Mixing right-to-left texts with left-to-right texts

Donald E. Knuth and Pierre MacKay

TeX was designed to produce documents that are read from left-to-right and top-to-bottom, according to the conventions of English and other Western languages. If such documents are turned 90°, they can also be read from top-to-bottom and right-to-left, as in Japan. Another 90° or 180° turn yields documents that are readable from right-to-left and bottom-to-top, or from bottom-to-top and left-to-right, in case a need for such conventions ever arises. However, TeX as it stands is not suitable for languages like Arabic or Hebrew, which are right-to-left and top-to-bottom.

It would not be difficult to use TeX for documents that are purely Arabic or purely Hebrew, by essentially producing the mirror image of whatever document is desired. A raster-oriented printing device could easily be programmed to reflect the bits from right to left as it puts them on the pages. (This is sometimes called "T-shirt mode", because it can be used to make iron-on transfers that produce readable T-shirt messages, when English language output is transferred to cloth after being printed in mirror image.)

Complications arise, however, when left-to-right conventions are mixed with right-to-left conventions in the same document. Consider an Arabic/English dictionary, or a Bible commentary that quotes Hebrew, or a Middle-Eastern encyclopedia that refers to Western names in roman letters; such documents, and many others, must go both ways.

The purpose of this paper is to clarify the issues involved in mixed-direction document production, from the standpoint of a Western author or reader or software implementor. We shall also consider changes to TeX that will extend it to a bidirectional formatting system.

1. Terminology and conventions. Let us say for convenience that an *L-text* is textual material that is meant to be read from left to right, and an *R-text* is textual material that is meant to be read from right to left. Similarly we might say that English and Spanish are L-languages, while Arabic and Hebrew are R-languages.

In order to make this paper intelligible to English readers who are unfamiliar with R-languages, we shall use "reflected English", i.e., **hsilgnE**, as an R-language. All texts in reflected English will be typeset in **dednetxE dloB nredoM retupmoC** type, which is a reflected version of Computer Modern Bold Extended type. To translate from English to **hsilgnE** and back again, one simply needs to reverse the order of reading. Both English and **hsilgnE** are pronounced in the same way, except that **hsilgnE** should be spoken in a louder and/or deeper voice, so that a listener can distinguish it.

2. The simplest case. It's not difficult to typeset single R-language words in an L-text document. TeX will work fine if you never need to deal with R-texts of more than one word at a time; all you have to do is figure out a macro that will reverse isolated words.

Let's suppose that we want to type '`the |English| script`' in order to typeset 'the **hsilgnE** script' with TeX. All we need is a font for **hsilgnE**, called `xbmc10`, say, and the following macros:

```
\font\revrm=xbmc10       \hyphenchar\revrm=-1
\catcode`\|=\active
\def|#1|{{\revrm\reflect#1\empty\tcelfer}}
\def\reflect#1#2\tcelfer{\ifx#1\empty\else\reflect#2\tcelfer#1\fi}
```

(The `xbmc10` font can be generated like `cmbx10` with the extra METAFONT statement

```
extra_endchar := extra_endchar &
 "currentpicture:=currentpicture reflectedabout((.5[l,r],0),(.5[l,r],1))"
```

added to the parameter file. It has the same character widths as `cmbx10`.)

3. Alternating texts. But the simple approach sketched above does not work when there are multiword R-text phrases, i.e., **sesarhp txet-R drowitlum**, embedded in an L-text document—because of the possibility

of line breaks, i.e., **because of the possibility of line breaks**. For example, let's consider the problem of typesetting the following paragraph:*

```
Leonardo da Vinci made a sweeping statement in his notebooks:
|``Let no one who is not a mathematician read my works.''|
In fact, he said it twice, so he probably meant it.
```

Here are samples of the proper results, considering two different column widths:

Leonardo da Vinci made a sweeping statement in his notebooks: **"Let no one who is not a mathematician read my works."** In fact, he said it twice, so he probably meant it.

Leonardo da Vinci made a sweeping statement in his notebooks: **"Let no one who is not a mathemati-cian read my works."** In fact, he said it twice, so he probably meant it.

Notice that the R-text in each line is reflected; in particular, a hyphen that has been inserted at the right of an R-segment will appear at the left of that segment.

