DITROFF/FFORTID, An Adaptation of the UNIX™ *DITROFF* for Formatting Bidirectional Text

CARY BUCHMAN and DANIEL M. BERRY
University of California, Los Angeles
JAKOB GONCZAROWSKI
Hebrew University

DITROFF/FFORTID, a collection of pre- and postprocessors for the UNIX *DITROFF* (Device Independent Typesetter RunOFF) is described. *DITROFF/FFORTID* permits formatting of text involving a mixture of languages written from left to right and from right to left, such as English and Hebrew. The programs are table driven or macro-generated to permit them to be used for any languages written from left to right and from right to left so long as fonts with the proper character sets can be mounted on a typesetting device supported by *DITROFF*. The preprocessors are set up to permit phonetic, unidirectional input of all of the alphabets needed using only the two alphabets (each case counts as an alphabet) available on the input device. These macro-generated preprocessors can be adjusted to the user's pronunciation, the language's rules about a letter's form, depending on its position in the word, and the language of the user's input keyboard. The postprocessor is set up to properly change direction of formatting when the text switches to a language written in a different direction. The collection of programs is also designed to allow use of any of *DITROFF*'s preprocessors, such as *PIC*, *EQN*, *TBL*, and the various device drivers.

Categories and Subject Descriptors: H.4.1 [**Information System Applications**]: Office Automation—*word processing*; I.7.2 [**Text Processing**]: Document Preparation—*format and notation*

General Terms: Design, Human Factors, Languages

Additional Key Words and Phrases: Bidirectional text, bidirectional formatting, macro, preprocessor, postprocessor

1. INTRODUCTION

DITROFF/FFORTID [2] is a collection of programs developed at the University of California, Los Angeles, for augmenting the UNIX [13] *DITROFF* (Device Independent Typesetter RunOFF) [4] so that it can be used to format text involving languages written left to right, right to left, and in both directions. It

™ UNIX is a trademark of AT&T Bell Laboratories.

Authors' addresses: C. Buchman, Deloitte Haskins & Sells, Crocker Center, 333 So. Grand Ave., Los Angeles, CA 90071; D. M. Berry, Department of Computer Science, University of California, Los Angeles, CA 90024; J. Gonczarowski, Department of Computer Science, Hebrew University, Jerusalem, Israel.

is intended to be another of the many macro, pre-, and postprocessors available for *DITROFF* to permit a variety of useful text generation functions such as dealing with bibliographies, drawing pictures, building tables, writing equations, typesetting on a variety of devices, etc. The centered, in-line examples and the appendixes were formatted and typeset on an Imagen[1] laser printer with the help of the *DITROFF/FFORTID* system.[2]

2. GOALS AND DESIGN

Assume that one wishes to write a document involving text in two languages. One of these languages is a left-to-right language called, for the purpose of keeping the discussion language independent, LR. LR is typically written with the Latin alphabet. The other is a right-to-left language called RL. The two languages also involve different alphabets. First consider what is available for input and output.

2.1 Available Input and Output Devices

Typically, the input device available is a standard ASCII keyboard with two cases of Latin letters designed for use with language LR. There also exist keyboards with the alphabet of the language RL in all of its forms, with possibly one case of Latin available also. In each of these cases, the associated screen can usually exhibit only what is capable of being input. There do exist terminals that can switch between the above two systems; however, at any one time only one alphabet or the other, but not both, can appear on the screen. Currently, with the advent of high resolution graphic devices, there exist terminals that can input both LR and RL in all cases and forms, using the standard keyboard arrangement for each, and can exhibit all letters of each on the screen at the same time [1]. Function keys are used to switch from one language to another; upon a switch, the key-to-byte code mapping is changed. The language under which a character is entered is remembered so that all characters can be exhibited on the screen in their correct glyph. Ideally then, the formatting system should be able to accommodate input from any kind of keyboard.

The output device may be anything from a lineprinter to a high-resolution phototypesetter. The device may have a programmable character set or it may have fixed character sets. In any case, it is assumed that whatever the output device, the alphabets of LR and RL in all of their respective cases may be mounted.

The desire is to be able to prepare input on the input device, giving the text interspersed with formatting commands, and produce nicely formatted output on the output device, with each language written in its own alphabet and direction.

[1] Imagen is a trademark of Imagen, Inc.

[2] The manuscript that the referees saw was typeset on the Imagen laser printer in as close an approximation of *TOOIS* format as possible, in the hopes that this copy would be accepted camera-ready for direct inclusion in the journal. The low, 240 dots-per-inch (dpi) resolution of the device made that copy unacceptable for direct inclusion in the journal. Furthermore, no typesetter of sufficiently high resolution, available to the authors, has a full Hebrew alphabet. Hence it was decided to have the article typeset by the journal's normal printer and to use the manuscript copy directly only for those portions that actually make use of the bidirectional capabilities, that is, the in-line examples and the appendixes.

2.2 Software Engineering, Human Interface, and Program Design Considerations

There are a number of software engineering, human interface, and program design considerations. It is necessary to consider existing formatting systems with an eye to reusing existing software, if possible, and to consider the preferred way of entering the text that is to be formatted.

It is highly desirable that the formatting commands be similar to or even identical to those of an existing formatting system. If so, then

(1) at least the users of that system do not have much new to learn,
(2) new users have a higher payoff when learning to use the bidirectional formatting system because what they learn is useful for ordinary formatting,
(3) there exists a large environment consisting of prepasses, macros, databases, experience, expertise, folklore, etc., that makes it easier for both the new and not so new user of the bidirectional formatter,
(4) it may be possible to use the software for the existing formatting system as at least the basis for the bidirectional formatter.

