The breqn package

Authors: Michael J. Downes, Morten Høgholm
Maintained by Morten Høgholm, Will Robertson
Feedback: `https://github.com/wspr/breqn/issues`

breqn bundle: 2018/09/14 0.98f

Abstract

The breqn package facilitates automatic line-breaking of displayed math expressions.

Contents

Part I
User's guide

1 A bit of history

Originally breqn, flexisym, and mathstyle were created by Michael J. Downes from the American Mathematical Society during the 1990's up to late 2002. Sadly—and much to the shock of the TEX world—Michael passed away in early 2003 at the age of only 44.

The American Mathematical Society kindly allowed Morten Høgholm to assume maintainership of this part of his work and we wish to express our gratitude to them and to Barbara Beeton in particular for providing the files needed.

MH brought Michael's work to a wider audience, thereby allowing users to create more *masterpieces of the publishing art* as we think he would have wanted.

Following the July 2008 breqn release, breqn was left in the hands of a maintenance team, while MH moved on with other projects.

2 Package loading

The recommended way of loading the breqn package is to load it *after* other packages dealing with math, i.e.,, after amsmath, amssymb, or packages such as mathpazo or mathptmx.

The flexisym package (described in section 10 on page 10) is required by breqn and ensures the math symbols are set up correctly. By default breqn loads it with support for Computer Modern but if you use a different math package requiring slightly different definitions, it must be loaded before breqn. Below is an example of how you enable breqn to work with the widely used mathpazo package.

```
\usepackage{mathpazo}
\usepackage[mathpazo]{flexisym}
\usepackage{breqn}
```

Currently, the packages mathpazo and mathptmx are supported. Despair not: Chances are that the package will work using the default settings. If you find that a particular math font package doesn't work then please see implementation in `flexisym.dtx` for how to create a support file—it is easier than one might think. Contributions welcome.

The documentation for the package was formerly found in `breqndoc`. It has now been added to this implementation file. Below follows the contents of the original breqn documentation. Not all details hold anymore but I have prioritized fixing the package.

3 Introduction

The breqn package for LaTeX provides solutions to a number of common difficulties in writing displayed equations and getting high-quality output. For example, it is a well-known inconvenience that if an equation must be broken into more than one line, `\left` ... `\right` constructs cannot span lines. The breqn package makes them work as one would expect whether or not there is an intervening line break.

The single most ambitious goal of the breqn package, however, is to support automatic linebreaking of displayed equations. Such linebreaking cannot be done without substantial changes under the hood in the way math formulas are processed. For this reason, especially in the alpha release, users should proceed with care and keep an eye out for unexpected glitches or side effects.

4 Principal features

The principal features of the breqn package are:

semantically oriented structure The way in which compound displayed formulas are subdivided matches the logical structure more closely than, say, the standard `eqnarray` environment. Separate equations in a group of equations are written as separate environments instead of being bounded merely by `\\` commands. Among other things, this clears up a common problem of wrong math symbol spacing at the beginning of continuation lines. It also makes it possible to specify different vertical space values for the space between lines of a long, broken equation and the space between separate equations in a group of equations.

automatic line breaking Overlong equations will be broken automatically to the prevailing column width, and continuation lines will be indented following standard conventions.

line breaks within delimiters Line breaks within `\left` ... `\right` delimiters work in a natural way. Line breaks can be forbidden below a given depth of delimiter nesting through a package option.

mixed math and text Display equations that contain mixed math and text, or even text only, are handled naturally by means of a `dseries` environment that starts out in text mode instead of math mode.

ending punctuation The punctuation at the end of a displayed equation can be handled in a natural way that makes it easier to promote or demote formulas from/to inline math, and to apply special effects such as adding space before the punctuation.

flexible numbering Equation numbering is handled in a natural way, with all the flexibility of the amsmath package and with no need for a special `\nonumber` command.

special effects It is easy to apply special effects to individual displays, e.g., changing the type size or adding a frame.

using available space Horizontal shrink is made use of whenever feasible. With most other equation macros it is frozen when it occurs between `\left` ... `\right` delimiters, or in any sort of multiline structure, so that some expressions require two lines that would otherwise fit on one.

high-quality spacing The `\abovedisplayshortskip` is used when applicable (other equation macros fail to apply it in equations of more than one line).

abbreviations Unlike the amsmath equation environments, the breqn environments can be called through user-defined abbreviations such as `\beq` ... `\eeq`.

5 Shortcomings of the package

The principal known deficiencies of the breqn package are:

5.1 Incompatibilities

As it pushes the envelope of what is possible within the context of LaTeX 2_ε, the breqn package will tend to break other packages when used in combination with them, or to fail itself, when there are any areas of internal overlap; successful use may in some cases depend on package loading order.

5.2 Indention of delimited fragments

When line breaks within delimiters are involved, the automatic indention of continuation lines is likely to be unsatisfactory and need manual adjustment. I don't see any easy way to provide a general solution for this, though I have some ideas on how to attain partial improvements.

5.3 Math symbol subversion

In order for automatic line breaking to work, the operation of all the math symbols of class 2, 3, 4, and 5 must be altered (relations, binary operators, opening delimiters, closing delimiters). This is done by an auxiliary package flexisym. As long as you stick to the advertised LaTeX interface for defining math symbols (`\DeclareMathSymbol`), things should work OK most of the time. Any more complex math symbol setup is quite likely to quarrel with the flexisym package. See Section 10 on page 10 for further information.

5.4 Subscripts and superscripts

Because of the changes to math symbols of class 2–5, writing certain combinations such as `^+` or `_\pm` or `^\geq` without braces would lead to error messages; (The problem described here already exists in standard LaTeX to a lesser extent, as you may know if you ever tried `^\neq` or `^\cong`; and indeed there are no examples in the LaTeX book to indicate any sanction for omitting braces around a subscript or superscript.)

The flexisym package therefore calls, as of version 0.92, another package called mathstyle which turns `^` and `_` into active characters. This is something that I believe is desirable in any case, in the long run, because having a proper mathstyle variable eliminates some enormous burdens that affect almost any nontrivial math macros, as well as many other things where the connection is not immediately obvious, e.g., the LaTeX facilities for loading fonts on demand.

Not that this doesn't introduce new and interesting problems of its own—for example, you don't want to put usepackage statements after flexisym for any package that refers to, e.g., `^^J` or `^^M` internally (too bad that the LaTeX package loading code does not include automatic defenses to ensure normal catcodes in the interior of a package; but it only handles the `@` character).

But I took a random AMS journal article, with normal end-user kind of LaTeX writing, did some straightforward substitutions to change all the equations into dmath environments, and ran it with active math sub/sup: everything worked OK. This suggests to me that it can work in the real world, without an impossible amount of compatibility work.

6 Incomplete

In addition, in the **alpha release [1997/10/30]** the following gaps remain to be filled in:

documentation The documentation could use amplification, especially more illustrations, and I have undoubtedly overlooked more than a few errors.

group alignment The algorithm for doing alignment of mathrel symbols across equations in a `dgroup` environment needs work. Currently the standard and `noalign` alternatives produce the same output.

single group number When a `dgroup` has a group number and the individual equations are unnumbered, the handling and placement of the group number aren't right.

group frame Framing a group doesn't work, you might be able to get frames on the individual equations at best.

group brace The `brace` option for `dgroup` is intended to produce a large brace encompassing the whole group. This hasn't been implemented yet.

darray environment The `darray` environment is unfinished.

dseries environment The syntax and usage for the `dseries` environment are in doubt and may change.

failure arrangements When none of the line-breaking passes for a `dmath` environment succeeds—i.e., at least one line is overfull—the final arrangement is usually rather poor. A better fall-back arrangement in the failure case is needed.

7 Package options

Many of the package options for the breqn package are the same as options of the dmath or dgroup environments, and some of them require an argument, which is something that cannot be done through the normal package option mechanism. Therefore most of the breqn package options are designed to be set with a \setkeys command after the package is loaded. For example, to load the package and set the maximum delimiter nesting depth for line breaks to 1:

```
\usepackage{breqn}
\setkeys{breqn}{breakdepth={1}}
```

See the discussion of environment options, Section 9 on page 8, for more information.