How can we get TEX to do this? The best approach is probably to extend the driver programs that produce printed output from the DVI files that TEX writes, instead of trying to do tricky things with TEX macros. Then TEX itself merely needs to put special codes into the DVI output files, in order to tell the "DVI-IVD" drivers what to do.

For example, one idea that almost works is to put '\special{R}' just before an R-text begins, and '\special{L}' just after it ends. In other words, we can change the '|' macro in our earlier example to the simple form

```
\def|#1|{{\revrm\special{R}#1\special{L}}}
```

which does not actually reverse the characters; we can also leave the '\hyphenchar' of \revrm at its normal value, so that R-texts will be hyphenated. Line breaking will proceed in the normal way, and it will be the job of the DVI-IVD driver program to reflect every segment that it sees between an R and an L.

Reflecting might involve arbitrary combinations of characters, rules, accents, kerns, etc.; for example, the R-text might be in **français**, or it might even refer to **TEX**!

4. An approach to implementation. In order to understand how DVI-IVD programs might do the required tasks, we need to look into the information that TEX puts into a DVI file. The basic idea is that whenever TEX outputs an hbox or a vbox, the DVI file gets a '*push*' command, followed by various commands to typeset the box contents, followed by a '*pop*' command. Therefore we can try the following strategy:

a) Whenever '\special{R}' is found in the DVI file, remember the current horizontal position h_0 and vertical position v_0; also remember the current location p_0 in the DVI file. Set $c \leftarrow 0$. Then begin to skim the next DVI instructions instead of actually using them for typesetting; but keep updating the horizontal and vertical page positions as usual.

b) When '\special{L}' is found in the DVI file, stop skimming instructions. Then typeset all instructions between p_0 and the current location, in mirror-reflected mode, as explained below.

c) When '*push*' occurs when skimming instructions, increase c by 1.

d) When '*pop*' occurs when skimming instructions, there are two cases. If $c > 0$, decrease c by 1. (This '*pop*' matches a previously skimmed '*push*'.) But if $c = 0$, effectively insert '\special{L}' at this point and '\special{R}' just after the very next '*push*'.

The mirror-reflected mode for DVI commands in positions p_0 to p_1 in the DVI file, beginning at (h_0, v_0) and ending at (h_1, v_1), works like this: A character of width w whose box sits on the baseline between (h, v) and $(h+w, v)$ in normal mode should be placed so that its box sits on the baseline between $(h'-w, v)$ and (h', v) in mirror mode, where h' is defined by the equation

$$h - h_0 \;=\; h_1 - h'.$$

Similarly, a rule of width w whose lower edge runs from (h, v) to $(h+w, v)$ in normal mode should run from $(h'-w, v)$ to (h', v) in mirror mode.

* After Leonardo lost the use of his right hand, he began to make lefthanded notes in mirror writing. Of course, he actually wrote in **Italian** instead of **English**.

5. Fixing bugs. We stated above that the approach just sketched will "almost" work. But it can fail in three ways, when combined with the full generality of TEX. First, there might be material "between the lines" that is inserted by `\vadjust` commands; this material might improperly be treated as R-text. Second, the suggested mechanism doesn't always find the correct left edge of segments that are being reflected, since the reflection should not always begin at the extreme left edge of a typeset line; it should begin after the `\leftskip` glue and before other initial spacing due to things like accent positioning. Third, certain tricks that involve '`\unhbox`' can make entire lines disappear from the DVI file; however, this problem is not as serious as the other two, because people shouldn't be playing such tricks.

A much more reliable and robust scheme can be obtained by building a specially extended version of TEX, which puts matching special commands into every line that has reflected material. It is not difficult to add this additional activity to TEX's existing line-breaking mechanism; the details appear in an appendix below. When this change has been made, parts (c) and (d) of the DVI-IVD skimming algorithm can be eliminated, since the case $c = 0$ will never arise in part (d).

6. L-chauvinism. We have been discussing mixed documents as if they always consist of R-texts inserted into L-texts; but people whose native script is right-to-left naturally think of mixed documents as the insertion of L-texts into R-texts. In fact, there are two ways to read every page of a document, one in which the eye begins to scan each line at the left and one in which the eye begins to scan each line at the right.