Of course, the bidirectional formatter has new capabilities for which new commands must be provided. These will require new learning and new software in any case. These must be designed carefully to mesh well with the features and commands of the existing system.

The existing system chosen for extensions is the *Device Independent TROFF system that can be obtained from* AT&T *for running on* UNIX systems. This is a powerful, stable, existing system with many macro sets for making it appear higher level and many existing preprocessors for dealing with bibliographical citations, pictures, tables, equations, etc. There is a group of experts on this system available for consultation on the USENET UNIX network. Most UNIX sites have resident experts. Most important from the point of view of the designers of the bidirectional formatter, it is available in source form so it is easy to modify. Also the UNIX system, by its very design, encourages composing existing software with new small programs to obtain programs with more function; the UNIX pipe and file redirection concepts are indispensable for this purpose [7].

For what kind of user should the input scheme be designed? It seems appropriate to design the input scheme for a person who is a touch-typist in both languages.[3] Such a person would type all the characters of a document in the order in which they are heard, hitting a function key and/or dropping in a language or font changing command when the language is changed. Thus, for such a person, all input would be in *time order* and each language would be in its own ASCII code.[4] For devices which do not support direct input of all languages in their own ASCII codes, it is easy enough to provide translators which map from, say, a phonetic encoding of the missing language to its ASCII coding. Then no matter what, all input can be in time order. Note that if the input is in time order, then the characters of the input are stored in the file in time order also.

[3] The second author has taught himself to touch-type in Hebrew and English. So, such people exist. Also many secretaries in Israel touch-type in both.

[4] The ASCII encoding for any alphabet should assign codes to letters in lexicographical order, so that sorting into alphabetical order can be done by sorting the codes numerically.

The next question to consider is that of printing each language in its own direction. For the purposes of this discussion let LR be English in the Roman font, called R, and let RL be English in the Typewriter font, called T, and written right to left.[5] The input to the formatter would be[6]

```
\fRThe next sentence contains one verb.
\fTThis sentence contains one verb.
\fRThe previous sentence contains one verb.
```

Note that all input is in the order in which it is heard, although in this representation, the second sentence, which is in RL, is printed backward with respect to the way it is supposed to be printed. This yields the output:

```
The next sentence contains one
verb.      This      sentence
contains   one   verb.   The
previous sentence contains one
verb.
```

Note that in this output the Typewriter text, in language RL, is written backward, with respect to the way it is supposed to appear.

Now the problem is to get RL to be printed right to left. Reversing the input,

```
\fRThe next sentence contains one verb.
\fT.brev eno sniatnoc ecnetnes sihT
\fRThe previous sentence contains one verb.
```

does not help. In the output,

```
The next sentence contains one
verb.   .brev  eno  sniatnoc
ecnetnes  sihT The previous
sentence contains one verb.
```

The second sentence is split on the lines incorrectly causing the beginning of the sentence to be on the line *after* that containing the end of the sentence.

The solution proposed by the third author, in an earlier prototype[7] [2] of the present system, is to format the text in its input form, that is, with all text in time order, to discover where the line breaks are, and *then* to reverse the RL text within each line. This scheme works because of the simple fact that the decision of where to break lines depends on the sums of the widths of the letters on the line and *not on the direction in which these letters are written*; a character's width

[5] As might have been written by Leonardo da Vinci had he known English!

[6] **\f** is the in-line command to switch fonts. **\fR** means "switch to Roman font." **\fT** means "switch to Typewritter font."

[7] The present system is an improvement over the earlier prototype in that it is built on top of *DITROFF*, which is better structured than *TROFF* [12] upon which the prototype runs. In particular, the prototype requires the use of a slightly modified *TROFF*, while the new programs run with a standard version of *DITROFF*. Additionally, as a result of having to use the original *TROFF* with its various restrictions, the prototype imposes a number of restrictions on overall document format; the new programs eliminate almost all the restrictions and give the user greater formatting flexibility.

does not change when the direction of printing changes. With such a scheme, a slight variation of the first input above,[8]

```
\fRThe next sentence contains one verb.
\fTThis sentence contains one verb\fR.
\fRThe previous sentence contains one verb.
```

yields the following output:

```
The next sentence contains one
verb.     ecnetnes     sihT
brev  eno  sniatnoc.  The
previous sentence contains one
verb.
```

This scheme is general enough to be applied to any system in which a representation of the text after breaking into lines, but before printing, exists. In the *DITROFF* system, the output of the device-independent part of the system, actually the *DITROFF* program itself, is of the desired form. This output is sent to a device driver that interprets the output in order to print the text of each line. The scheme then requires that a new program be written to sit between *DITROFF* and the device driver, reorganizing the *DITROFF* output so that the RL text is printed by the device driver in the right-to-left direction. This program is called *FFORTID.* This bidirectional capability can easily be added also to the TEX [8] system by writing a program that reorganizes the device-independent *DVI* form output in a similar way.[9]

Evidently, this scheme is not obvious at first sight. The evidence of this is a suggestion in Knuth's TEX book [8, p. 66] that one way to handle right-to-left languages is to have characters with negative widths [8]. This information is used in order to determine which text goes on what line and how far apart to spread the words on a line by the stage causing the generation of the *DVI* output. In fairness to Knuth, it must be noted that the suggestion also carries the observation that using negative widths works only to a limited extent, since the line-breaking algorithm is based on the assumption that words do not have negative widths. The present authors' observation is that trying to deal with direction of printing during the line-breaking stage is hopelessly complicated and that dealing with it after the lines have been broken is the simplest.