Debugging information is no longer available as a package option. Instead, the tracing information has been added in a fashion so that it can be enabled as a docstrip option:

```
\generate{\file{breqn.sty}{\from{breqn.dtx}{package,trace}}}
```

8 Environments and commands

8.1 Environments

All of the following environments take an optional argument for applying local effects such as changing the typesize or adding a frame to an individual equation.

dmath Like equation but supports line breaking and variant numbers.

dmath* Unnumbered; like displaymath but supports line breaking

dseries Like equation but starts out in text mode; intended for series of mathematical expressions of the form 'A, B, and C'. As a special feature, if you use

```
\begin{math} ... \end{math}
```

for each expression in the series, a suitable amount of inter-expression space will be automatically added. This is a small step in the direction of facilitating conversion of display math to inline math, and vice versa: If you write a display as

```
\begin{dseries}
\begin{math}A\end{math},
\begin{math}B\end{math},
and
\begin{math}C\end{math}.
\end{dseries}
```

then conversion to inline form is simply a matter of removing the `\begin{dseries}` and `\end{dseries}` lines; the contents of the display need no alterations.

It would be nice to provide the same feature for `$` notation but there is no easy way to do that because the `$` function has no entry point to allow changing what happens before math mode is entered. Making it work would therefore require turning `$` into an active character, something that I hesitate to do in a LaTeX 2_ε context.

`dseries*` Unnumbered variant of `dseries`

`dgroup` Like the `align` environment of `amsmath`, but with each constituent equation wrapped in a `dmath`, `dmath*`, `dseries`, or `dseries*` environment instead of being separated by `\\`. The equations are numbered with a group number. When the constituent environments are the numbered forms (`dmath` or `dseries`) they automatically switch to 'subequations'-style numbering, i.e., something like (3a), (3b), (3c), ..., depending on the current form of non-grouped equation numbers. See also `dgroup*`.

`dgroup*` Unnumbered variant of `dgroup`. If the constituent environments are the numbered forms, they get normal individual equation numbers, i.e., something like (3), (4), (5), ...

`darray` Similar to `eqnarray` but with an argument like `array` for giving column specs. Automatic line breaking is not done here.

`darray*` Unnumbered variant of `darray`, rather like `array` except in using `\displaystyle` for all column entries.

`dsuspend` Suspend the current display in order to print some text, without loss of the alignment. There is also a command form of the same thing, `\intertext`.

8.2 Commands

The commands provided by the `breqn` package are:

`\condition` This command is used for a part of a display which functions as a condition on the main assertion. For example:

```
\begin{dmath}
f(x)=\frac{1}{x} \condition{for $x\neq 0$}
\end{dmath}.
```

$$f(x) = \frac{1}{x}, \quad \text{for } x \neq 0\,. \tag{1}$$

The `\condition` command automatically switches to text mode (so that interword spaces function the way they should), puts in a comma, and adds an appropriate amount of space. To facilitate promotion/demotion of formulas, `\condition` "does the right thing" if used outside of display math.

To substitute a different punctuation mark instead of the default comma, supply it as an optional argument for the `\condition` command:

```
\condition[;]{...}
```

(Thus, to get no punctuation: `\condition[]{...}`.)

For conditions that contain no text, you can use the starred form of the command, which means to stay in math mode:

```
\begin{dmath}
f(x)=\frac{1}{x} \condition*{x\neq 0}
\end{dmath}.

If your material contains a lot of conditions like these, you might like
to define shorter abbreviations, e.g.,
\begin{verbatim}
\newcommand{\mc}{\condition*}% math condition
\newcommand{\tc}{\condition}%  text condition
```

But the breqn package refrains from predefining such abbreviations in order that they may be left to the individual author's taste.

`\hiderel` In a compound equation it is sometimes desired to use a later relation symbol as the alignment point, rather than the first one. To do this, mark all the relation symbols up to the desired one with `\hiderel`:

```
T(n) \hiderel{\leq} T(2^n) \leq c(3^n - 2^n) ...
```

9 Various environment options

The following options are recognized for the `dmath`, `dgroup`, `darray`, and `dseries` environments; some of the options do not make sense for all of the environments, but if an option is used where not applicable it is silently ignored rather than treated as an error.

```
\begin{dmath}[style={\small}]
\begin{dmath}[number={BV}]
\begin{dmath}[labelprefix={eq:}]
\begin{dmath}[label={xyz}]
\begin{dmath}[indentstep={2em}]
\begin{dmath}[compact]
\begin{dmath}[spread={1pt}]
\begin{dmath}[frame]
\begin{dmath}[frame={1pt},framesep={2pt}]
\begin{dmath}[background={red}]
\begin{dmath}[color={purple}]
\begin{dmath}[breakdepth={0}]
```

Use the `style` option to change the type size of an individual equation. This option can also serve as a catch-all option for altering the equation style in other ways; the contents are simply executed directly within the context of the equation.

Use the `number` option if you want the number for a particular equation to fall outside of the usual sequence. If this option is used the equation counter is not incremented. If for some reason you need to increment the counter and change the number at the same time, use the `style` option in addition to the `number` option:

```
style={\refstepcounter{equation}}
```

Use of the normal `\label` command instead of the `label` option works, I think, most of the time (untested). `labelprefix` prepends its argument to the label (only useful as a global option, really), and must be called before `label`.

Use the `indentstep` option to specify something other than the default amount for the indention of relation symbols. The default is 8pt.

Use the `compact` option in compound equations to inhibit line breaks at relation symbols. By default a line break will be taken before each relation symbol except the first one. With the `compact` option LaTeX will try to fit as much material as possible on each line, but breaks at relation symbols will still be preferred over breaks at binary operator symbols.

Use the `spread` option to increase (or decrease) the amount of interline space in an equation. See the example given above.

Use the `frame` option to produce a frame around the body of the equation. The thickness of the frame can optionally be specified by giving it as an argument of the option. The default thickness is `\fboxrule`.

Use the `framesep` option to change the amount of space separating the frame from what it encloses. The default space is `\fboxsep`.

Use the `background` option to produce a colored background for the equation body. The breqn package doesn't automatically load the color package, so this option won't work unless you remember to load the color package yourself.

Use the `color` option to specify a different color for the contents of the equation. Like the `background` option, this doesn't work if you forgot to load the color package.

Use the `breakdepth` option to change the level of delimiter nesting to which line breaks are allowed. To prohibit line breaks within delimiters, set this to 0:

```
\begin{dmath}[breakdepth={0}]
```

The default value for breakdepth is 2. Even when breaks are allowed inside delimiters, they are marked as less desirable than breaks outside delimiters. Most of the time a break will not be taken within delimiters until the alternatives have been exhausted.

Options for the `dgroup` environment: all of the above, and also

```
\begin{dgroup}[noalign]
\begin{dgroup}[brace]
```

By default the equations in a `dgroup` are mutually aligned on their relation symbols ($=$, $<$, $\geq$, and the like). With the `noalign` option each equation is placed individually without reference to the others.

The `brace` option means to place a large brace encompassing the whole group on the same side as the equation number.

Options for the `darray` environment: all of the above (where sensible), and also

```
\begin{darray}[cols={lcr@{\hspace{2em}}lcr}]
```

The value of the `cols` option for the darray environment should be a series of column specs as for the `array` environment, with the following differences:

- For l, c, and r what you get is not text, but math, and displaystyle math at that. To get text you must use a 'p' column specifier, or put an `\mbox` in each of the individual cells.
- Vertical rules don't connect across lines.

10 The flexisym package

The flexisym package does some radical changes in the setup for math symbols to allow their definitions to change dynamically throughout a document. The breqn package uses this to make symbols of classes 2, 3, 4, 5 run special functions inside an environment such as `dmath` that provide the necessary support for automatic line breaking.

The method used to effect these changes is to change the definitions of `\DeclareMathSymbol` and `\DeclareMathDelimiter`, and then re-execute the standard set of LaTeX math symbol definitions. Consequently, additional mathrel and mathbin symbols defined by other packages will get proper line-breaking behavior if the other package is loaded after the flexisym package and the symbols are defined through the standard interface.