The Leonardo illustration above is an example of the first kind, and we shall call it an *L-document*. To read a given line of an L-document, you start at the left and read any L-text that you see. Whenever your eyes encounter an R-character, they skim ahead to the end of the next R-segment (i.e., until the next L-character, or until the end of the line, whichever comes first); then you read the R-segment right-to-left, and continue as before. The rules for reading an R-document are similar, but with right and left reversed.

It's usually possible to distinguish an L-document from an R-document because of the indentation on the first line of a paragraph and/or the blank space on the last line. For example, the R-documents that correspond to the two L-document settings of the paragraph about Leonardo look like this:

Leonardo da Vinci made a sweeping statement in his **"Let no one who is not a mathematician** notebooks: In fact, he said it twice, so he probably **read my works."** meant it.

Leonardo da Vinci made a sweep- **"Let** ing statement in his notebooks: **no one who is not a mathemati-** **cian read my works."** In fact, he said it twice, so he probably meant it.

We can imagine that these R-documents were composed on an R-terminal and processed by **XeTeX** from an **input file** that looks like this:

```
|Leonardo da Vinci made a sweeping statement in his notebooks:| ''Let
                                                    no one who is not a mathematician read my
                                                    works.'' |In fact, he said it twice, so he probably meant it.|
```

In this case it is the L-text, not the R-text, that is enclosed in |'s. (The reader is urged to study this example carefully; there *is* **method** in't!)

A poet could presumably construct interesting poems that have both L-meanings and R-meanings, when read as L-documents and R-documents.

Notice that our examples from Leonardo have used boldface quotation marks (i.e., the quotation marks of **English**), so that these marks belong to the text being quoted. This may seem erroneous; but it is in fact a necessary convention in documents that are meant to display no favoritism between L-readers and R-readers, because it ensures that the quotation marks will stay with the text being reflected. (See the examples of contemporary typesetting at the end of this paper.) If we had put the quotation marks into English rather than **English**, the R-documents illustrated above would have looked very strange indeed:

Leonardo da Vinci made a sweeping statement in his **"Let no one who is not a mathematician** notebooks: **"** In fact, he said it twice, so he probably**.read my works** meant it.

Leonardo da Vinci made a sweep- **"Let** ing statement in his notebooks: **"no one who is not a mathemati-** **cian read my works.e** In fact, he said it twice, so he probably meant it.

7. Multi-level mixing. The problems of mixed R- and L-typesetting go deeper than this, because there might be an L-text inside an R-text inside an L-text. For example, we might want to typeset a paragraph whose TEX source file looks like this:

```
\R{Alice} said, \R{``You think English is \L{`English written backwards'};
but to me, \L{English} is English written backwards.  I'm sure \L{Knuth}
and \L{MacKay} will both agree with me.''}  And she was right.
```

An intelligent bidirectional reader will want this to be typeset as if it were an R-document inside an L-document. In other words, the eyes of such a reader will naturally scan some of the lines beginning at the left, and some of them beginning at the right. Here are examples of the desired output, set with two different line widths:

"You think English is Alice said, 'English written backwards'; but to me, English is English written backwards. I'm sure Knuth and MacKay will both agree with me." And she was right.

"You think English is Alice said, 'En- glish written backwards'; but to me, English is English written backwards. I'm sure Knuth and MacKay will both agree with me." And she was right.

(Look closely.)

Multi-level documents are inherently ambiguous. For example, the right-hand setting above might be interpreted as the result of

```
...\R{... I'm sure and \L{MacKay} will both agree with} Knuth \R{me.''}...
```

and the left-hand setting would also result from a source file like this(!)

```
\indent\R{``You think English is \L{said,} Alice
\L{`English}; but to me,} written backwards'
\R{written backwards.} \R{\L{English} is English}
will both} MacKay \R{and} Knuth \R{I'm sure
\L{And she} agree with me.''} was right.
```

except for slight differences in spacing due to TEX's "spacefactor" for punctuation.

In general, we have `\R{\L{a}\L{b}}` = ba, hence any permutation of the characters on each line is theoretically possible. A reader has to figure out which of the different ways to parse each line makes most sense. Yet there is unanimous agreement in Middle Eastern countries that a mixture of L-document and R-document styles is preferable to an unambiguous insistence on L-reading or R-reading throughout a document, because it is so natural and because the actual ambiguities arise rarely in practice. The quotation marks in the example above make it possible to reconstruct the invisible \R's and \L's; in this way an author can cooperate with a literate reader so that the meaning is clear.