2.3 System Flow and Module Function

Thus the bidirectional formatting system, called *DITROFF/FFORTID,* consists of a variety of input translators called **TRN* plus a program called *FFORTID.* The schematic for using the pieces of *DITROFF/FFORTID* with *DITROFF* is shown in Figure 1 [5, 6, 9, 10, 15].

The in-line examples, the flow diagram, and the appendixes of the present paper were done with the help of *REFER, PIC, TBL, EQN, DITROFF, FFORTID,* and a device driver. In what follows, the reader is assumed to be familiar with

[8] It is necessary to switch to the R font before the period, because the RL sentence is a phrase within a sentence in a left-to-right language document. Section 4 deals with documents in a right-to-left language.

[9] A group at the Weizmann Institute and Pierre MacKay are building such programs.

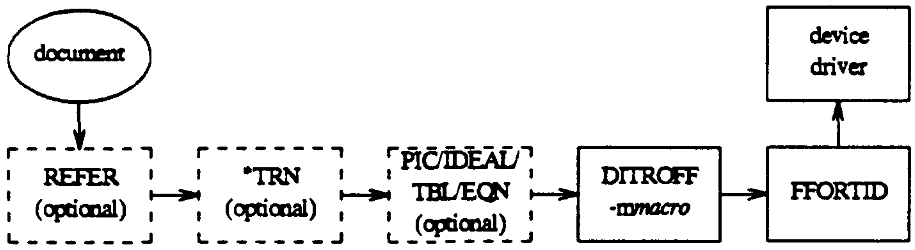

Figure 1

the commands of *TBL*, *EQN*, and *DITROFF*. The discussion herein makes use of such knowledge with little or no further explanation.

Each **TRN*, named with a different expansion of "*", is capable of translating input involving only the characters available on some keyboard into a fuller set involving all needed characters in all alphabets. The use of a **TRN* is optional because one may just happen to have a keyboard with all needed characters. If one's keyboard completely lacks a particular alphabet, then one may wish to phonetically encode the letters of the missing alphabet with letters of an available alphabet. The **TRN* in this case translates the coding letter into a suitable internal representation of the coded letter. The most suitable internal code for any alphabet is that used by its own standard keyboard, that is, its ASCII encoding. Because there is, in general, no unique transliteration for a language into the alphabet of another language, each user may prefer a different transliteration scheme based on his or her dialect of that language. In addition, each output device may have a different mapping between internal character codes and letters of the output alphabet. Each transliteration **TRN* is constructed from a single *TRN* scheme in order to support a particular transliteration and a particular device. Thus, either the transliteration or the device's encoding can be changed without changing the remaining programs.

Another possibility is that one case, but not both, of the letters for an alphabet is available. Then one wants the ability to let the available case represent the most common case for the language and to be able to use some escape character to denote a case shift for the following character. For such a translation, the **TRN* takes the escape character as a parameter.

As a consequence, whether or not the input must be subjected to some **TRN*, by the time *DITROFF* gets the document, the entire text in both languages is in a form such that if it is sent to the output device with the proper fonts mounted, the correct letters are printed, but not in the proper direction.

DITROFF takes the command-laden input, makes use of tables telling it how wide each character of each alphabet is, and breaks the text into properly adjusted lines with the commands obeyed. These commands include those for indenting paragraphs, changing fonts, centering lines, breaking the text into lines and pages, numbering pages, printing footnotes, printing figures, printing equations, printing tables, drawing pictures, making tables of contents, etc. If this output were to be sent directly to the device driver, each line would contain all the characters it should, but the characters in RL would be in the reverse direction on a line-by-line basis.

FFORTID takes *DITROFF* output and produces output that appears to the subsequent device driver as if it had come from *DITROFF* itself. *FFORTID* searches for right-to-left text and properly rearranges the output so that when the output is sent to a device with the proper alphabets mounted, the output appears as it should with each language printed in its own direction.

3. **TRN*

There are two kinds of **TRN*s available: transliterating, and missing case. The transliterating **TRN* provides a missing alphabet by building an encoding for its letters using letters of an available alphabet. The missing-case **TRN* provides a missing case of a present alphabet by using an escape character to denote a case shift in the following letter. Due to space limitations, these **TRN*s are not discussed further here. See [2] for more details.

3.1 Transliterating **TRN*s

Each transliterating *TRN is a generically generated, table-driven program which defines the mapping between the characters of the input device and the character codes for the output device. The mapping is built out of two table-defined mappings. One maps characters of the input device to characters of the language's alphabet. The second maps characters of the language's alphabet to the codes causing the printing of the characters' glyphs on the output device. Since each mapping is table driven, each can easily be varied to produce another version of **TRN*. The first mapping can be constructed to represent a phonetic encoding based on the user's own pronunciation of the language or can be constructed to represent the standard typewriter keyboard for the language for the convenience of a user who can touch-type. The second mapping can be constructed to fit the output device available. For example, the codes for the Hebrew letters on a Diablo[10] daisy-wheel printer are different from those of the Berkeley Versatec[11] fonts [14], so a different output would be needed for each. Use of these tables permits the translator and the other programs in the system to be input and output device independent.