11 Caution! Warning!

Things to keep in mind when writing documents with the breqn package:

- The notation $:=$ must be written with the command `\coloneq`. Otherwise the $:$ and the $=$ will be treated as two separate relation symbols with an "empty RHS" between them, and they will be printed on separate lines.
- Watch out for constructions like `^+` where a single binary operator or binary relation symbol is subscripted or superscripted. When the breqn or flexisym package is used, braces are mandatory in such constructions: `^{+}`. This applies for both display and in-line math.
- If you want LaTeX to make intelligent decisions about line breaks when vert bars are involved, use proper pairing versions of the vert-bar symbols according to context: `\lvert n\rvert` instead of `|n|`. With the nondirectional `|` there is no way for LaTeX to reliably deduce which potential breakpoints are inside delimiters (more highly discouraged) and which are not.

- If you use the `german` package or some other package that turns double quote `"` into a special character, you may encounter some problems with named math symbols of type mathbin, mathrel, mathopen, or mathclose in moving arguments. For example, `\leq` in a section title will be written to the `.aux` file as something like `\mathchar "3214`. This situation probably ought to be improved, but for now use `\protect`.
- Watch out for the `[` character at the beginning of a `dmath` or similar environment, if it is supposed to be interpreted as mathematical content rather than the start of the environment's optional argument.

  This is OK:

  ```
  \begin{dmath}
  [\lambda,1]...
  \end{dmath}
  ```

  This will not work as expected:

  ```
  \begin{dmath}[\lambda,1]...\end{dmath}
  ```

- Watch out for unpaired delimiter symbols (in display math only):

  ```
  ( ) [ ] \langle \rangle \{ \} \lvert \rvert ...
  ```

  If an open delimiter is used without a close delimiter, or vice versa, it is normally harmless but may adversely affect line breaking. This is only for symbols that have a natural left or right directionality. Unpaired `\vert` and so on are fine.

  When a null delimiter is used as the other member of the pair (`\left.` or `\right.`) this warning doesn't apply.

- If you inadvertently apply `\left` or `\right` to something that is not a delimiter, the error messages are likely to be a bit more confusing than usual. The normal LaTeX response to an error such as

  ```
  \left +
  ```

  is an immediate message

  ```
  ! Missing delimiter (. inserted).
  ```

  When the `breqn` package is in use, LaTeX will fail to realize anything is wrong until it hits the end of the math formula, or a closing delimiter without a matching opening delimiter, and then the first message is an apparently pointless

  ```
  ! Missing \endgroup inserted.
  ```

12 Examples

Knuth, SNA p74

Example 1

```
Replace $j$ by $h-j$ and by $k-j$ in these sums to get [cf.~(26)]
\begin{dmath}[label={sna74}]
\frac{1}{6} \left(\sigma(k,h,0) +\frac{3(h-1)}{h}\right)
  +\frac{1}{6} \left(\sigma(h,k,0) +\frac{3(k-1)}{k}\right)
=\frac{1}{6} \left(\frac{h}{k} +\frac{k}{h} +\frac{1}{hk}\right)
  +\frac{1}{2} -\frac{1}{2h} -\frac{1}{2k},
\end{dmath}
which is equivalent to the desired result.
```

Replace j by $h-j$ and by $k-j$ in these sums to get [cf. (26)]

$$\frac{1}{6}\left(\sigma(k,h,0)+\frac{3(h-1)}{h}\right)+\frac{1}{6}\left(\sigma(h,k,0)+\frac{3(k-1)}{k}\right)=\frac{1}{6}\left(\frac{h}{k}+\frac{k}{h}+\frac{1}{hk}\right)+\frac{1}{2}-\frac{1}{2h}-\frac{1}{2k}, \quad (12.2)$$

which is equivalent to the desired result.

Knuth, SNA 4.6.2, p387

Example 2

```
\newcommand\mx[1]{\begin{math}#1\end{math}}% math expression
%
Now every column which has no circled entry is completely zero;
so when $k=6$ and $k=7$ the algorithm outputs two more vectors,
namely
\begin{dseries}[frame]
\mx{v^{[2]} =(0,5,5,0,9,5,1,0)},
\mx{v^{[3]} =(0,9,11,9,10,12,0,1)}.
\end{dseries}
From the form of the matrix $A$ after $k=5$, it is evident that
these vectors satisfy the equation $vA =(0,\dotsc,0)$.
```

math expression

Now every column which has no circled entry is completely zero; so when $k=6$ and $k=7$ the algorithm outputs two more vectors, namely

$$\boxed{v^{[2]}=(0,5,5,0,9,5,1,0)\,, \quad v^{[3]}=(0,9,11,9,10,12,0,1)\,.} \quad (12.3)$$

From the form of the matrix A after $k=5$, it is evident that these vectors satisfy the equation $vA=(0,\dots,0)$.

Example 3

```
\begin{dmath*}
T(n) \hiderel{\leq} T(2^{\lceil\lg n\rceil})
  \leq c(3^{\lceil\lg n\rceil}
    -2^{\lceil\lg n\rceil})
  <3c\cdot3^{\lg n}
  =3c\,n^{\lg3}
\end{dmath*}.
```

$$
\begin{aligned}
T(n) \leq T(2^{\lceil\lg n\rceil}) &\leq c(3^{\lceil\lg n\rceil} - 2^{\lceil\lg n\rceil}) \\
&< 3c\cdot3^{\lg n} \\
&= 3c\,n^{\lg3} .
\end{aligned}
$$

Example 4

```
The reduced minimal Gr\"obner basis for $I^q_3$ consists of
\begin{dgroup*}
\begin{dmath*}
H_1^3 = x_1 + x_2 + x_3
\end{dmath*},
\begin{dmath*}
H_2^2 = x_1^2 + x_1 x_2 + x_2^2 - q_1 - q_2
\end{dmath*},
\begin{dsuspend}
and
\end{dsuspend}
\begin{dmath*}
H_3^1 = x_1^3 - 2x_1 q_1 - x_2 q_1
\end{dmath*}.
\end{dgroup*}
```

The reduced minimal Gröbner basis for I^q_3 consists of

$$
\begin{aligned}
H_1^3 &= x_1 + x_2 + x_3 , \\
H_2^2 &= x_1^2 + x_1 x_2 + x_2^2 - q_1 - q_2 ,
\end{aligned}
$$

and

$$H_3^1 = x_1^3 - 2x_1 q_1 - x_2 q_1 .$$

13 Technical notes on tag placement

The method used by the breqn package to place the equation number is rather more complicated than you might think, and the whole reason is to allow the number to stay properly centered on the total height even when the height fluctuates due to stretching or shrinking of the page.

Consider the following equation:

$$N_0 \simeq \left(\frac{\nu}{\|u\|_{H^i}}\right) |I|^{-1/2} \tag{3.15}$$

It will have only one line, if the column width is not too narrow.

Scrutinizing the vertical list will shed light on some of the basic properties shared by all breqn equations. After that we will look at what would happen if two or more lines were needed. The numbers added on the left in the following `\showlists` output mark the points of interest.

```
[1] \penalty 10000
    \glue(\abovedisplayskip) 0.0
    \penalty 10000
    \glue(\belowdisplayskip) 0.0
[2] \glue 4.0 plus 4.0
    \glue(\lineskip) 1.0
[3] \vbox(16.53902+0.0)x0.0, glue set 16.53902fil
    .\glue 0.0 plus 1.0fil minus 1.0fil
    \penalty 10000
[4] \glue -8.51945
[5] \hbox(7.5+2.5)x25.55563
    .\OT1/cmr/m/n/10 (
    .\OT1/cmr/m/n/10 3
    .\OT1/cmr/m/n/10 .
    .\OT1/cmr/m/n/10 1
    .\OT1/cmr/m/n/10 5
    .\kern 0.0
    .\OT1/cmr/m/n/10 )
    \penalty 10000
[6] \glue(\parskip) -18.01956
[7] \hbox(16.53902+9.50012)x360.0, glue set 1.78647
```

1. These four lines are a hidden display structure from TEX's primitive `$$` mechanism. It is used only to get the value of `\predisplaysize` so that we can later calculate by hand whether to use the short display skips or the regular ones. (The reason that we have to do it by hand traces back to the fact that TEX 3.x does not allow unhboxing in math mode.) The penalties come from `\predisplaypenalty` and `\postdisplaypenalty`, which were locally set to 10000 to ensure there would be no unintended page breaks at these glue nodes.

2. These two glue nodes are the ones that would normally have been produced at the top of a display; the first one is the above-display skip node (though we had to put it in by hand with `\vskip`) and the second one is the usual baselineskip/lineskip node.