Multi-level texts arise not only when quotes are inside quotes or when R-document footnotes or illustrations are attached to L-documents; they also arise when mathematics is embedded in R-text. For example, consider the TEX source code

```
The \R{English} version of `the famous identity $e^{i\pi}+1=0$ due to Euler'
is \R{`the famous identity $e^{i\pi}+1=0$ due to Euler'}.
```

It should be typeset like this:

The English version of 'the famous identity $e^{i\pi}+1=0$ due to Euler' is 'the famous identity $e^{i\pi}+1=0$ due to Euler'.

An extension of TEX called TEX-XET, described in the appendix, properly handles multi-level mixtures including math, as well as the simpler case of alternating R-texts and L-texts.

8. Conclusions. When right-to-left and left-to-right texts are mixed in the same document, problems can arise that are more subtle than simple examples might suggest. The difficulties can be overcome by extending TEX to TEX-XET and by extending DVI drivers to DVI-IVD drivers. Neither of these extensions is extremely complex.

585. The description of DVI commands is augmented by two new ones at the end:

begin_reflect 250. Begin a (possibly recursive) reflected segment.

end_reflect 251. End a (possibly recursive) reflected segment.

Commands 250–255 are undefined in normal DVI files, but 250 and 251 are permitted in the special 'DVI-IVD' files produced by this variant of TEX.

When a DVI-IVD driver encounters a *begin_reflect* command, it should skim ahead (as previously described) until finding the matching *end_reflect*; these will be properly nested with respect to each other and with respect to *push* and *pop*. After skimming has located a segment of material to be reflected, that segment should be re-scanned and obeyed in mirror-image mode as described earlier. The reflected segment might recursively involve *begin_reflect*/*end_reflect* pairs that need to be reflected again.

586. Two new definitions are needed:

```
define begin_reflect = 250  { begin a reflected segment (allowed in DVI-IVD files only) }
define end_reflect = 251  { end a reflected segment (allowed in DVI-IVD files only) }
```

638. At the beginning of *ship_out*, we will initialize a stack of \beginL and \beginR instructions that are currently in force; this is called the LR stack, and it is maintained with the help of two global variables called *LR_ptr* and *LR_tmp* that will be defined later. The instructions inserted here (just before testing if *tracing_output* > 0) say that on the outermost level we are typesetting in left-to-right mode. The opening '**begin**' is replaced by:

```
begin LR_ptr ← get_avail; info(LR_ptr) ← 0;  { begin_L_code at outer level }
```

639. At the end of *ship_out*, we want to clear out the LR stack. Thus, '*flush_node_list*(*p*)' is replaced by:

```
flush_node_list(p); ⟨Flush the LR stack 1382⟩;
```

649. The *hpack* routine is modified to keep an LR stack as it packages a horizontal list, so that errors of mismatched \beginL...\endL and \beginR...\endR pairs can be detected and corrected. Changes are needed here at the beginning of the procedure and at the end.

```
function hpack(p : pointer; w : scaled; m : small_number): pointer;
                    ⋮
  b: integer;  { badness of the new box }
  LR_ptr, LR_tmp: pointer;  { for LR stack maintenance }
  LR_problems: integer;  { counts missing begins and ends }
begin LR_ptr ← null; LR_problems ← 0;
r ← get_node(box_node_size);
                    ⋮
common_ending: ⟨Finish issuing a diagnostic message for an overfull or underfull hbox 663⟩;
exit: ⟨Check for LR anomalies at the end of hpack 1387⟩;
hpack ← r;
end;
```

877. Similarly, the *post_line_break* should keep an LR stack, so that it can output \endL or \endR instructions at the ends of lines and \beginL or \beginR instructions at the beginnings of lines. Changes occur at the beginning and the end of this procedure:

```
procedure post_line_break(final_widow_penalty : integer);
                    ⋮
  cur_line: halfword;  { the current line number being justified }
  LR_ptr, LR_tmp: pointer;  { for LR stack maintenance }
begin LR_ptr ← null;
⟨Reverse the links of the relevant passive nodes, setting cur_p to the first breakpoint 878⟩;
```

```
        ⋮
prev_graf ← best_line − 1: ⟨Flush the LR stack 1382⟩;
end;
```

