# Bidirectional line breaking with TEX macros

Klaus Lagally
Institut für Informatik, Universität Stuttgart
Breitwiesenstraße 20-22, D-70565 Stuttgart, Germany
Internet: lagally@informatik.uni-stuttgart.de

**Abstract**

TEX has originally been designed with European languages in mind, and thus, whenever a paragraph contains text portions running in opposite directions, e.g. when combining English and Arabic or Hebrew in the same document, the task of line-breaking becomes rather complicated.

For a clean solution, Knuth and MacKay have proposed an modification to TEX, `TeX-XeT`, which will produce an extended DVI file containing additional directional information to be exploited by a modified DVI driver; and by now there exist several implementations of this idea, including `TeX--XeT` that produces a standard DVI file. The main drawback is just that we have to go outside the TEX standard.

We present a portable technique to handle bidirectional line-breaking by using TEX macros alone, albeit at some sacrifice in quality. This technique has been implemented in the version 3.02 of the author's multi-lingual ArabTEX package.

## Introduction

Folklore says, and most people believe, that correct line-breaking of a paragraph consisting of mixed left-to-right and right-to-left passages is impossible to do with TEX. Knuth and MacKay [KM87] explain why it is difficult, and propose a bidirectional extension of TEX. There have since been several implementations, e.g. `TeX-XeT` and `TeX--XeT` which provide additional facilities to handle bidirectional typesetting. Whereas the problem is thus solved in principle, still many users have no access to these extensions, or are reluctant to leave the TEX standard.

We show that the problem can also be solved with TEX alone, although in a rather complicated way, and at some expense and some loss of quality. The solution we present is a simplified version of what actually happens inside our ArabTEX package [Lag92a, Lag93], and should not be considered as an instruction for use of the real system.[1]

In a bidirectional typesetting system every text element has an implicit or explicit direction attribute, called L-text and R-text in [KM87]. TEX is very good at processing L-text whereas it needs some assistance for R-text. To be able to do this we assume that R-text passages are identified by appropriate markup commands. For the sake of this presentation we ignore the fact that L-text and R-text passages may be nested within LATEX environments possibly modifying the layout parameters, the `\par` command and the `\everypar` mechanism; and we restrict our discussion to the case of a single paragraph of one kind containing an insertion of the other kind, and possibly contained within a `\vbox`.

There are two sorts of paragraphs to consider: LR-paragraphs processed by TEX, and RL-paragraphs for which we have to do the line breaking ourselves. An LR-paragraph may contain R-text insertions where TEX needs assistance, and a RL paragraph may contain L-text insertions.

Let us assume the following markup conventions:

- a LR paragraph is just an ordinary TEX paragraph, delimited by an empty line or an explicit `\par` command.
- for special purposes a LR paragraph may also be written `\Lpar {L-text}`.
- a RL paragraph is written as `\Rpar {R-text}`.
- we denote a RL insertion within L-text by the command `\Rinsert {R-text}`, and a LR insertion in R-text is marked `\Linsert {L-text}`,
- we also assume the presence of a command `\Rtext {R-text}` which will put the included text into a `\hbox`, without any line-breaking.

The structure of the paper is as follows: we first describe a simplified model of our processing

[1] The ArabTEX system is in the public domain. It can be downloaded from the author's institution at `ftp.informatik.uni-stuttgart.de`, Internet address (129.69.211.2), login `ftp`, and also from the CTAN server network.

of R-text. We then indicate how L-text insertions within R-text, and R-text insertions within L-text are handled. After discussing the remaining problems and drawbacks with our solution, we speculate on some possible improvements.

## Processing of R-text

For this report we assume the following, simplified model:

- a passage of R-text is a sequence of words in some suitable external notation, separated by single spaces and/or newlines;
- this sequence is split into individual items, and each item is transformed into its graphical representation;
- these representations are aligned from right to left, with interspersed glue, in a line buffer of maximal length `\hsize`. Whenever this buffer gets full, its contents will be aligned by suitable stretching of the glue between, and also within, the individual items, and will be output as a single `\hbox` of width `\hsize`. The last line will be filled up with glue at the left end.