3. This is a dummy copy of the equation's first line, which is thrown in here to get the proper value of baselineskip (or lineskip in this case). Why do we need this? Because this ensures that we get the top spacing right before we fiddle with the glue nodes surrounding the equation number. And if the equation has a frame, this box is a good place to add it from.

4. This is a special glue node that brings us to the right vertical position for adding the equation number. Its value is calculated from the variables that you would expect, given the presence of the dummy first line above the num- ber: starting position of the equation, height of first line, total height of equation body. If the equation body had more than one line, with stretchable glue between the lines, half of the stretch would be added in this glue node.

5. The hbox containing the equation number.

6. Backspace to bring the equation body to the right starting point. We use `\parskip` to put this glue in place because we're going to get a parskip node here in any case when we add the equation body with (in essence). If we didn't do this we'd get two glue nodes instead of one, to no purpose.

   `\ \unhbox\EQ@box`.

7. And lastly we see here the first line of the equation body, which appears to have height 16.5pt and depth 9.5pt.

For comparison, the vertical list produced from the above equation in standard LaTeX would look like this, if the same values of columnwidth and abovedisplayskip are used:

```
[1] \penalty 10000
[2] \glue(\abovedisplayskip) 4.0 plus 4.0
    \glue(\lineskip) 1.0
    \hbox(16.53902+9.50012)x232.94844
[3] .\hbox(7.5+2.5)x25.55563
    ..\hbox(7.5+2.5)x25.55563
    ...\OT1/cmr/m/n/10 (
    ...\OT1/cmr/m/n/10 3
    ...\OT1/cmr/m/n/10 .
    ...\OT1/cmr/m/n/10 1
    ...\OT1/cmr/m/n/10 5
    ...\kern 0.0
    ...\OT1/cmr/m/n/10 )
    .\kern101.49591
[4] .\hbox(16.53902+9.50012)x105.8969
    ...
[5] \penalty 0
[6] \glue(\belowdisplayskip) 4.0 plus 4.0
    \glue(\lineskip) 1.0
    \hbox(6.94444+1.94444)x345.0, glue set 62.1106fil
```

1. `\predisplaypenalty`

2. `\abovedisplayskip`

3. equation number box

4. equation body

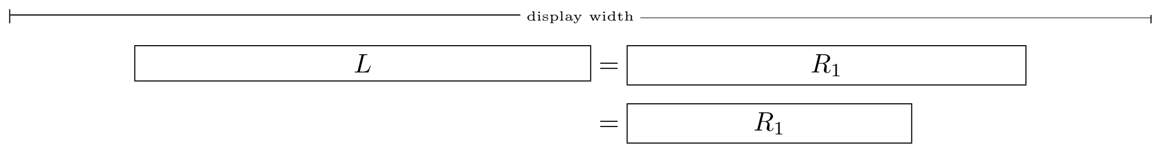
5. \postdisplaypenaltly
6. \belowdisplayskip

14 Technical notes on Equation Layouts

MJD [1998/12/28]

14.1 Misc examples

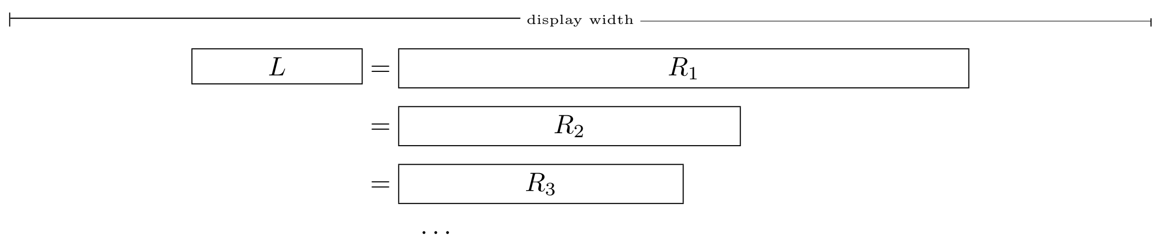
Let us consider which of these have 50% or more of wasted whitespace *within the bounding box of the visible material*.



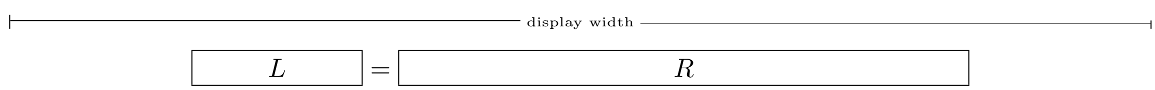
14.2 Ladder and step layouts

14.2.1 Straight ladder layout

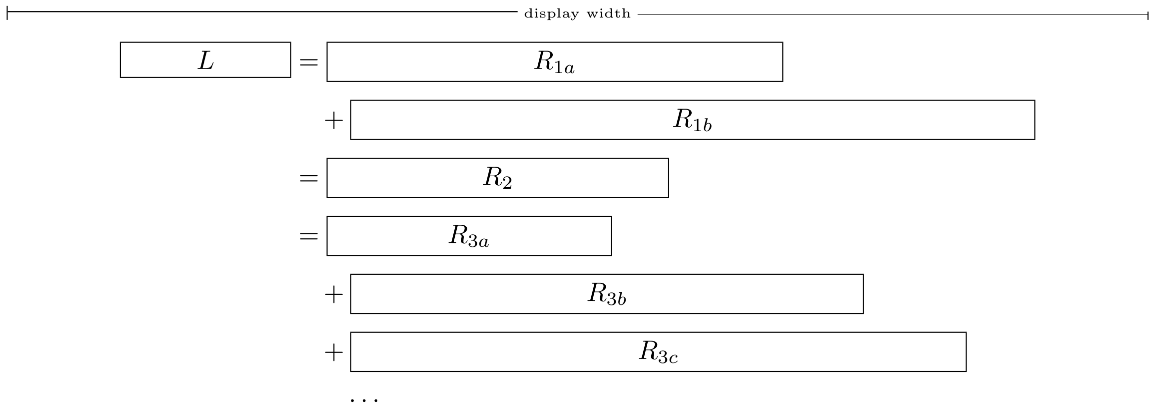
This is distinguished by a relatively short LHS and one or more RHS's of any length.



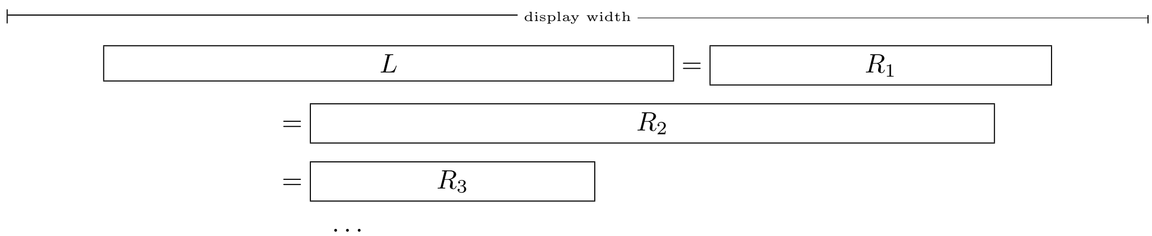
The simplest kind of equation that fits on one line and has only one RHS may be viewed as a trivial subcase of the straight ladder layout:



If some of the RHS's are too wide to fit on a single line they may be broken at binary operator symbols such as plus or minus. This is still classified as a straight ladder layout if none of the fragments intrude into the LHS column, because the underlying parshape is the same.