880. The new actions to be performed when broken lines are being packaged are accomplished by three new steps added to this section of the program.

```
⟨Justify the line ending at breakpoint cur_p, and append it to the current vertical list, together with
      associated penalties and other insertions 880⟩ ≡
  ⟨Insert LR nodes at the beginning of the current line 1383⟩;
  ⟨Adjust the LR stack based on LR nodes in this line 1384⟩;
  ⟨Modify the end of the line to reflect the nature of the break and to include \rightskip; also set the
      proper value of disc_break 881⟩;
  ⟨Insert LR nodes at the end of the current line 1385⟩;
  ⟨Put the \leftskip glue at the left and detach this line 887⟩;
        ⋮
```

1090. We add '*vmode* + *LR*' as a new subcase after '*vmode* + *ex_space*' here. This means that the new primitive operations will become instances of what *The TEXbook* calls a ⟨horizontal command⟩.

1196. Math-in-text will be formatted left-to-right, because two new 'append' instructions are inserted into this section of the code.

```
⟨Finish math in text 1196⟩ ≡
  begin tail_append(new_math(math_surround, before));
  ⟨Append a begin_L to the tail of the current list 1380⟩;
  cur_mlist ← p; cur_style ← text_style; mlist_penalties ← (mode > 0); mlist_to_hlist;
  link(tail) ← link(temp_head);
  while link(tail) ≠ null do tail ← link(tail);
  ⟨Append an end_L to the tail of the current list 1381⟩;
  tail_append(new_math(math_surround, after)); space_factor ← 1000; unsave;
  end
```

1341. The new primitive operations put new kinds of whatsit nodes into horizontal lists. Therefore two additional definitions are needed here:

```
define LR_node = 4   { subtype in whatsits that represent \beginL, etc. }
define LR_type(#) ≡ mem[# + 1].int   { the sub-subtype }
```

1344. Here's where the new primitives get established.

```
define immediate_code = 4   { command modifier for \immediate }
define begin_L_code = 0   { command modifier for \beginL }
define begin_R_code = 1   { command modifier for \beginR }
define end_L_code = 2   { command modifier for \endL }
define end_R_code = 3   { command modifier for \endR }
define begin_LR(#) ≡ (LR_type(#) < end_L_code)
define begin_LR_type(#) ≡ (LR_type(#) − end_L_code)
⟨Put each of TEX's primitives into the hash table 226⟩ +≡
  primitive("beginL", LR, begin_L_code);
  primitive("beginR", LR, begin_R_code);
  primitive("endL", LR, end_L_code);
  primitive("endR", LR, end_R_code);
  primitive("openout", extension, open_node);
        ⋮
```

1346. The new primitives call for a new case of cases here.

```
LR: case chr_code of
  begin_L_code: print_esc("beginL");
  begin_R_code: print_esc("beginR");
  end_L_code: print_esc("endL");
  othercases print_esc("endR")
  endcases;
```

1356. We also need to be able to display the newfangled whatsits.

```
LR_node: case LR_type(p) of
    begin_L_code: print_esc("beginL");
    begin_R_code: print_esc("beginR");
    end_L_code: print_esc("endL");
    othercases print_esc("endR")
    endcases;
```

1357, 1358. Copying and deleting the new nodes is easy, since they can be handled just like the `\closeout` nodes already present. We simply replace '*close_node*' by '*close_node*, *LR_node*' in these two sections.

1360. We used to *do_nothing* here, but now we must *do_something*:

```
⟨ Incorporate a whatsit node into an hbox 1360 ⟩ ≡
  if subtype(p) = LR_node then ⟨ Adjust the LR stack for the hpack routine 1386 ⟩
```

This code is used in section 651.

1366.

```
⟨ Output the whatsit node p in an hlist 1366 ⟩ ≡
  if subtype(p) ≠ LR_node then out_what(p)
  else ⟨ Output a reflection instruction if the direction has changed 1388 ⟩
```

This code is used in section 622.

1376. Most of the changes have been saved up for the end, so that the section numbers of TEX in [2] can be left unchanged. Now we come to the real guts of this extension to mixed-direction texts.