It is recognized that most of the use of these programs will be for right-to-left Middle Eastern languages, such as Arabic, Hebrew, and Farsi. These languages have some letters whose forms change depending on their positions in the containing words. Therefore, the table for the first mapping is set up so that the translation is also dependent on a character's position within a word. Thus, there are indications in this table of which letters change form depending on their position within a word. Additionally, there is a third table giving the set of characters that serve to delimit a word. The end-of-word indicators in this table can be both single ASCII chracters and *DITROFF* escape sequences.[12]

Finally, in these languages, vowels tend not to be printed. However, vowels are useful for increasing the readability of phonetic input. Thus, the tables also allow input characters to be translated into no or zero-width characters on the output.

[10] Diablo is a trademark of Xerox Corporation.

[11] Versatec is a trademark of Xerox Corporation.

[12] This includes only *DITROFF* escape sequence characters of the form \X or \\(*XY*, for any *X* and *Y*.

Any input character not included in the first table is translated into no character and does not appear in the output. On the other hand, an input character can be translated to a zero-width character by mapping it to an appropriate output code in the second table. It should be noted that an input character not included in the first mapping table, thus translating into no character, can still be included as an end-of-word indicator in the third table. In this manner, the character can serve as a word delimiter without appearing in the output. This capability is useful for forcing a word-ending form of a letter even when it is not followed by an end-of-word indicator.

Because the transliterating **TRN*s are generally used to represent the letters of a right-to-left alphabet by letters of a left-to-right alphabet, it must be possible for such **TRN*s to exchange the common bracketing pairs. For example, an open parenthesis might be represented by "(" in the input and might be represented as ")" in the output.

Appendix A contains a user's description of *ISRATRN*, a **TRN* for Hebrew assuming an Israeli pronunciation of the letters. Other **TRN*s include one based on a Yiddish pronunciation and one designed for a Hebrew touch-typist; this last **TRN* assumes that a Latin letter maps to a particular Hebrew letter if, in the standard touch-typing keyboards of both languages, they occupy the same key. All three of these assume the same output device, one in which the representation of the standard letters and punctuation is ASCII.

It was considered to let the transliterating **TRN* programs turn translation on and off on the basis of font changes. That is, whenever the text is in a font for the language of RL, the translation is turned on. However, because of the various preprocessors and macro packages [11], it may not be possible to discern a font change from the text prepared by the user. For example, in most macro packages, the "new section" macro causes its argument, the title of the section, to be printed in another font. Thus either the translator would have to know about all possible macros and preprocessor commands or else it would have to be applied just before entering *DITROFF* when all macro and preprocessor commands have been converted to *DITROFF* commands. However, besides being very inflexible, this approach complicates the process of identifying word boundaries. The macro processors and preprocessors tend to build up long words out of **.ds** and **.as** commands. It would then be impossible to find the end of a word in the human sense of the word. Thus it was decided to make the **TRN* programs have their own control characters that turn translation on and off almost independently of the text.

Each transliterating **TRN* has an argument, **−e**, for setting its escape character in order that more than one language be processible in a document. The default escape character is **%**. In what follows, **%** is used to stand for whatever is in fact the escape character. The program distinguishes two kinds of environments. One kind is the single, global textual environment, and the other kind consists of individual, local, in-command environments occurring within each command line and each *DITROFF* escape sequence. In any environment, **%N** turns translation on, **%F** turns translation off, and **%%** stands for **%** itself. Entering a command line, recognized by a line beginning with a "." or "'", or entering a *DITROFF* escape sequence, recognized by an occurrence of "\" anywhere, temporarily suspends whatever global translation may be going on. Ending the command or escape sequence resumes whatever global translation was suspended. Within each command line or escape sequence, embedded text may be translated by issuing a **%N** that remains in effect until the next **%F** or the end of the command line or escape sequence, whichever comes first. In this manner, a **TRN* processes the input on a strict character-by-character basis, recognizing and preserving *DITROFF* macro package commands and *DITROFF* escape sequences, and applying the translation mapping to all other input. It is therefore the user's responsibility to turn the translation off before any *DITROFF* preprocessor command that does not look like a *DITROFF* command or escape sequence. Examples of such commands are *TBL*'s table layout commands.

Within either kind of **TRN* translation environment, determining if an input character is at the beginning or in the middle of a word depends only on the occurrence of textual and word delimiting characters within the environment. Thus, if an end-of-word indicator within the global text environment is followed by *DITROFF* command lines or escape sequences, which constitute local **TRN* environments, the next text character is still considered to be at the beginning of a word. Likewise, even if a non-end-of-word text character is followed by *DITROFF* command lines or escape sequences, the next text character is still considered to be in the middle of a word.

When determining if an input character is at the end of a word, **TRN* does only a one-character look-ahead. If the next input character, whether it is a text character or even part of a *DITROFF* escape sequence construct, is one of those found in the end-of-word indicator table, the character is considered to be at the end of a word. Thus, even if a character is followed by a nonbreaking *DITROFF* escape sequence, which is in turn followed by one of the characters in the word delimiter table, the character is not considered to be at the end of a word. **TRN*'s one-character look-ahead also limits the use of escape sequence character as end-of-word indicators. If an escape sequence is included in the word delimiter table, any text character following it would be considered the beginning of a word. However, **TRN*'s one-character look-ahead would recognize and evaluate the "\" part of the escape sequence construct only when processing any preceding text character. In most cases, these limitations do not restrict **TRN*'s usefulness. In the first case, the escape sequence can usually follow the end-of-word indicator and still yield the same formatting results. In the second case, the escape sequence character can be preceded by a single character that is included in the word delimiter table but is mapped to no character (i.e., it is not included in the first mapping table). In this manner, the escape sequence character would indicate that the following text character is at the beginning of a word, while the extra single character would indicate that the preceding character is at the end of the word. On the other hand, the fact that a character is at the end of a word can easily be concealed by following it with a **\&**, the *DITROFF* zero-width character.