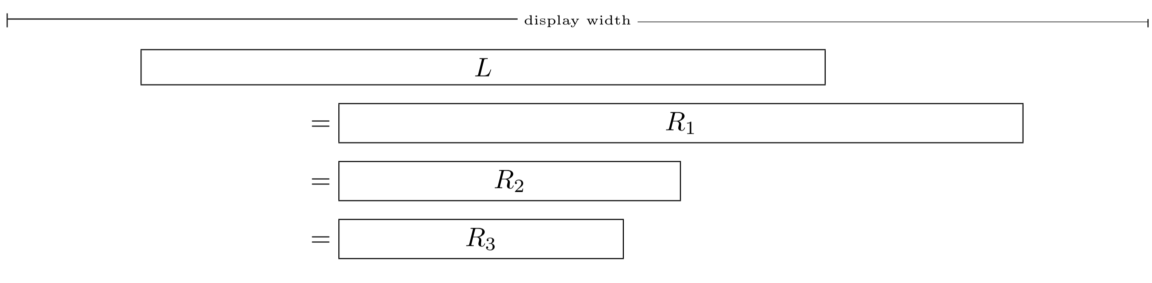


14.2.2 Skew ladder layout



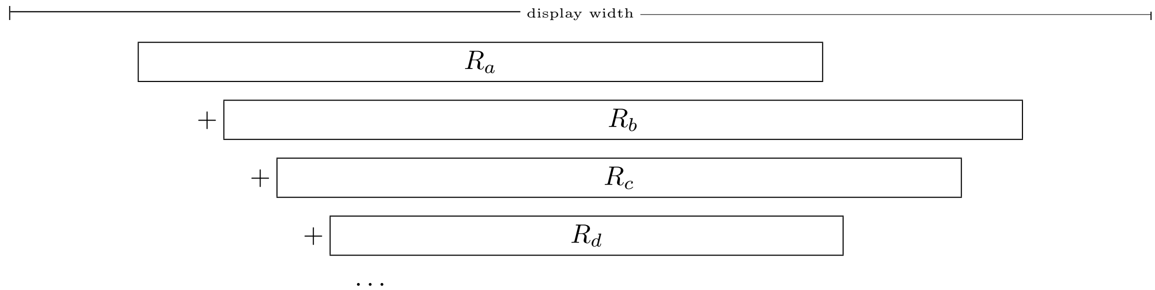
In a skew ladder layout, the combined LHS width plus width of R_1 does not exceed the available width, but one of the other RHS's is so wide that aligning its relation symbol with the others cannot be done without making it run over the right margin: $\mathrm{width}(L) + \mathrm{width}_{\mathrm{max}}(R_i) > \mathrm{width}_{\mathrm{avail}}$. In that case we next try aligning all but the first relation symbol, allowing all the R_i after R_1 to shift leftward.

14.2.3 Drop ladder layout



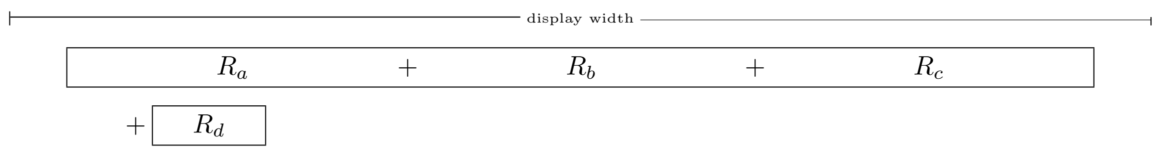
The drop ladder layout is similar to the skew ladder layout but with the width of R_1 too large for it to fit on the same line as the LHS. Then we move R_1 down to a separate line and try again to align all the relation symbols. Note that this layout consumes more vertical space than the skew ladder layout.

14.2.4 Step layout

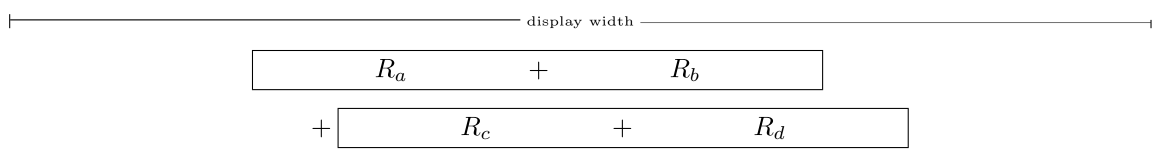


The chief characteristic of the step layout is that there is no relation symbol, so that the available line breaks are (usually) all at binary operator symbols. Let w_1 and w_l be the widths of the first and last fragments. We postulate that the ideal presentation is as follows: Choose a small stairstep indent I (let's say 1 or 2 em). We want the last fragment to be offset at least I from the start of the first fragment, and to end at least I past the end of the first fragment. If there are only two lines these requirements determine a target width $w_T = \max(w_1 + I, w_l + I)$. If there are more than two lines ($l > 2$) then use $w_T = \max(w_1 + (l-1)I, w_l + I, w_{\text{avail}}$ and reset I to $w_T/(l-1)$ if $w_T = w_{\text{avail}}$.

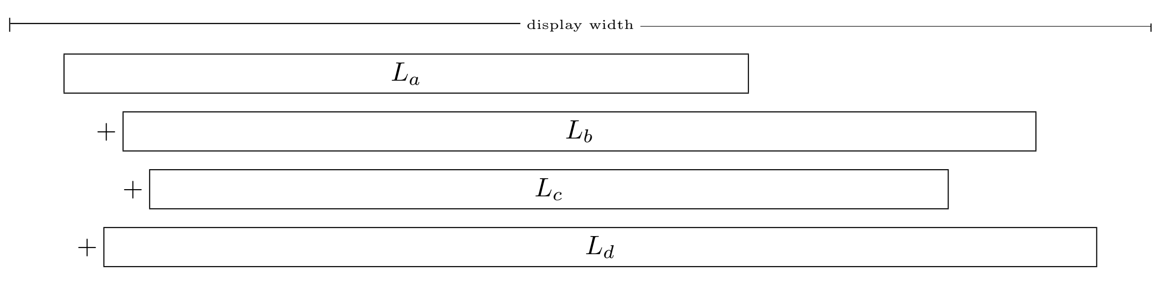
Furthermore, we would like the material to be distributed as evenly as possible over all the lines rather than leave the last line exceedingly short. If the total width is $1.1(\text{width}_{\text{avail}})$, we don't want to have .9 of that on line 1 and .2 of it on line 2:



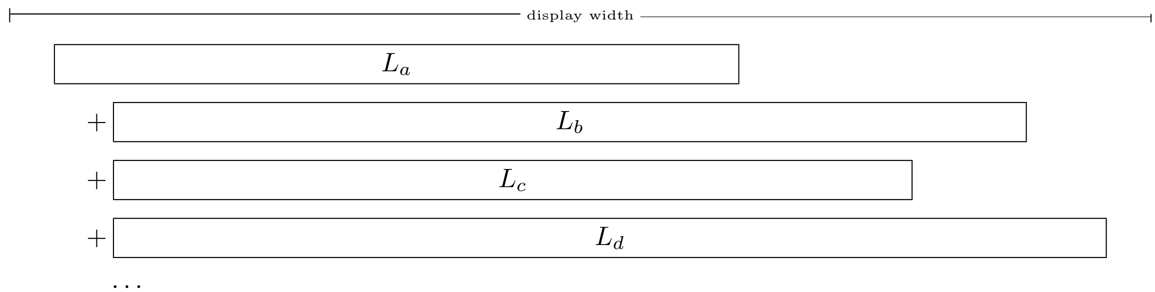
Better to split it as evenly as possible, if the available breakpoints permit.



A degenerate step layout may arise if an unbreakable fragment of the equation is so wide that indenting it to its appointed starting point would cause it to run over the right margin. In that case, we want to shift the fragment leftward just enough to bring it within the right margin:



And then we may want to regularize the indents as in the drop ladder layout. Let's call this a dropped step layout:



14.3 Strategy

Here is the basic procedure for deciding which equation layout to use, before complications like equation numbers and delimiter clearance come into the picture. Let A be the available width, w_{total} the total width of the equation contents, $w(L)$ the width of the left-hand side, $w_{\max}(R)$ the max width of the right-hand sides, I the standard indent for step layout, and O the standard offset for binary operators if a break occurs in the middle of an RHS. Also let t_L and t_R represent certain thresholds for the width of the LHS or the RHS at which a layout decision may change, as explained below.

(1) *Does everything fit on one line?* $w_{\text{total}} \leq A$?

Yes: print the equation on a single line (done).

No: Check whether the equation has both LHS and RHS (2).

(2) *Is there a left-hand side?* Are there any relation symbols in the equation?

Yes: Try a ladder layout (3).

No: Try a step layout (10).

(3) *Does the LHS leave room to fit the widest RHS?* $w(L) + w_{\max}(R) < A$?

Yes: Use a straight ladder layout (5).

No: Check the width of the LHS (4).

(4) *Is the LHS relatively short?* $w(L) \leq t_L$? (where t_L is typically $0.4A$).

Yes: Subdividing one or more of the RHS's may permit us to use a straight ladder layout (5).

No: The straight ladder layout is unlikely to work. Try a skew or drop ladder layout (6).