First, we allow the new primitives in horizontal mode, but not in math mode:

```
⟨ Cases of main_control that build boxes and lists 1056 ⟩ +≡
hmode + LR: begin new_whatsit(LR_node, small_node_size); LR_type(tail) ← cur_chr; end;
mmode + LR: report_illegal_case;
```

1377. A number of routines are based on a stack of one-word nodes whose *info* fields contain either *begin_L_code* or *begin_R_code*. The top of the stack is pointed to by *LR_ptr*, and an auxiliary variable *LR_tmp* is available for stack manipulation.

```
⟨ Global variables 13 ⟩ +≡
LR_ptr, LR_tmp: pointer;  { stack of LR codes and temp for manipulation }
```

1378.

```
⟨ Declare functions needed for special kinds of nodes 1378 ⟩ ≡
function new_LR(s : small_number): pointer;
  var p: pointer;  { the new node }
  begin p ← get_node(small_node_size); type(p) ← whatsit_node; subtype(p) ← LR_node; LR_type(p) ← s;
  new_LR ← p;
  end;
```

See also section 1379.

This code is used in section 161.

1379.

```
⟨ Declare functions needed for special kinds of nodes 1378 ⟩ +≡
function safe_info(p : pointer): integer;
  begin if p = null then safe_info ← −1 else safe_info ← info(p);
  end;
```

1380. ⟨ Append a *begin_L* to the tail of the current list 1380 ⟩ ≡
tail_append(*new_LR*(*begin_L_code*))
This code is used in section 1196.

1381. ⟨ Append an *end_L* to the tail of the current list 1381 ⟩ ≡
tail_append(*new_LR*(*end_L_code*))
This code is used in section 1196.

1382. When the stack-manipulation macros of this section are used below, variables *LR_ptr* and *LR_tmp* might be the global variables declared above, or they might be local to *hpack* or *post_line_break*.

```
define push_LR(#) ≡
        begin LR_tmp ← get_avail; info(LR_tmp) ← LR_type(#); link(LR_tmp) ← LR_ptr;
        LR_ptr ← LR_tmp;
        end
define pop_LR ≡
        begin LR_tmp ← LR_ptr; LR_ptr ← link(LR_tmp); free_avail(LR_tmp);
        end
```

⟨ Flush the LR stack 1382 ⟩ ≡

```
while LR_ptr ≠ null do pop_LR
```

This code is used in sections 639 and 877.

1383. ⟨ Insert LR nodes at the beginning of the current line 1383 ⟩ ≡

```
while LR_ptr ≠ null do
  begin LR_tmp ← new_LR(info(LR_ptr)); link(LR_tmp) ← link(temp_head);
  link(temp_head) ← LR_tmp; pop_LR;
  end
```

This code is used in section 880.

1384. ⟨ Adjust the LR stack based on LR nodes in this line 1384 ⟩ ≡

```
q ← link(temp_head);
while q ≠ cur_break(cur_p) do
  begin if ¬is_char_node(q) then
    if type(q) = whatsit_node then
      if subtype(q) = LR_node then
        if begin_LR(q) then push_LR(q)
        else if LR_ptr ≠ null then
            if info(LR_ptr) = begin_LR_type(q) then pop_LR;
  q ← link(q);
  end
```

This code is used in section 880.

1385. We use the fact that q now points to the node with `\rightskip` glue.

⟨ Insert LR nodes at the end of the current line 1385 ⟩ ≡

```
if LR_ptr ≠ null then
  begin s ← temp_head; r ← link(s);
  while r ≠ q do
    begin s ← r; r ← link(s);
    end;
  r ← LR_ptr;
  while r ≠ null do
    begin LR_tmp ← new_LR(info(r) + end_L_code); link(s) ← LR_tmp; s ← LR_tmp; r ← link(r);
    end;
```

```
    link(s) ← q;
    end
```

This code is used in section 880.

1386. ⟨ Adjust the LR stack for the *hpack* routine 1386 ⟩ ≡

```
  if begin_LR(p) then push_LR(p)
  else if safe_info(LR_ptr) = begin_LR_type(p) then pop_LR
    else begin incr(LR_problems);
      while link(q) ≠ p do q ← link(q);
      link(q) ← link(p); free_node(p, small_node_size); p ← q;
      end
```

This code is used in section 1360.