**TRN* is invoked by a command line of the form:

TRN** [**−e*TRN escape character*] [**−h***hyphenation-on-argument*]
[**−l***ligature-mode-on-argument*]
[*input file list*]

As explained above, the **−e** argument is used to set **TRN*'s escape character to something other than the default **%**.

As was mentioned before, **TRN*'s translation capabilities are generally used for the non-English portions in a document. In most cases, then, when starting a portion of text to be translated by **TRN*, it is necessary to turn off *DITROFF*'s ligature and hyphenation mechanisms [11]. Similarly, it may be desired to turn them back on at the end of these portions of text. Therefore, as a convenience for the user, if the **−h** and or **−l** arguments are specified, **TRN* automatically turns the appropriate *DITROFF* mechanism *off* and *on* when it encounters the **%N** and **%F** control characters, respectively. Unless otherwise specified in the command line, **TRN* uses the *DITROFF* **.hy** command with an argument of 1 to turn automatic hyphenation on. Similarly, by default, **TRN* uses the *DITROFF* **.lg** command without any argument to turn the ligature mode on. In both cases, **TRN* uses the appropriate *DITROFF* command with an argument of 0 to turn these mechanisms off.

If no input files are specified, **TRN* reads from the standard input. Additionally, since **TRN* is intended to be a *DITROFF* preprocessor, it always writes to the standard output.

4. FFORTID

FFORTID's job is to take the *DITROFF* output which is formatted strictly left-to-right, to find occurrences of text in a right-to-left font, and to rearrange each line so that the text in each font is written in its proper direction. As illustrated in Figure 1, *FFORTID* deals exclusively with *DITROFF* output; it does not know and does not need to know anything about any of *DITROFF*'s preprocessors. Therefore, the results of using *FFORTID* with any of *DITROFF*'s preprocessors depends only on the *DITROFF* output generated by the use of the preprocessors. Furthermore, the output of *FFORTID* goes on to the same device drivers to which the *DITROFF* output would go; therefore, *FFORTID*'s output must be in the same form as that of *DITROFF*.

FFORTID is invoked by a command line of the form:

FFORTID [−r*font position list*][−w*paperwidth*]

The **−r** argument is used to specify which font positions are to be considered right to left. *FFORTID*, like *DITROFF*, recognizes up to 256 possible font positions (0–255). The actual number of available font positions depends only on the typesetting device and its associated *DITROFF* device driver. The default font direction for all possible font positions is left to right. Once the font direction is set, either by default or with the **−r** argument, it remains in effect throughout the entire document. Observe then that *FFORTID*'s processing is independent of what glyphs actually get printed for the mounted fonts. It processes the designated fonts as right-to-left fonts even if, in fact, the alphabet is that of a left-to-right language. Indeed, the examples of this section use a typewriter font in position 4 as the right-to-left font. In fact, it is possible that the same font be mounted in two different positions, only one of which is designated as a right-to-left font position. This is how the typewriter font can also be printed left-to-right in the same document.

The **−w** argument is used to specify the width of the paper, in inches, on which the document will be printed. As explained later, *FFORTID* uses the specified paper width to determine the width of the right margin. The default paper width is 8.5 inches and, like the font directions, remains in effect throughout the entire document.

In addition to the specified font directions, the results of *FFORTID*'s reformatting also depends on the document's *current formatting direction*, which can be either left to right or right to left. The default formatting direction is left to right and can be changed by the user at any point in the document through the use of the **.PL** and **.PR** macro commands. These commands set the current formatting direction to left to right and right to left, respectively.

If the current formatting direction is left to right, all formatting, filling, adjusting, indenting, etc., is to appear as occurring from left to right. In each output line, any sequence of right-to-left font characters is rotated about its center axis. For example, the following *DITROFF* input

```
.PL
.ll 3.25i
.ti + .5i
\f1This is an example of how FFORTID
reformats \f4 right-to-left
fonts \f1 when the current \f4 formatting
direction is \f1left-to-right.
```

produces when using just *DITROFF*:

```
        This is an example of how FFORTID refor-
mats   right-to-left  fonts  when the current
formatting direction is left-to-right.
```

When the same *DITROFF* output is presented to *FFORTID* with **−r4**, the following output is obtained:

```
        This is an example of how FFORTID refor-
mats   stnof  tfel-ot-thgir  when the current
si noitcerid gnittamrof left-to-right.
```

If the current formatting direction is right to left, all formatting, filling, adjusting, indenting, etc., is to appear as occurring from right to left. Each *DITROFF* output line is rotated about its center axis, including both the left and right margins. Then, any sequence of left-to-right font characters is rotated about its own center axis. For example, the following *DITROFF* input:

```
.PR
.ll 3.25i
.ti + .5i
\f4 This is an example of how FFORTID reformats
\f1DITROFF
output \f4 when the \f1 formatting
direction is \f4right-to-left.
```

produces with *DITROFF* alone:

```
                    This is an example of how FFORTID
            reformats   DITROFF   output    when    the
            formatting direction is right-to-left.
```