(5) *Straight ladder layout* Set up a straight ladder parshape [0pt A $w(L)$ $A - w(L)$] and run a trial break. If the combined width of the LHS plus the longest RHS is no greater than A then we should get a satisfactory layout with all line breaks occurring at major division points (relation symbols). Otherwise, we hope, some additional line breaks at minor division points will allow everything to fit within the text column.

Line breaks OK?

Yes: The straight ladder layout succeeded (done).

No: Try a skew or drop ladder layout (6).

(6) *Do the LHS and the first RHS fit on one line?* $w(L) + w(R_1) \leq A$?

Yes: Try a skew ladder layout (7).

No: Try a drop ladder layout (8).

(7) *Skew ladder layout* Set up a parshape [0pt A I $A-I$] and run a trial break.

Line breaks OK?

Yes: Skew ladder layout succeeded (done).

No: One of the unbreakable fragments of the R_i $(i > 1)$ is wider than $A - I$; try an almost-columnar layout (9).

(8) *Drop ladder layout* Set up a parshape [0pt $w(L)$ I $A-I$] and run a trial break. This is the same parshape as for a skew ladder layout except that the width of the first line is limited to the LHS width, so that the RHS is forced to drop down to the next line.

Line breaks OK?

Yes: Drop ladder layout succeeded (done).

No: One of the unbreakable fragments of the R_i $(i > 1)$ is wider than $A - I$; try an almost-columnar layout (9).

(9) *Almost-columnar layout* This presupposes a trial break that yielded a series of expressions or fragments, one per line. Let $w(F)$ denote the width of the first fragment and $w(R_i)$ the widths of the remaining fragments. Set up a parshape [0pt $w(F)$ $A - w_{\max}(R_i)$ $w_{\max}(R_i)$]: in other words, set the first line flush left and the longest line flush right and all other lines indented to the same position as the longest line. But as a matter of fact there is one other refinement for extreme cases: if $w_{\max}(R_i) > A$ then the parshape can be simplified without loss to [0pt $w(F)$ 0pt A]—for that is the net effect of substituting $\min(A, w_{\max})$ in stead of $w_{\max}$. (Done.)

(10) *Step layout* Set target width w_T to $A - 2I$. Set parshape to [0pt w_T I $w_T - I$ $2I$ $w_T - 2I$ $\ldots$ $(l-1)I$ $w_T - (l-1)I$], where $l = \lceil w_{\text{total}}/A \rceil$ is the expected number of lines that will be required. Trial break with that parshape in order to find out the width of the last line.

Indents OK?

Yes: Step layout succeeded (done).

No: One of the fragments is too wide to fit in the allotted line width, after subtracting the indent specified by the parshape. Try a dropped step layout (11)

(11) *Dropped step layout* Set up a parshape [0pt A I $A-I$] and run a trial break. Note that this is actually the same parshape as for a skew ladder layout.

Line breaks OK?

Yes: Dropped step layout succeeded (done).

No: One of the unbreakable fragments of the R_i $(i > 1)$ is wider than $A - I$; as a last resort try an almost-columnar layout (9).

15 To do

- Handling of QED
- Space between `\end{dmath}` and following punctuation will prevent the punctuation from being drawn into the equation.
- Overriding the equation layout
- Overriding the placement of the equation number
- “alignid” option for more widely separated equations where shared alignment is desired (requires two passes)

- Or maybe provide an "alignwidths" option where you give lhs/rhs width in terms of ems? And get feedback later on discrepancies with the actual measured contents?
- `\intertext` not needed within dgroup! But currently there are limitations on floating objects within dgroup.
- `align={1}` or 2, 3, 4 expressing various levels of demand for group-wide alignment. Level 4 means force alignment even if some lines then have to run over the right margin! Level 1, the default, means first break LHS-RHS equations as if it occurred by itself, then move them left or right within the current line width to align them if possible. Levels 2 and 3 mean try harder to align but give up if overfull lines result.
- Need an `\hshift` command to help with alignment of lines broken at a discretionary times sign. Also useful for adjusting inside-delimiter breaks.

Part II

Implementation

The package version here is Michael's v0.90 updated by Bruce Miller. Michael's changes between v0.90 and his last v0.94 will be incorporated where applicable.

The original sources of `breqn` and related files exist in a non-dtx format devised by Michael Downes himself. Lars Madsen has kindly written a Perl script for transforming the original source files into near-perfect dtx state, requiring only very little hand tuning. Without his help it would have been nigh impossible to incorporate the original sources with Michael's comments. A big, big thank you to him.

16 Introduction

The `breqn` package provides environments `dmath`, `dseries`, and `dgroup` for displayed equations with *automatic line breaking*, including automatic indention of relation symbols and binary operator symbols at the beginning of broken lines. These environments automatically pull in following punctuation so that it can be written in a natural way. The `breqn` package also provides a `darray` environment similar to the `array` environment but using `\displaystyle` for all the array cells and providing better interline spacing (because the vertical ruling feature of `array` is dropped). These are all autonumbered environments like `equation` and have starred forms that don't add a number. For a more comprehensive and detailed description of the features and intended usage of the `breqn` package see `breqndoc.tex`.

17 Strategy

Features of particular note are the ability to have linebreaks even within a `\left`–`\right` pair of delimiters, and the automatic alignment on relations and binary operators of a split equation. To make `dmath` handle all this, we begin by setting the body of the equation in a special paragraph form with strategic line breaks whose purpose is not to produce line breaks in the final printed output but rather to mark significant points in the equation and give us entry points for unpacking `\left`–`\right` boxes. After the initial typesetting, we take the resulting stack of line fragments and, working backward, splice them into a new, single-line paragraph; this will eventually be poured into a custom parshape, after we do some measuring to calculate what that parshape should be. This streamlined horizontal list may contain embedded material from user commands intended to alter line breaks, horizontal alignment, and interline spacing; such material requires special handling.

To make the 'shortskip' possibility work even for multiline equations, we must plug in a dummy TeX display to give us the value of `\predisplaysize`, and calculate for ourselves when to apply the short skips.

In order to measure the equation body and do various enervating calculations on whether the equation number will fit and so on, we have to set it in a box. Among other things, this means that we can't unhbox it inside `$$`...`$$`, or even `$` ...`$`: TeX doesn't allow you to `\unhbox` in math mode. But we do want to unhbox it rather than just call `\box`, otherwise we can't take advantage of available shrink from `\medmuskip` to make equations shrink to fit in the available width. So even for simple one-line equations we are forced to fake a whole display without going through TeX's primitive display mechanism (except for using it to get `\predisplaysize` as mentioned above).

In the case of a framed equation body, the current implementation is to set the frame in a separate box, of width zero and height zero, pinned to the upper left corner of the equation body, and then print the equation body on top of it. For attaching an equation number it would be much simpler to wrap the equation body in the frame and from then on treat the body as a single box instead of multiple line boxes. But I had a notion that it might be possible some day to support vertical stretching of the frame.

18 Prelim

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}
```

Declare package name and date.

```
\RequirePackage{expl3}
\ProvidesExplPackage{breqn}{2018/09/14}{0.98f}{Breaking equations}
```

Regrettably, breqn is internally a mess, so we have to take some odd steps.

```
\ExplSyntaxOff
```

19 Package options

Most options are set with the `\options` command (which calls `\setkeys`) because the standard package option mechanism doesn't provide support for key-value syntax.

First we need to get the catcodes sorted out.

```
\edef\breqnpopcats{%
  \catcode\number`\"=\number\catcode`\"
  \relax}
\AtEndOfPackage{\breqnpopcats}%
\catcode`\^=7 \catcode`\_=8 \catcode`\"=12 \relax
\DeclareOption{mathstyleoff}{%
  \PassOptionsToPackage{mathstyleoff}{flexisym}%
}
```

Process options.

```
\ProcessOptions\relax
```

20 Required packages

The flexisym package makes it possible to attach extra actions to math symbols, in particular mathbin, mathrel, mathopen, and mathclose symbols. Normally it would suffice to call `\RequirePackage` without any extra testing, but the nature of the package is such that it is likely to be called earlier with different (no) options. Then is it really helpful to be always warning the user about 'Incompatible Package Options!'? I don't think so.