1387. ⟨ Check for LR anomalies at the end of *hpack* 1387 ⟩ ≡

```
  if LR_ptr ≠ null then
    begin while link(q) ≠ null do q ← link(q);
    repeat link(q) ← new_LR(info(LR_ptr) + end_L_code); q ← link(q);
      LR_problems ← LR_problems + 10000; pop_LR;
    until LR_ptr = null;
    end;
  if LR_problems > 0 then
    begin print_ln; print_nl("\endL␣or␣\endR␣problem␣(");
    print_int(LR_problems div 10000); print("␣missing,␣");
    print_int(LR_problems mod 10000); print("␣extra");
    LR_problems ← 0; goto common_ending;
    end
```

This code is used in section 649.

1388. ⟨ Output a reflection instruction if the direction has changed 1388 ⟩ ≡

```
  if begin_LR(p) then
    begin if safe_info(LR_ptr) ≠ LR_type(p) then
      begin synch_h; synch_v; dvi_out(begin_reflect);
      end;
    push_LR(p);
    end
  else if safe_info(LR_ptr) = begin_LR_type(p) then
      begin pop_LR;
      if info(LR_ptr) + end_L_code ≠ LR_type(p) then
        begin synch_h; synch_v; dvi_out(end_reflect);
        end;
      end
    else confusion("LR")
```

This code is used in section 1366.

Final Important Note

The extensions to TEX just described are "upward compatible" with standard TEX, in the sense that ordinary TEX programs will still run correctly (although more slowly) on TEX-XET. However, TEX-XET must *not* be called a new version of 'TEX', even though it runs all TEX programs; the reason is, of course, that TEX will not run all TEX-XET programs.

A name change is necessary to distinguish all programs that do not agree precisely with the real TEX. Anybody who runs a program called 'TEX' should be able to assume that it will give identical results from all its implementations.

Bibliography

[1] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting* (Reading, Mass.: Addison Wesley, 1986).

[2] Donald E. Knuth, *TEX: The Program*, Volume B of *Computers & Typesetting* (Reading, Mass.: Addison Wesley, 1986).

[3] Pierre MacKay, "Typesetting problem scripts," *Byte* **11**, 2 (February 1986), 201–218.

Examples of Typesetting Practice

1. From *Textus* **5** (1966), p. 12; Magnes Press, Hebrew University of Jerusalem. (Notice the Hebrew quotation marks surrounding the Hebrew title in footnote 6.)

ters adhered,[10] and which may have been similar to that adopted, by normative Jewry presumably somewhat later, during the period of the Second Temple.[10]

Frag. E. Yadin correctly states: "Sanders' cautious indication '103 (? 104)' can now be eliminated" (*ib.*, p. 5).

6 Sanders' *editio princeps* of Ps. 151 already has been discussed by various scholars. The present author deals with the text of Ps. 151, and its literary genre in: "מזמורים חיצוניים בלשון העברית מקומראן", *Tarbiz* 35 (1966) 214–228.

7 See: W. Wright, "Some Apocryphal Psalms in Syriac", PSBA 9 (1887) 257–266; M. Noth, "Die fuenf apokryphen Psalmen", ZAW 48 (1930) 1–23.

2. Fragments from the third edition of William Wright's classic nineteenth century grammar of Arabic, volume 2, pages 295–297. (Notice the page break in the midst of right-to-left text, and some left-to-right brackets.)

gnawed at us; كُنْتُمْ خَيْرَ أُمَّةٍ أُخْرِجَتْ لِلنَّاسِ *ye are the best people that has been brought forth (created) for mankind;* مَشَيْنَ كَمَا ٱهْتَزَّتْ رِمَاحٌ تَسَفَّهَتْ أَعَالِيَهَا مَرُّ ٱلرِّيَاحِ ٱلنَّوَاسِمِ *they walked as spears wave, the tops of which are bent by the passing of gentle breezes;* إِنَارَةُ ٱلْعَقْلِ

296 PART THIRD.—*Syntax.* [§ 152

A مَكْسُوفٌ بِطَوْعِ هَوًى *the brightness of the intellect is obscured* (or *eclipsed*) *by obeying lust.* As the above examples show, this agreement

§ 152] *Sentence and its Parts.—Concord of Predicate & Subject.* 297

verb is placed after a collective subject (see § 148); as A وَلٰكِنَّ أَكْثَرَ ٱلنَّاسِ لَا يَشْكُرُونَ *but the greatest part of mankind are thankless;* فَرِيقٌ مِنْهُمْ يَخْشَوْنَ ٱلنَّاسَ *a part of them are afraid of men:* [ٱتْرُكُوا ٱلتُّرْكَ مَا تَرَكُوكُمْ *let the Turks alone as long as they let you alone;* لِأَنَّ جَيْشَهُ هَلَكُوا *because his army had perished*].

