘Chinese Workbench: An Interactive Environment for Chinese Writers’, Technical Report TR-87-07, Centre of Computer Studies and Applications, University of Hong Kong (June, 1987).
9. *EgWord Version 2.2 English Reference Manual*, Ergosoft Corp., Tokyo, Japan, October 1986.
10. *FeiMa-II, Chinese Word Processor, English Reference Manual (Version 3.0)*, Wu Corp., Avon, Connecticut, August, 1986.
11. 長谷部紀元 and 亀山豊久, ‘和欧混合文書組み版システムの試作と組み版規則の検討’, in 日本語文書処理研究会資料, pp. 1-10, (1985).
12. C.H. Ip, D.M. Berry, and K.P. Chow, ‘CWPR, a Chinese/Japanese Word-Processing System for Use with UNIX Device-Independent TROFF’, in *Proceedings of Second International Conference on Computers and Applications*, Beijing, PRC, (June, 1987).
13. R. Kasbarian, ‘The Language of Choice’, *International UNIX* 41-46 (1989).
14. B.W. Kernighan and L.L. Cherry, *Typesetting Mathematics — User’s Guide (Second Edition)*, Bell Laboratories, Murray Hill, NJ 07974, 1978.
15. B.W. Kernighan, ‘PIC — A Graphics Language for Typesetting, User Manual’, Computing Science Technical Report No. 85, Bell Laboratories (March, 1982).
16. B.W. Kernighan, ‘A Typesetter-independent TROFF’, Computing Science Technical Report No. 97, Bell Laboratories (March, 1982).
17. 亀山豊久 and 長谷部紀元, ‘和欧混合文書組み版システムJTROFFの開発’, in 情報処理学会第31回(昭和60年後期)全国大会.
18. D.E. Knuth and P. MacKay, ‘Mixing Right-to-left Texts with Left-to-right Texts’, *TUGboat*, **8** (1), (1987).
19. D.E. Knuth, *The* TEX*book*, Addison-Wesley, Reading, MA, 1984.
20. M.E. Lesk, ‘Some Applications of Inverted Indexes on the UNIX System’, Computing Science Technical Report No. 69, Bell Laboratories (June 21, 1978).
21. M.E. Lesk, *TBL — A Program to Format Tables*, Bell Laboratories, Murray Hill, NJ 07974, 1978.
22. 長島正明 and 川端洋一, ‘TEX の日本語出力’, 第2回 TEXユーザ会 配布資料, CANON株式会車 (1986年 7月 17日).
23. J.F. Ossana, *NROFF/TROFF User’s Manual*, Bell Laboratories, Murray Hill, NJ 07974, October 11, 1976.
24. Y. Saito, ‘Report on JTEX: a Japanese TEX’, *TUGboat* (2), (1987).

25. H. Trickey, ‘DRAG — A Graph Drawing System’, in *Electronic Publishing ’88*, ed. J. André and H. van Vliet, Cambridge University Press, Cambridge, UK, (1988).
26. ‘The UNIX Programmer’s Manual’, Technical Report, Bell Telephone Laboratories, Murray Hill, NJ 07974 (June, 1981).
27. *International UNIX*, Supplement to UNIX World, 1989.
28. Z. Wu, W. Islam, J. Jin, S. Janbolatov, and J. Song, ‘A Multi-Language Characters Operating System on IBM PC/XT Microcomputer’, in *Proceedings of Second International Conference on Computers and Applications*, Beijing, PRC, (June, 1987).
29. T. Wolfman, ‘flo — A Language for Typesetting Flowcharts’, M.Sc. Thesis, Technion, Haifa, Israel (1989).
30. C. J. Van Wyk, ‘IDEAL User’s Manual’, Computing Science Technical Report No. 103, Bell Laboratories (December 17, 1981).

APPENDIX

This appendix contains the Japanese and Chinese abstracts printed in the additional form mentioned in the section on weaknesses, i.e., with variable width Latin language text printed sideways so that its left-to-right flow matches the top-to-bottom flow of the containing Far East language text. Due to the limitations mentioned in that section, these abstracts could not be typeset as part of the document that ends with the end of this paragraph. Instead they had to be typeset as a single other document and pasted in.

Figure 2. Japanese Abstract.

Figure 3. Chinese Abstract.