```
\@ifpackageloaded{flexisym}{}{%
  \RequirePackage{flexisym}[2009/08/07]
```

```
    \edef\breqnpopcats{\breqnpopcats
    \catcode\number`\^=\number\catcode`\^
    \catcode\number`\_=\number\catcode`\_
  }%
  \catcode`\^=7 \catcode`\_=8 \catcode`\"=12 \relax
}
```

The keyval package for handling equation options and calc to ease writing computations.

```
\RequirePackage{keyval,calc}\relax
```

And add an `\options` cmd for processing package options that require an argument. Maybe this will get added to the keyval package eventually.

```
\@ifundefined{options}{%
```

`\options`

Get the package options and run setkeys on them.

```
\newcommand{\options}[2]{%
  \expandafter\options@a\csname opt@#1.sty\endcsname{#2}%
  \setkeys{#1}{#2}%
}
```

`\options@a` `\options@b` `\options@c` `\options@d`

Redefine `\opt@pkgname.sty` as we go along to take out the options that are handled and leave the ones that are not.

```
\def\options@a#1#2{%
  \edef\@tempa{\options@b#2,\@empty\@nil}%
  \ifx#1\relax \let#1\@empty\fi
  \xdef#1{#1\ifx#1\@empty\@xp\@gobble\@tempa\@empty\else\@tempa \fi}%
}
```

Add the next option, and recurse if there remain more options.

```
\def\options@b#1,#2#3\@nil{%
  \options@c#1 \@nil
  \ifx#2\@empty \else\options@b#2#3\@nil\fi
}
```

Discard everything after the first space.

```
\def\options@c#1 #2\@nil{\options@d#1=\@nil}
```

Discard everything after the first = sign; add a comma only if the remainder is not empty.

```
\def\options@d#1=#2\@nil{\ifx\@empty #1\@empty\else,\fi#1}
```

The tail of the `\@ifundefined` test.

```
}{}% end @ifundefined test
```

21 Some useful tools

`\@nx` `\@xp`

The comparative brevity of `\@nx` and `\@xp` is valuable not so much for typing convenience as for reducing visual clutter in code sections that require a lot of expansion control.

```
\let\@nx\noexpand
\let\@xp\expandafter
```

\@emptytoks Constant empty token register, analogous to `\@empty`.

```
\@ifundefined{@emptytoks}{\newtoks\@emptytoks}{}
```

\f@ur Constants 0–3 are provided in plain TeX, but not 4.

```
\chardef\f@ur=4
```

\inf@bad `\inf@bad` is for testing box badness.

```
\newcount\inf@bad \inf@bad=1000000
```

\maxint We want to use `\maxint` rather than coerced `\maxdimen` for `\linepenalty` in one place.

```
\newcount\maxint \maxint=2147483647
```

\int@a
\int@b
\int@b
Provide some shorter aliases for various scratch registers.

```
\let\int@a=\@tempcnta
\let\int@b=\@tempcntb
\let\int@c=\count@
```

\dim@a
\dim@b
\dim@c
\dim@d
\dim@e
\dim@A
Same for dimen registers.

```
\let\dim@a\@tempdima
\let\dim@b\@tempdimb
\let\dim@c\@tempdimc
\let\dim@d\dimen@
\let\dim@e\dimen@ii
\let\dim@A\dimen@i
```

\skip@a
\skip@b
\skip@c
Same for skip registers.

```
\let\skip@a\@tempskipa
\let\skip@b\@tempskipb
\let\skip@c\skip@
```

\toks@a
\toks@b
\toks@c
\toks@d
\toks@e
\toks@f
Same for token registers.

```
\let\toks@a\@temptokena
\let\toks@b\toks@
\toksdef\toks@c=2
\toksdef\toks@d=4
\toksdef\toks@e=6
\toksdef\toks@f=8
```

\abs@num We need an absolute value function for comparing penalties.

```
\def\abs@num#1{\ifnum#1<\z@-\fi#1}
```

\@ifnext
\@ifnexta
The `\@ifnext` function is a variation of `\@ifnextchar` that doesn't skip over intervening whitespace. We use it for the optional arg of \\nside `dmath` etc. because we don't want unwary users to be tripped up by an unexpected attempt on LaTeX's part to interpret a bit of math as an optional arg:

```
\begin{equation}
...\\
[z,w]...
\end{equation}
```

.

```
\def\@ifnext#1#2#3{%
  \let\@tempd= #1\def\@tempa{#2}\def\@tempb{#3}%
  \futurelet\@tempc\@ifnexta
}
```

Switch to `\@tempa` iff the next token matches.

```
\def\@ifnexta{\ifx\@tempc\@tempd \let\@tempb\@tempa \fi \@tempb}
```

`\@ifstar` Similarly let's remove space-skipping from `\@ifstar` because in some rare case of \\nside an equation, followed by a space and a `*` where the `*` is intended as the math binary operator, it would be a disservice to gobble the star as an option of the \\ommand. In all other contexts the chance of having a space *before* the star is extremely small: either the command is a control word which will get no space token after it in any case because of TEX's tokenization rules; or it is a control symbol such as \\"*" which is exceedingly unlikely to be written as \\"*" by any one who really wants the `*` to act as a modifier for the \\ommand.

```
\def\@ifstar#1#2{%
  \let\@tempd*\def\@tempa*{#1}\def\@tempb{#2}%
  \futurelet\@tempc\@ifnexta
}
```

`\@optarg` Utility function for reading an optional arg *without* skipping over any intervening spaces.

```
\def\@optarg#1#2{\@ifnext[{#1}{#1[#2]}}
```

`\@True` `\@False` `\@Not` `\@And` After `\let\foo\@True` the test

```
\if\foo
```

evaluates to true. Would rather avoid `\newif` because it uses three csnames per Boolean variable; this uses only one.

```
\def\@True{00}
\def\@False{01}
\def\@Not#1{0\ifcase#11 \or\@xp 1\else \@xp 0\fi}
\def\@And#1#2{0\ifcase#1#2 \@xp 0\else \@xp 1\fi}
\def\@Or#1#2{0\ifnum#1#2<101 \@xp 0\else \@xp 1\fi}
```

```
\def\theb@@le#1{\if#1 True\else False\fi}
```

`\freeze@glue` Remove the stretch and shrink from a glue register.

```
\def\freeze@glue#1{#11#1\relax}
```

`\z@rule` `\keep@glue` Note well the intentional absence of `\relax` at the end of the replacement text of `\z@rule`; use it with care.

```
\def\z@rule{\vrule\@width\z@}% no \relax ! use with care
```

Different ways to keep a bit of glue from disappearing at the beginning of a line after line breaking:

- Zero-thickness rule
- Null character

- `\vadjust{}` (*The TEXbook*, Exercise ??)

The null character idea would be nice except it creates a mathord which then screws up math spacing for e.g., a following unary minus sign. (the vrule *is* transparent to the math spacing). The vadjust is the cheapest in terms of box memory—it vanishes after the pass through TEX's paragrapher. It is what I would have used, except that the equation contents get run through two paragraphing passes, once for breaking up LR boxes and once for the real typesetting. If `\keep@glue` were done with an empty vadjust, it would disappear after the first pass and—in particular—the pre-bin-op adjustment for relation symbols would disappear at a line break.

```
\def\keep@glue{\z@rule\relax}
```

`\replicate` This is a fully expandable way of making N copies of a token list. Based on a post of David Kastrup to comp.text.tex circa January 1999. The extra application of `\number` is needed for maximal robustness in case the repeat count N is given in some weird TEX form such as `"E9` or `\count9`.

```
% usage: \message{H\replicate{5}{i h}ow de doo dee!}
\begingroup \catcode`\&=11
\gdef\replicate#1{%
  \csname &\expandafter\replicate@a\romannumeral\number\number#1 000q\endcsname
}
\endgroup
```

`\replicate@a`

```
\long\def\replicate@a#1#2\endcsname#3{#1\endcsname{#3}#2}
```

`\8m` fix

```
\begingroup \catcode`\&=11
\long\gdef\&m#1#2{#1\csname &#2\endcsname{#1}}
\endgroup
```

`\8q` fix

```
\@xp\let\csname\string &q\endcsname\@gobble
```

`\mathchars@reset` Need to patch up this function from flexisym a little, to better handle certain constructed symbols like `\neq`.