When the same *DITROFF* output is presented to *FFORTID* with **−r4**, the following is produced:

```
            DITROFF woh fo elpmaxe na si sihT
            eht    nehw    DITROFF    output    stamrofer
                    .tfel-ot-thgir formatting direction is
```

It is important to note that *DITROFF* uses the specified paper width to determine the margin widths in the reformatted output line. For instance, suppose that a document is formatted for printing on paper 8.5 inches wide with a left margin (page offset) of 1.5 inches and a line length of 6 inches. This results in a right margin of 1 inch. Suppose then that the text specifies a current formatting direction of right to left. Then, *FFORTID*'s reformatting of the output line results in left and right margins of 1 and 1.5 inches, respectively. This margin calculation works well for documents formatted entirely in one direction. However, as a document's formatting direction changes, the resulting margins widths are exchanged. Thus **.PL**'s right and left margins end up not being the same as **.PR**'s right and left margins. The use can make *FFORTID* preserve the left and right margins by specifying, with the **−w** argument, a paper width other than the actual paper width. This artificial paper width should be chosen so that both margins will appear to *FFORTID* to be as wide as the desired left margin. For example, for the document mentioned above, a specified paper width of 9.0 inches results in reformatted left and right margins of 1.5 inches each. The resulting excess in the right margin is just white space that effectively falls off the edge of the paper and does not effect the formatting of the document.

There is one exception to these simple rotation rules in that *FFORTID*, at present, makes no attempt to reverse any of *DITROFF*'s drawing functions, such as those used by *PIC* and *IDEAL* (which are also available directly to the user). It is therefore suggested that these drawing functions, and thus *PIC* and *IDEAL*, be used only when the current formatting direction is left to right. Additionally, due to the cleverness of the *DITROFF* output generated by most substantial *EQN* equations, it is suggested that *EQN*'s use also be limited to a left-to-right formatting direction for all but the simplest forms of equations. These rules do not in any way restrict the use of right-to-left fonts in the text dealt with by any of the preprocessors, but simply suggest that these particular preprocessors be used only when the formatting direction is left to right.

An additional point to keep in mind when preparing input both for *DITROFF*'s preprocessors and for *DITROFF* itself is that *FFORTID* rotates, as a unit, each sequence of characters of the same direction. In order to force *FFORTID* to rotate parts of a sequence independently, one must artificially separate them with a change to a font of the opposite direction. For example, the following input for a *TBL* table (⊕ stands for the tab character):

```
.PL
.TS
center;
cccc.
column 1⊕\f4column 2⊕column 3⊕\f1column 4
column 1⊕\f4column 2\f1\|⊕\f4column 3⊕\f1column 4
.TE
```

would produce

```
column 1   3 nmuloc   2 nmuloc   column 4
column 1   2 nmuloc   3 nmuloc   column 4
```

when presented to *FFORTID* with **−r4**. The additional small white space (Look for it! It is right there after the `2 nmuloc` in the second line of the table.) introduced by the artificial font break is usually negligible and if necessary can be balanced by adding additional small white spaces to other input.

In short, some experimentation on the part of the user may be necessary to achieve the desired results. However, keeping in mind *FFORTID*'s reformatting approach under both possible formatting directions, formatting results are for the most part predictable.

5. CONCLUSIONS

The *DITROFF*/*FFORTID* system meets the goals given in Section 1. It provides a bidirectional formatting system whose basic commands are identical to those of an existing system, *DITROFF*, and which adds only two macro commands for dealing with the new capabilities. It, in fact, makes use of the existing *DITROFF* program so that all associated prepasses, macros, experts, etc., may be used without change. It was built in a way that only one new program is needed, sitting between the unchanged *DITROFF* and its current unchanged device drivers.

The scheme used in the new program is general enough to be applied to other formatting systems.

The system is currently being extended with Arabic, Farsi, and Urdu, three other languages with almost identical right-to-left alphabets. In addition, the system is being extended to handle Japanese and Chinese. Although these do not require right-to-left processing, they have extremely large alphabets, which *DITROFF* is not normally equipped to handle, and they have problems being input with standard ASCII keyboards.

APPENDIX A. ISRAELI **TRN*

Under control of *ISRATRN*, when translation is turned on, the correspondence between the input character set and the letters of the Hebrew alphabet is as shown in Table 1. Note though that when actually creating *ISRATRN*, it is necessary to include an entry in the input-to-alphabet mapping table indicating the input character for all characters available in the output font that are desired to be useable, including punctuation characters. As mentioned above, any input not included in this input-to-alphabet table would be translated into no character and would not appear in the output.

Table 1 — Input to Alphabet Mapping for ISRATRN
(Any character not appearing in second or fourth column is translated into no character on output.)
(□ means "followed by end-of-word indication")

Represented Character	Input (Use any one listed)	Represented Character	Input (Use any one listed)
א	"	0	0
ב	b	1	1
ג	g	2	2
ד	d	3	3
ה	h	4	4
ו	o u v w	5	5
ז	z	6	6
ח	j	7	7
ט	T	8	8
י	i y	9	9
כ	k	)	(
ך	k□	(	)
ל	l	]	[
מ	m	[	]
ם	m□	}	{
נ	n	{	}
ן	n□	>	<
ס	s	<	>
ע	/	'	‘
פ	p f	‘	'
ף	p□ f□	*newlinenewline*	
צ	c	*tab*	*tab*
ץ	c□	*blank*	*blank*
ק	q	!	!
ר	r	@	@
ש	x	#	#
ת	t	$	$
אַ	A	%	%
אָ	Q	^	^
בֿ	V	*	*
בּ	B		
הּ	H	-	-
וֹ	O	+	+
וּ	U	=	=
װ	W	~	~
יִ	E	|	|
ײַ	I	\\	
ײ	Y	:	:
כּ	K	;	;
כֿ	J	,	,
ל	L	.	.
פֿ	F	/	/
פּ	P	?	?
שׂ	S		

where the end-of-word indications are *newline tab blank* , – . ; ? ! :) '] } > ˜ +
&

Table 2 – Alphabet to Output Mapping
(Also approximately ASCII coding)
(Any Hebrew font character not shown here is coded by itself.)