To get an idea of the techniques used the interested reader might want to inspect the following macros. These are a grossly simplified version of the routines actually used, which contain many additional technical details. The effect of some routines not explained should be obvious from their names. Note that we do not read our RL text immediately; depending on the coding for the RL language in question we might e.g. have to adjust some `\catcode`s first.

```
\def \Rpar {% no parameters !
\bgroup \arab@codes \set@arabfont
\setbox\lineb@x \hbox {}\Ritems }

\def \Ritems #1{% read the argument now
\Rwords #1 \Rend \flushlineb@x \egroup }

\def \Rwords #1 #2{% split the sequence
\setbox\wordb@x {\Rword #1}\putwordb@x
\ifx #2\Rend \let \next \relax \else
\def \next {\Rwords #2}\fi \next }

\def \putwordb@x {% deposit one R-word
\setbox0 \copy\lineb@x % save old value
\setbox\lineb@x \hbox {\unhcopy\wordb@x
\arab@space \unhbox\lineb@x }% prepend
\ifdim \wd\lineb@x > \hsize % overflow
\hbox to \hsize {\unhbox0 }% old line
\setbox\lineb@x \box\wordb@x % new word
\fi }

\def \flushlineb@x {% last line
\hbox to \hsize {\hfill \unhbox\lineb@x }}
```

The full version of the analogous macros within ArabTEX allows for an arbitrary number of space tokens between R-words and caters, among others, for indentation, handling of commands within R-text, transliteration, mathematical insertions, and also L-text insertions.

Contrary to the L-text case where TEX does the line-breaking, there is no global optimization of line breaks for R-text. This leads to better results than expected because

- in the RL languages considered, Arabic and Hebrew, words are comparatively short and there is no hyphenation;
- in Arabic, we exploit the fact that also the words are elastic, as some of the letters may stretch.

## L-text within R-text

This case is comparatively straightforward. We proceed as follows:

- We deposit the L-text insertion as a paragraph into a `\vbox` of width `\hsize`. The first line will be shorter than the rest because it later has to fit into the partially filled R-line; this is accomplished by suitable values of the paragraph parameters `\hangindent` and `\hangafter`.
- TEX will break this L-paragraph optimally into lines [Knu84, Chapter 14, p. 94] which will be deposited in the `\vbox` as a sequence of `\hbox`es, with interspersed glue.
- We split the individual LR lines off this `\vbox` and deposit them, `\unhbox`-ed and after deleting any glue at the right end, into the RL line buffer. When the buffer gets full, its contents will be output, and this will happen exactly at the same line breaks as before. Thus the lines, including the first one, will still be optimally spaced horizontally, and the last one will be right adjusted.

We end up with a partially filled RL line buffer containing some LR text, and resume RL processing.

The macros below again show only the main features; for the full version see the appendix.

```
\def \Linsert #1{\dimen@ \wd\lineb@x
% splice Ltext into a open RL paragraph
\setbox\insertb@x \vbox
{\rm \hangindent -\dimen@ \hangafter \m@ne
\vskip \a@vglue \noindent #1\endgraf }%
\a@Lunpack \Ritems }% go on with RL text
```

```
\def \a@Lunpack {% unpack and deposit
% all LR lines from the insertion box
\loop \a@getline \a@spacefalse
\putwordb@x {\unhbox\tempb@x \unskip }%
\ifvbox \insertb@x \repeat }

\def \a@getline {% unpack the next line
% globally to \box\tempb@x
\splittopskip \a@vglue \setbox\tempb@x
\vsplit \insertb@x to \a@splitht
\setbox0 \vbox {\unvbox \tempb@x
\global \setbox\tempb@x \lastbox }}
```

## R-text within L-text

This case is more complicated. We can try to follow the strategy for L-text within R-text with the necessary modifications; but various problems turn up, depending on the mode we are in [Knu84, Chapter 13, p. 85].

**Vertical mode.** If we are in vertical mode, that means we in fact have not yet started a LR paragraph. So the first idea is just to deposit our RL material as a RL paragraph, and we are done. That is, almost; only if a `\par` command follows we may fill up the RL buffer at the left, and deposit the last line. Otherwise we start a new LR paragraph without indentation, and with the contents of the RL buffer at the beginning of the first line, and let TEX go on with LR processing.

At first sight this looks logical, but it is wrong. We are in L-text mode, and thus our new paragraph must be indented at the left, even if there is no preceding L-text, and not at the right like a genuine RL paragraph. So instead we start a new LR paragraph by switching to horizontal mode by `\leavevmode`, and process our RL material as a genuine insertion, as described below.