3. From page 233 of the same book. Here right-reading texts are equated with = signs; the left sides of each equation are to be read first.

اَلصَّلٰوةُ فِى ٱلسَّاعَةِ, i.e. صَلٰوةُ ٱلسَّاعَةِ ٱلْأُولَى = صَلٰوةُ ٱلْأُولَى understood ; B جَانِبُ ٱلْغَرْبِيِّ (see § 77). Similarly, some grammarians consider مَسْجِدُ ٱلْمَكَانِ ٱلْجَامِعِ = مَسْجِدُ ٱلْجَامِعِ ,جَانِبُ ٱلْمَكَانِ ٱلْغَرْبِيِّ = بَقْلَةُ ٱلْحَبَّةِ ٱلْحَمْقَاءِ = بَقْلَةُ ٱلْحَمْقَاءِ ,مَسْجِدُ ٱلْوَقْتِ ٱلْجَامِعِ or أَفْضَلُ ٱلْآخِرَةِ = دَارُ ٱلْحَيَاةِ ٱلْآخِرَةِ*. Here too the constructions

4. From *Bulletin of the Iranian Mathematical Society* **8** (Tehran, 1978), p. 78L. (Left-to-right mathematics in right-to-left text.)

کلمه جدید بازشود ازعبارت زیرتعیین میشود

$$\frac{n-1}{2m} + \frac{2(n-1)(n-2)}{3m^2} + \frac{3(n-1)(n-2)}{4m^3} + \ldots.$$

که تقریبا مساوی $1/(1-\alpha) + [\log_e(1-\alpha)]/\alpha$ است که در آن $e=2.71828$ میباشد .بنابراین اضافه کردن کلمات به جدول باکمك الگوریتم F کارچندان

5. From *Introduction to Mathematics* [sikitamatemeL ovaM] by Abraham A. Fraenkel, vol. 1 (Jerusalem, 1942), p. 38. (Page numbers are '96–90' because '90' and '96' are Hebrew numbers.)

בהשתמשנו במושג הקונגרואנציה, נוכל לכתוב את המשפט הקטן של פ ר מ ה בצורה: $a^p \equiv a \pmod{p}$, ואת משפט ו י ל ס ו ן בצורה:

$$1\cdot 2\cdot 3\cdots(p-1) \equiv -1 \pmod{p}.$$

1. מן המלים הרומיות congruens = מתאים, modulus = אומן, מדה.
2. עיין בהוכחה שנתן M. HAMBURGER בשנת 1896 בכרך ה 116 של ה Journal f. d. reine u. ang. Mathematik (עמ' 90—96) לנוסחה של גאוס משנת 1802, הקובעת את התאריך הנוצרי

6. Page 200 of the same book illustrates the difference between ellipses '⋯' in formulas and ellipses in the text. None of this book's math-in-text is broken between lines.

בריבוע של מספר אי־זוגי, ב) שאפשר לפתור את הבעיה אם n מתפרק למספרים ראשוניים שונים $n=p_1\cdot p_2\cdot\cdots\cdot p_k$, בתנאי שהבעיה נפתרת בשביל כל הערכים $n=p_1$, $n=p_2$, ..., $n=p_k$. נסתפק בכך שנבאר את הטענה האחרונה ביחס לדוגמה $n=15=3\cdot 5$. ידוע בוודאי לרוב הקוראים שאפשר לבנות בעזרת

⋮

קודם כל יצויין שבמקרה זה, $n=p$, כל שרשי המשואה [1] (לשון אחר: כל שרשי המשואה $x^p=1$, פרט ל 1) הנם שרשים „פרימיטיביים" במובן שהוגדר בעמ' 199 ; שהרי כל ערך k מתוך הסדרה $(1, 2, \ldots, p-1)$ זר ל p. יהי אפוא $n=p$, ו ξ שורש כלשהו של המשואה [1] ; אז יהיו $\xi, \xi^2, \ldots, \xi^{p-1}$ כל שרשיה של המשואה [1] [1]. השרשים האלה נקראים שרשי היחידה.