Character	Code	Character	Code
א	'	וֹ	O
ב	a	וּ	U
ג	b	װ	W
ד	c	ֵ	E
ה	d	ײַ	I
ו	e	ײ	Y
ז	f	כּ	K
ח	g	ף	L
ט	h	כֿ	J
י	i	פֿ	F
ך	j	פּ	P
כ	k	שׂ	S
ל	l	תּ	T
ם	m	0	0
מ	n	1	1
ן	o	2	2
נ	p	3	3
ס	q	4	4
ע	r	5	5
ף	s	6	6
פ	t	7	7
ץ	u	8	8
צ	v	9	9
ק	w	)	)
ר	x	(	(
ש	y	]	]
ת	z	[	[
אַ	A	}	}
אָ	Q	{	{
בֿ	V	>	>
בּ	B	<	<
הּ	H	'	,
		'	"

The following principles guided the construction of this **TRN*.

1. There are no possibly ambiguous diphthongs. For example if **th** represents ת (Tav), **t** represents ט (Tet), and **h** represents ה (Heh), then one cannot easily tell whether to take **th** as one letter or two.
2. The correspondence is phonetic, particularly if one recalls the Castellano B (pronounced as the Hebrew ב (Bet) which, as in Castellano, is pronounced sometimes as the English B and sometimes as almost the English V), the Castellano J (pronounced as the Hebrew ח (Chet)), the German or Serbian C (pronounced as the Hebrew צ (Tsaddik)), the Brazilian Portuguese X (pronounced sometimes as the Hebrew שׂ (Sin) and sometimes as the Hebrew ש (Shin)).
3. Some Hebrew letters have more than one pronunciation. Therefore whenever possible without ambiguity, each of the corresponding Latin letters maps to the same Hebrew letter. Thus שלום (Shalom) can be written as **xlom**, **xlwm**, **xlvm**, or **xlum**.
4. The non-letters " and / are used as codings for the so-called silent letters א

(Aleph) and ע (Ayin). This frees their more usual representations **a** and **e**, which are not always their pronunciations, for other purposes. It was decided to leave these to represent no characters so that they can be used to improve the human-readability of the phonetic input without affecting the output. Thus שלום can be also be written as **xalom** and אל על (El Al) can be written as **"l /l** or as **"el /al**.

5. Note that representing no letter is not the same as being the zero-width character, because the zero-width character is used as a letter for preventing letters sitting in the last position of a word from being converted into final form, so that for example, it is possible to specify בע"מ by **b/"m\&**.)
6. There are occasionally two Hebrew letters with the same pronunciation. In each case, the more commonly occurring Hebrew letter is assigned the lower case Latin letter with the same pronunciation, and the less commonly occurring Hebrew letter is assigned the corresponding upper case Latin letter. Thus מתמטיקה can be written as **matemaTiqah** and פרד can be written as **Fred**.
7. Digits are provided only as a convenience. When given in Hebrew, digit sequences are rotated in the same manner as the other text. Thus numerals have to be entered backwards, just as they do with a Hebrew typewriter.

The output of *ISRATRN* assumes that the output device is one of the Hebrew fonts in use or developed at UCLA. In all of these fonts, each basic Hebrew letter, each digit, and most of the punctuation symbols are addressed by their ASCII codes. A number of other nonstandard glyphs are provided, including some ligatures, and vowels. These nonstandard glyphs are addressed by the unused ASCII codes. The mapping of ASCII characters to glyphs is as shown in Table 2.

APPENDIX B. INPUT AND OUTPUT SAMPLE

This Appendix exhibits inputs to generate some interesting mixed English-Hebrew text. Specifically, The text below involves both Hebrew and *EQN* equations. The two Hebrew lines say "And G-d said" and "And there was light" respectively! Below this text is a possible *ISRATRN* input for this.

ויאמר אלוקים.

$$\epsilon_o \oint \mathbf{E} \cdot d\mathbf{S} = q$$

$$\oint \mathbf{B} \cdot d\mathbf{S} = 0$$

$$\oint \mathbf{B} \cdot d\mathbf{l} = \mu_o \epsilon_o \frac{d\Phi_{\mathrm{E}}}{dt} + \mu_o i$$

$$\oint \mathbf{E} \cdot d\mathbf{l} = \frac{-d\Phi_{\mathrm{B}}}{dt}$$

$\therefore$

$$c = \frac{1}{\sqrt{\mu_o \epsilon_o}}$$

ויהי אור

Input:

```
.ll 4.5i
.EQ
delim $$
define circint %% "\s+8\f(sy\z\(is\s0\fP\(ci" %%
define thf %% ".\v'-.5m'.\v'.5m'." %%
.EN
.PR
\f(HF%Nvay"mer "eloqim:%F\fR
.sp
.ce 100
$epsilon sub o circint bold E cdot d bold S = q$
.sp
$circint bold B cdot d bold S = 0$
.sp
$circint bold B cdot d bold l = mu sub o epsilon sub o {d PHI sub {bold E}}
over {d t} + mu sub o I$
.sp
$circint bold E cdot d bold l = {- d PHI sub {bold B}} over {d t}$
.ce 0
.sp
$thf$
.sp
.ce 100
$c = 1 over {sqrt {mu sub o epsilon sub o}}$
.ce 0
.sp
\f(HF%Nvayehi "or
```

APPENDIX C. HEBREW TITLE AND ABSTRACT

This appendix shows the Hebrew title and abstract for this paper.