**Restricted horizontal mode.** If we are in restricted horizontal mode, this means we are within an `\hbox` that can grow arbitrarily wide. We just collect our RL material within a line buffer of unlimited length, and add the contents, `\unhbox`-ed, to the current `\hbox`; and if the box thus gets too wide for the current line, that is the user's problem as an `\hbox` by definition will never be automatically split, and we can do nothing about it.

To find out whether this case applies, and also to handle the case of vertical mode discussed above, we can proceed as follows:

```
\def \Rinsert {% change catcodes and font
\bgroup \arab@codes \set@arabfont
\a@Rinsert }

\def \a@Rinsert #1{% now read RL argument
\leavevmode % hmode if not there already
\ifinner \Rtext {#1}% inside hbox: append
\else \a@Rsplit{#1}% splice into paragraph
\fi \egroup }
```

**Horizontal mode.** When we are in horizontal mode this means we are in the process of building a LR paragraph and want to include some RL material. We again can try to collect the RL lines temporarily within a suitable `\vbox` and add them to the current paragraph; the first RL line has to be made sufficiently short so its contents will fit on the partially completed last line of the current LR paragraph.

Thus our simplified algorithm would run along the following lines:

```
\def \a@Rsplit #1{% not inside a hbox
\a@Ldimen % dimensions of the last line
\a@Rtobox {#1}% pack insertion into \vbox
\a@Runpack }% unpack lines and splice

\def \a@Rtobox #1{% pack into \insertb@x
\setbox \insertb@x \vbox {\hsize \a@Lwidth
\parshape \@ne \a@Lindent \hsize
\setbox\lineb@x {\hskip \a@Llength }%
\Rwords #1 \Rend % get RL material
\box\lineb@x }}% flush last line

\def \a@Runpack {% get and adjust
% all the lines from the insertion box
\loop \a@getline % line to \tempb@x
\unhbox \tempb@x \unskip \break % add line
\ifvbox \insertb@x \repeat
\unpenalty }% no break after the last line
```

Here the routine `\a@Ldimen` has to deliver the dimensions `\a@Llength`, `\a@Lwidth`, `\a@Lindent` of the last LR line. But — there is as yet no such line, so we have to finish the current LR paragraph first, before we can sensibly talk about the length of its last line, and open it up again afterwards.

Unfortunately it is not quite as simple as that. In (external) vertical mode the current paragraph will go to the main vertical list at once, and we can no more get at the last line to find out about its length. In internal vertical mode, that is, if we are within a `\vbox`, the last line would be still accessible, but we cannot find out in time whether that is the case. So we have to resort to a device that always does what we need, that is, display mode [Knu84, Chapter 19], which has some side effects we exploit.

Whenever we interrupt a paragraph by opening a display, the material collected up to this point will be formatted *without exercising the page builder,* and the dimensions of the last line

will be accessible through the internal TEX parameters \predisplaysize, \displaywidth, and \displayindent. So we can get at the information required to format our RL material, but we have to get rid again of the spurious display introduced. We want to handle this differently depending on whether we are inside a \vbox or not, and in order to find out which case applies we have to close the paragraph first by finishing the display and adding a \par command, carefully inserting penalties so the page builder will not interfere. Only now we can find out in which context we are, by the \ifinner test.

We compute the length of the last visible LR line from the value of \predisplaysize, and there are a few cases to distinguish: [Knu84, Chapter 19, p. 188]

- If the value is −\maxdimen, there was no previous line. This cannot happen.
- If the value is \maxdimen, this signifies that the last line has not been typeset at its natural width. Either there was some infinite stretch in it, or the \parfillskip is finite, or the line has been shrunk to \hsize. In any case we adjust \predisplaysize to \hsize.
- Otherwise we have to subtract two ems in the current font.

The result is the amount of horizontal skip by which the overlayed first RL line must start at the left, possibly after a strut and a suitable penalty to prevent TEX from breaking the line at this position.

Now at last we can collect our RL material inside a temporary \vbox, and we split off the first line to get at its vertical dimensions. We need them as \prevdepth by now will have been set to zero because of the empty box within the display, and we have to adjust it so that the newly started paragraph, after skipping back the height of the dummy display plus \parskip, will start again at the correct vertical position.

A first version of \a@Ldimen thus looks as follows:

```
\def \a@Ldimen {% get the dimensions
% of the current LR line
$$\global \a@Llength \predisplaysize
\global \a@Lwidth \displaywidth
\global \a@Lindent \displayindent
\hbox to \a@Lwidth {}% filler
\postdisplaypenalty 10000
$$\endgraf % close the paragraph
\a@Ladjust % update \a@Llength
\ifinner \a@killdisplay \else
\a@skipback \fi }
```

```
\def \a@Ladjust {% correct \a@Llength
\ifdim \a@Llength = \maxdimen
\a@Llength \hsize % flexible line
\else \advance \a@Llength -2em \fi
\advance \a@Llength -\a@Lindent }
```

**Continuing a paragraph within a box.** Let us first discuss the case that we are within a \vbox. This happens rather frequently, e.g. while building up a footnote, a minipage, a parbox, and also while collecting an LR insertion. In this case we take away all the extraneous material by a suitable sequence of \lastbox, \unskip, and \unpenalty operations. We also grab the last LR line, unpack it and delete the \parfillskip glue and the final penalty, and deposit it again as the first line of a new paragraph without indentation. Then we collect the RL material, and deposit it into the current paragraph as in the R-text case above.

There is a complication here, as well as in the case of vertical mode: the new portion of the first line might well have a greater height than the part already processed once, and thus we may not reuse the interline skip already deposited without risking that the distance of the baselines might increase without need. So we have to make the new paragraph respect the depth of the *preceding* line, by explicitly computing it from the height of the current line, the \baselineskip, and the last interline glue deposited. Only after adjusting \prevdepth and skipping back by \parskip we may open the new paragraph to get to the old vertical position again.

```
\def \a@killdisplay {% remove garbage
\unskip \unskip \unpenalty
\setbox0 \lastbox % contents of display
\unskip \unskip \unpenalty
\setbox0 \lastbox % grab last line
\dimen@ \baselineskip
\advance \dimen@ -\ht0
\advance \dimen@ -\lastskip
\prevdepth \dimen@ % new value
\unskip \vskip -\parskip
\noindent \unhbox0 % open last line
\unskip \unskip \unpenalty }% trim it
```

It is interesting to note that in this case, as well as in the case of R-text within L-text, the result looks internally like a single paragraph consisting of a sequence of lines with intervening glue and penalties. It can therefore be taken apart again line by line, as is done e.g. in the EDMAC system [LW90] and also when it is used recursively as an insertion.

**Continuing a paragraph on the main vertical list.** This is the most risky case, and in some instances we may not arrive at a satisfactory result. We can no more take away and open up the last line; we just can try to simulate this by visually overlaying it with another line, starting with a skip over the already deposited LR text, and continuing with RL material. We try hard to get back to exactly the vertical position at which we started, and to do this we need tight control of the dimensions of our invisible display.

We know that depending on the width of the display and the previous line, either the parameter `\abovedisplayskip` or `\abovedisplayshortskip` will be used, and likewise for the skip after the display. We enforce the use of `\abovedisplayskip` by depositing an empty box of height and depth zero and the full line width into the display. We also set `\postdisplaypenalty` to 10000 in order to prevent the page builder that gets control after closing the display and the current paragraph, from breaking the page at this place, before skipping back.

Now in case we are not within a box we skip back explicitly:

```
\def \a@skipback {% to the old position
\vskip -\belowdisplayskip
\vskip -\baselineskip
\vskip -\abovedisplayskip
\vskip -\baselineskip
\vskip -\parskip \noindent
\a@strut \nobreak \hskip \a@Llength }
```

Apart from the fact that we did not yet compensate for the effect that we lost the old value of `\prevdepth` by the box contained in the display, we are at the old vertical and horizontal position again. But our solution still has some severe drawbacks:

- in case the first RL word will not fit on the last LR line, this line will have white space, in fact `\parfillskip`, at the end;
- if exactly one RL word still fits on the last LR line, the new RL line visually overlayed will be severely underfull, and there will be an ugly gap between LR and RL material.

Fortunately we can do much better, however at additional expense.

First we have to make sure that the last LR line, if the first RL word will no more fit on it, is not filled up at the right end with the `\parfillskip` glue but spread out to the full line length. For this purpose we collect the first RL word in a temporary box; and if while doing this we find out that it is the only RL word, we just deposit it and are done with the RL insertion. Otherwise we look at its dimensions and add an empty box with the same width, depth, and height to the still open LR paragraph. If this empty box, after formatting the LR material by entering display mode, lands on a line by itself the previous line will be spread out correctly. Otherwise we know there is room on the current line for the first RL word, and its height will have been taken into account for determining the interline skip, also its depth influences `\prevdepth`. Of course we have to subtract the width of the invisible box from the value of `\a@Llength` computed as above, before collecting and depositing the RL material.