DITROFF/FFORTID, התאמה של DITROFF של UNIX לעריכת מקסט דו-כיווני

ארי בוכמן

דניאל ברי

יעקוב גונצ'רובסקי

המאמר הזה מתאר את *DITROFF/FFORTID*, שהנה אוסף של קדם-מעבדים ואחר-מעבדים עבור *DITROFF* (*Device Independent Typesetter RunOFF*) של UNIX. *DITROFF/FFORTID* מאפשר עריכה של טקסט הכתוב בשתי שפות, כאשר האחת נכתבת מימין לשמאל והשניה

משמאל לימין (כמו עברית ואנגלית). התכניות מונחות על ידי טבלה, או נוצרות על
ידי מקרו, דבר המאפשר להן לעבד טקסט בכל שפה, בין אם היא כתובה משמאל
לימין ובין אם היא כתובה מימין לשמאל. זאת כל עוד fonts עם מערכת התוים
המתאימה נמצאים על התקן סדר־דפוס הנתמך על ידי *DITROFF*. קדם־המעבדים
מאפשרים קלט פונטי, חד כיווני של כל האלף־בית הדרוש, בהשתמשם רק בשני
האלף־בית הקימים על אמצעי הקלט (האותיות הגדולות נחשבות אלף־בית אחד
והאותיות הקטנות נחשבות אלף־בית אחר). מעבדים אלה, הנוצרים על ידי מקרו,
ניתנים להתאמה למבטא המשתמש, חוקי השפה באשר לצורת האות בהתאם למקומה
במילה, ושפת לוח־המקשים של המשתמש. אחר־המעבד משנה במידת הצורך את
כיוון העריכה כאשר הטקסט עובר לשפה בעלת כיוון שונה. אוסף התוכניות מאפשר
שמוש בכל אחד מקדם־המעבדים של *DITROFF* כמו *PIC*, *EQN*, *TBL*, ומספר נוהגי־
התקנים.

ACKNOWLEDGMENTS

The authors thank Danny Braniss for providing the Hebrew font used herein, and they thank Tim Morgan and Lou Salkind for their help in getting *DITROFF* to run and the ability to use the Berkeley Versatec fonts in the available output devices. They thank Brian Kernighan for his help in dealing with the intricacies of *DITROFF*. They also thank DDD Ltd., Israel, for providing a Hebrew terminal to the authors for experimentation.

REFERENCES

1. BECKER, J. D. Multilingual word processing. *Sci. Am. 251*, 1 (July 1984).
2. BUCHMAN, C., AND BERRY, D. M. User's manual for DITROFF/FFORTID: An adaptation of the UNIX DITROFF for formatting bi-directional text. Tech. Rep., Computer Science Department, UCLA, Los Angeles, Calif., July 1984.
3. GONCZAROWSKI, J. HNROFF/HTROFF Hebrew formatters based on NROFF/TROFF. Tech. Rep., Hebrew University, Jerusalem, Israel, 1980.
4. KERNIGHAN, B. W. A typesetter-independent TROFF. Computing Science Tech. Rep. 97, Bell Laboratories, Murray Hill, N.J., March 1982.
5. KERNIGHAN, B. W. PIC—A graphics language for typesetting, user manual. Computing Science Tech. Rept. 85, AT&T Bell Laboratories, Murray Hill, N.J., March 1982.
6. KERNIGHAN, B. W., AND CHERRY, L. L. Typesetting Mathematics—User's Guide, 2nd ed. AT&T Bell Laboratories, Murray Hill, N.J., 1978.
7. KERNIGHAN, B. W., AND PIKE, R. *The UNIX Programming Environment.* Prentice-Hall, Englewood Cliffs, N.J., 1984.
8. KNUTH, D. E. *The TEX Book.* Addison-Wesley, Reading, Mass., 1984.
9. LESK, M. E. Some applications of inverted indexes on the UNIX system. Computing Science Tech. Rep. 69, Bell Laboratories, Murray Hill, N.J., June 21, 1978.
10. LESK, M. E. TBL—A Program to Format Tables. Bell Laboratories, Murray Hill, N.J., 1978.
11. LESK, M. E. Typing Documents on the UNIX System: Using the -ms Macros with Troff and Nroff. Bell Laboratories, Murray Hill, N.J., March 1982.
12. OSSANA, J. F. NROFF/TROFF User's Manual. Bell Laboratories, Murray Hill, N.J., Oct. 11, 1976.
13. RITCHIE, D. M., AND THOMPSON, K. L. The UNIX time-sharing system. *Commun. ACM 17*, 7 (July 1974).
14. UNIX Programmer's Manual. Fourth Berkeley Distribution, University of California, Berkeley, Calif., Nov. 1981.
15. VAN WYK, C. J. IDEAL user's manual. Computing Science Tech. Rep. 103, Bell Laboratories, Murray Hill, N.J., Dec. 17, 1981.

Received May 1984; revised June 1985; accepted June 1985