In order to solve the second problem we generalize this idea: we deposit an empty box each for the first and the second RL word, in this order, into the current paragraph, with appropriate glue in between. Now if a line break falls between these boxes, the last visible LR line will have been spread out just enough so that the first RL word will fit snugly; if both boxes get to a new line the last visible line will extend to the right margin; and if both boxes fit on the current line there will be sufficient stretch available in the RL material for reasonable formatting. In all cases we know the vertical dimensions of the last LR line, at least under the assumption that the RL material dominates; this is usually the case, as we routinely deposit a suitable strut. So we can again skip back to the correct vertical position and adjust `\prevdepth` accordingly.

More details are given in the appendix; they are straightforward but tedious.

**Manual tuning.** The solution just given still has a drawback: the two visually overlayed lines are formatted independently, and thus might be stretched by a noticably different amount. The worst case occurs when the second line contains just two RL words whereas the third RL word would nearly fit in the remaining space, so practically its whole width will go into glue. This only can happen within (external) vertical mode, and whereas there is as yet no automatic solution, the user can help by modifying the input text, and while fine tuning the document (s)he would do so probably anyway.

The remedy is actually rather simple: the paragraph in question has to go into a `\vbox` first, and we make the command `\Lpar {L-text}` mentioned above do just that. As usual we have to make sure that this command will not read its argument but expand it, otherwise any `\catcode` changes, e.g. inside a `\verbatim` insertion, would be lost; (we borrowed the technique from the Plain TEX footnote mechanism; LATEX 2.09 [Lam86] has a flaw there!)

A possible alternative would be to deposit more than two empty boxes for the first RL words into the open LR paragraph. As there are still more cases to distinguish than before, we have not implemented this extension due to its sheer complexity, and also its additional dynamical costs. In our experience the same effect, or even better results, can be easily obtained by manual tuning so the extension is probably not worth while.

## Other mode combinations

One of the characteristic features of TEX is that its various modes may be nested recursively. Our extension fits well into this scheme for the following reasons:

- it works recursively, as noted above;
- the transition to R-text is triggered by an explicit command,
- R-text has its own command processing, thus we can handle the mode switching explicitly, and have full control.

The following mode transitions remain to be discussed:

- R-text to (in-line) mathematical mode:
  We start a dummy L-text insertion that is handled as above. Inside it TEX does the normal paragraph processing, including possible line-breaking, and L-text may contain mathematical mode material. At the end of this insertion we get back to R-text mode as described.
- R-text to display mode:
  This can be handled by brute force. We finish the current RL paragraph, let TEX do the processing of the display, and restart R-text afterwards. The display parameters get reasonable values by prepending an invisible dummy LR paragraph.
- display mode to R-text:
  and
- in-line mathematical mode to R-text:
  All longer R-text inserts will be within a `\hbox`, and we already covered this case. Isolated RL-language symbols may be introduced via macros, analogous to Greek symbols.

We have to admit that we have had little experience with these transitions, but in our tests we experienced no severe problems. When testing we assumed, supported by the examples that we have seen, that all mathematical insertions run from left to right, even within R-text documents, and that mathematical symbols are taken from Latin script, with some well-known exceptions handled by TEX.

## Conclusion

We have demonstrated that the problem of bidirectional line breaking can indeed be solved within the context of TEX, but the price is high. Our algorithm is extremely involved, and the fact that it uses a lot of time is usually only concealed by the fact that also our method of processing RL words is, due to the complexity of the Arabic script, rather slow.

On the positive side, our method automatically covers the case where insertions of the two types are recursively nested. However this feature is not needed very often, and as shown in [KM87] its indiscriminate use may lead to texts with a very cryptical structure.

The technique we described is, with some minor modifications, available within ArabTEX version 3.02, and has already been used to process a paper of about 40 pages of mixed English and Hebrew text, also some Arabic, with quite satisfactory results.

## References

[KM87] Donald E. Knuth and Pierre A. MacKay. "Mixing right-to-left texts with left-to-right texts". *TUGboat*, 8(1):14–25, 1987.

[Knu84] Donald E. Knuth. *The TEXbook*, volume A of Computers & Typesetting. Addison-Wesley, Reading, Mass., 1984.

[Lag92a] Klaus Lagally. ArabTEX — Typesetting Arabic with Vowels and Ligatures. In *EuroTEX '92, Proc. 7th European TEX Conference*, pages 153–172, Prague, Czechoslovakia, September 14–18, 1992. See also [Lag92b].

[Lag92b] Klaus Lagally. ArabTEX — Typesetting Arabic with Vowels and Ligatures. Report 1992/07, Universität Stuttgart, Fakultät Informatik, 1992.

[Lag93] Klaus Lagally. ArabTEX, a System for Typesetting Arabic. User Manual Version 3.00. Report 1993/11, Universität Stuttgart, Fakultät Informatik, 1993.

[Lam86] Leslie Lamport. *LATEX, a Document Preparation System*. Addison-Wesley, Reading, Mass., 1986.

[LW90] John Lavagnino and Dominik Wujastyk. An Overview of EDMAC: A `plain` TEX format for critical editions. *TUGboat*, 11(4):623–643, 1990.

## Appendix

In the main part of this report we glossed over many technicalities. For readers interested in some more details, here is the actual code (still somewhat simplified). The meaning of some parameters and macros not described should be obvious.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% external commands:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% these do not read their arguments !

% \Rtext{#1} = RL insertion inside Ltext
\def \Rtext {\protect \a@RL }

% \Ltext{#1} = LR insertion inside Rtext
\def \a@c@Ltext {\unarab@codes \a@Linsert}
% this is called internally

\def \Lpar {%  usage: \Lpar {paragraphs}
% put around one or more paragraphs
% whenever the linebreaking is bad
\dimen@ \prevdepth
\setbox0 \vbox \bgroup \prevdepth \dimen@
\def \par{\egroup \endgraf \Lpar x}% dummy
\aftergroup \L@par \let \next=}

\def \L@par {\unvbox0 }% called internally

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% implementation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% internal declarations

\newdimen \a@Llength
\newdimen \a@Lwidth
\newdimen \a@Lindent
\newdimen \a@splitht

\newbox \insertb@x
\newbox \a@Rboxi
\newbox \a@Rboxii
\newbox \a@Rdummyi
\newbox \a@Rdummyii

\newif \ifR@split

\def \ins@skip {\hskip \z@ plus 0.1em }
% hglue before and after an insertion

\def \a@vglue {\z@ plus 2ex }
% vglue at top of the insertion \vbox

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% macros for LR insertions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\def \a@Linsert #1{% splice Ltext
% into a open RL paragraph
\a@spacetrue \putwordb@x {\ins@skip }%
\dimen@ \wd \lineb@x % current RL line
\setbox \insertb@x \vbox
{\rm \hangindent -\dimen@ \hangafter \m@ne
\parskip \z@ \rightskip \z@ plus .001fil
\vskip \a@vglue \noindent #1\endgraf }%
\a@Lunpack % get the LR lines again
\putwordb@x {\ins@skip }% after insertion
\a@spacetrue \arab@codes \test@token }
% go on with RL text

\def \a@Lunpack {% unpack and deposit
% all LR lines from the insertion box
\a@splitht 3.5ex
\loop \a@getline \a@spacefalse
\putwordb@x {\unhbox \tempb@x \unskip }%
\ifvbox \insertb@x \repeat }

\def \a@getline {% unpack the next line
% globally to \box \tempb@x
\splittopskip \a@vglue \setbox \tempb@x
\vsplit \insertb@x to \a@splitht
\setbox0 \vbox {\unvbox \tempb@x
\global \setbox \tempb@x \lastbox }}


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% macros for RL insertions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\def \a@RL {% change catcodes and font
\bgroup \arab@codes \set@arabfont
\a@Rinsert }

\def \a@Rinsert #1{% now read RL argument
\leavevmode % hmode if not there already
\ifinner \a@Rtext #1>% inside hbox: append
\else \a@Rsplit{#1}% splice into paragraph
\fi \egroup }

\def \a@Rsplit #1{% splice RL lines into
% an open LR paragraph
\a@Rdimen {#1}% length of first 2 RL words
\ifdim \wd\a@Rboxii = \z@ % only one !
\unhbox\a@Rboxi % done with the insertion
\else \a@Ldimen % dimensions of last line
\a@Rtobox {#1}% pack insertion into \vbox
\a@Runpack \fi }% unpack lines and splice
```

```
\def \a@Rdimen #1{% get the dimensions
% of the first two RL words
\a@Rfirst #1 \to \a@wordi \a@Rrest
\expandafter \a@Rwordtobox \a@wordi
\to \a@Rboxi \a@Rdummyi
% put into box and make dummy box
\expandafter \ifx \expandafter \relax
\a@Rrest \relax \def \a@wordii {}%
% no second RL word exists
\else \expandafter \a@Rfirst \a@Rrest
\to \a@wordii \a@Rrest \fi % get 2. word
\expandafter \a@Rwordtobox \a@wordii
\to \a@Rboxii \a@Rdummyii }

\def \a@Rfirst #1 #2\to #3#4{% split off
% first RL word from RL sequence
\ifx \relax #1\relax \a@Rfirst #2\to #3#4%
\else \def #3{#1}\def #4{#2}\fi }

\def \a@Rwordtobox #1\to #2#3{% pack word
% into \box #2 and make dummy \box #3
{\setbox#2\hbox {\a@strut \a@Rtext {#1}}%
\setbox#3\hbox to\wd#2{\hfill }% dummy box
\ht#3\ht#2\dp#3\dp#2}% same dimensions

\def \a@Rtobox #1{% pack into \insertb@x
\setbox \insertb@x \vbox {\hsize \a@Lwidth
\parshape \@ne \a@Lindent \hsize
\vskip \a@vglue \hbadness \@M % no warning
\putlineb@x {\a@strut \hskip \a@Llength }%
\a@spacefalse \Rwords #1 \Rend
% process the RL material using \lineb@x
\unskip \unskip \vskip \a@vglue
\box\lineb@x }}

\def \a@Runpack {\ins@skip
% unpack all RL lines from the insertion
\a@splitht 1.4\baselineskip
\loop \a@getline % split off and adjust
\unhbox \tempb@x \unskip \break
\ifvbox \insertb@x \repeat
\unpenalty \penalty 5000 \ins@skip }

\def \a@Ldimen {% close paragraph and get
% the dimensions of the current LR line
\leavevmode % cover the case of \vmode
\copy \a@Rdummyi \arab@space % glue
\copy \a@Rdummyii % may split the line !
\lineskiplimit \lineskip % to make sure
$$\global \a@Llength \predisplaysize
\global \a@Lwidth \displaywidth
\global \a@Lindent \displayindent
\hbox to \a@Lwidth {\hfill }% filler
\postdisplaypenalty \@M % no page break !
$$ \endgraf \a@Ladjust % update \a@Llength
\ifinner \a@killdisplay % within a \vbox ?
\else \a@skipback \fi }

\def \a@Ladjust {% correct \a@Llength
\ifdim \a@Llength = \maxdimen
\a@Llength \hsize % was a flexible line
\else \advance \a@Llength -2em \fi
\advance \a@Llength -\a@Lindent
\advance \a@Llength -\wd\a@Rdummyii % box2
\ifdim \a@Llength = \z@ % line was split
\R@splittrue \a@Llength \hsize % at margin
\else \setbox0 \hbox {\arab@space}% space
\R@splitfalse \advance \a@Llength -\wd0
\fi \advance \a@Llength -\wd\a@Rdummyi }

\def \a@killdisplay {% remove any garbage
% produced by the dummy display
\unskip \unskip \unpenalty
\setbox0 \lastbox % contents of display
\ifR@split \unskip \unskip \unpenalty
\setbox0 \lastbox \fi% last line was split
\unskip \unskip \unpenalty
\setbox0 \lastbox % last visible LR line
\dimen0 \baselineskip % compute \prevdepth
\advance \dimen@ -\ht0
\advance \dimen@ -\lastskip
\prevdepth \dimen@ % new \prevdepth value
\unskip \vskip -\parskip
\noindent \unhbox0 % open last LR line
\unskip \unskip \unpenalty
\setbox0 \lastbox % delete dummy box
\ifR@split \else \unskip % was space
\setbox0 \lastbox \fi }% second dummy

\def \a@skipback {% get back over the
% display to the old vertical position
\vskip -\belowdisplayskip
\vskip -\baselineskip
\vskip -\abovedisplayskip
\vskip -\baselineskip \vskip -\parskip
\ifR@split % line was split
\vskip -\ht\a@Rdummyii % second dummy box
\vskip -\dp\a@Rdummyii \vskip -\lineskip
\fi % now open next RL line
\noindent \a@strut \nobreak
\hskip \a@Llength }

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% end of macros
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```