```
\ExplSyntaxOn
\g@addto@macro\mathchars@reset{%
  %\let\@symRel\@secondoftwo \let\@symBin\@secondoftwo
  %\let\@symDeL\@secondoftwo \let\@symDeR\@secondoftwo
  %\let\@symDeB\@secondoftwo
  \cs_set_eq:NN \math_csym_Rel:Nn \use_ii:nn
  \cs_set_eq:NN \math_csym_Bin:Nn \use_ii:nn
  \cs_set_eq:NN \math_csym_DeL:Nn \use_ii:nn
  \cs_set_eq:NN \math_csym_DeR:Nn \use_ii:nn
  \cs_set_eq:NN \math_csym_DeB:Nn \use_ii:nn
}
\ExplSyntaxOff
```

`\eq@cons` LaTeX's `\@cons` appends to the end of a list, but we need a function that adds material at the beginning.

```
\def\eq@cons#1#2{%
  \begingroup \let\@elt\relax \xdef#1{\@elt{#2}#1}\endgroup
}
```

`\@saveprimitive` If some preceding package redefined one of the primitives that we must change, we had better do some checking to make sure that we are able to save the primitive meaning for internal use. This is handled by the `\@saveprimitive` function. We follow the example of `\@@input` where the primitive meaning is stored in an internal control sequence with a `@@` prefix. Primitive control sequences can be distinguished by the fact that `\string` and `\meaning` return the same information. Well, not quite all: `\nullfont` and `\topmark` and the other `\...mark` primitives being the exceptions.

```
\providecommand{\@saveprimitive}[2]{%
  \begingroup
  \edef\@tempa{\string#1}\edef\@tempb{\meaning#1}%
  \ifx\@tempa\@tempb \global\let#2#1%
  \else
```

If [arg1] is no longer primitive, then we are in trouble unless [arg2] was already given the desired primitive meaning somewhere else.

```
    \edef\@tempb{\meaning#2}%
    \ifx\@tempa\@tempb
    \else \@saveprimitive@a#1#2%
    \fi
  \fi
  \endgroup
}
```

Aux function, check for the special cases. Most of the time this branch will be skipped so we can stuff a lot of work into it without worrying about speed costs.

```
\providecommand\@saveprimitive@a[2]{%
  \begingroup
  \def\@tempb##1#1##2{\edef\@tempb{##2}\@car{}}%
  \@tempb\nullfont{select font nullfont}%
    \topmark{\string\topmark:}%
    \firstmark{\string\firstmark:}%
    \botmark{\string\botmark:}%
    \splitfirstmark{\string\splitfirstmark:}%
    \splitbotmark{\string\splitbotmark:}%
    #1{\string#1}%
    \@nil % for the \@car
  \edef\@tempa{\expandafter\strip@prefix\meaning\@tempb}%
  \edef\@tempb{\meaning#1}%
  \ifx\@tempa\@tempb \global\let#2#1%
  \else
    \PackageError{breqn}%
      {Unable to properly define \string#2; primitive
      \noexpand#1no longer primitive}\@eha
    \fi
```

```
  \fi
  \endgroup
}
```

\@@math \@@endmath \@@display \@@enddisplay

Move the math-start and math-end functions into control sequences. If I were redesigning TEX I guess I'd put these functions into primitive control words instead of linking them to a catcode. That way TEX would not have to do the special lookahead at a `$` to see if there's another one coming up. Of course that's related to the question of how to provide user shorthand for common constructions: TEX, or an editing interface of some sort.

```
\begingroup \catcode`\$=\thr@@ % just to make sure
  \global\let\@@math=$ \gdef\@@display{$$}% $$$
\endgroup
\let\@@endmath=\@@math
\let\@@enddisplay=\@@display
```

\@@insert \@@mark \@@vadjust

Save the primitives `\vadjust`, `\insert`, `\mark` because we will want to change them locally during equation measuring to keep them from getting in the way of our vertical decomposition procedures. We follow the example of `\@@input`, `\@@end`, `\@@par` where the primitive meaning is stored in an internal control sequence with a `@@` prefix.

```
\@saveprimitive\vadjust\@@vadjust
\@saveprimitive\insert\@@insert
\@saveprimitive\mark\@@mark
```

22 Debugging

Debugging help.

```
⟨*trace⟩
\errorcontextlines=2000\relax
\typeout{BREQN DEBUGGING MODE ACTIVE}
```

\breqn@debugmsg

Print a debugging message.

```
\long\def\breqn@debugmsg#1{\GenericWarning{||}{||=\space#1}}
```

\debugwr

Sometimes the newline behavior of `\message` is unsatisfactory; this provides an alternative.

```
\def\debugwr#1{\immediate\write\sixt@@n{||= #1}}
```

\debug@box

Record the contents of a box in the log file, without stopping.

```
\def\debug@box#1{%
  \batchmode{\showboxbreadth\maxdimen\showboxdepth99\showbox#1}%
  \errorstopmode
}
```

\eqinfo

Show lots of info about the material before launching into the trials.

```
\def\eqinfo{%
  \debug@box\EQ@copy
  \wlog{!! EQ@copy: \the\wd\EQ@copy\space x
    \the\ht\EQ@copy+\the\dp\EQ@copy
  }%
}
```

\debug@para

Check params that affect line breaking.

```
\def\debug@para{%
  \debugwr{\hsize\the\hsize, \parfillskip\the\parfillskip}%
  \breqn@debugmsg{\leftskip\the\leftskip, \rightskip\the\rightskip}%
  \breqn@debugmsg{\linepenalty\the\linepenalty, \adjdemerits\the\adjdemerits}%
  \breqn@debugmsg{\pretolerance\the\pretolerance, \tolerance\the\tolerance,
    \parindent\the\parindent}%
}
```

⟨/trace⟩

23 The \listwidth variable

The dimen variable \listwidth is \linewidth plus \leftmargin plus \rightmargin, which is typically less than \hsize if the list depth is greater than one. In case a future package will provide this variable, define it only if not yet defined.

```
\@ifundefined{listwidth}{\newdimen\listwidth}{}
\listwidth=\z@
```

24 Parameters

Here follows a list of parameters needed.

\eqfontsize
\eqcolor
\eqmargin
\eqindent
\eqbinoffset
\eqnumside
\eqnumplace
\eqnumsep
\eqnumfont
\eqnumform
\eqnumsize
\eqnumcolor
\eqlinespacing
\eqlineskip
\eqlineskiplimit
\eqstyle
\eqinterlinepenalty
\intereqpenalty
\intereqskip

Note: avoid M, m, P, p because they look like they might be the start of a keyword 'minus' or 'plus'. Then TEX looks further to see if the next letter is i or l. And if the next thing is an undefined macro, the attempt to expand the macro results in an error message.

```
\def\eqfontsize{}          % Inherit from context    [NOT USED?]
\def\eqcolor{black}        % Default to black        [NOT USED?]
\newdimen\eqnumsep \eqnumsep=10pt         % Min space between equ number and body
\newdimen\eqmargin \eqmargin=8pt          % For 'multline' gap emulation
```

The \eqindent and \eqnumside variables need to have their values initialized from context, actually. But that takes a bit of work, which is postponed till later.

```
\def\eqindent{C}%          % C or I, centered or indented
\def\eqnumside{R}%         % R or L, right or left
\def\eqnumplace{M}%        % M or T or B, middle top or bottom
```

Typesetting the equation number is done thus:

```
{\eqnumcolor \eqnumsize \eqnumfont{\eqnumform{\eq@number}}}
```

.

```
%d\eqnumfont{\upshape}% % Upright even when surrounding text is slanted
\def\eqnumfont{}%          % Null for easier debugging [mjd,1997/09/26]
\def\eqnumform#1{(#1\@@italiccorr)} % Add parens
\def\eqnumsize{}           % Allow numbers to have different typesize ...
```

Tricky questions on \eqnumsize. Should the default be \normalsize? Then the user can scale down the equation body with \small and not affect the equation number. Or should the default be empty? Then in large sections of smaller text, like the dangerous bend stuff in *TEXbook*, the equation number size will keep in synch with the context. Maybe need an `\eqbodysize` param as well to allow separating the two cases.

```
\def\eqnumcolor{}         % ... or color than eq body e.g. \color{blue}
\newlength\eqlinespacing \eqlinespacing=14pt plus2pt % Base-to-base space between lines
\newlength\eqlineskip \eqlineskip=3pt plus2pt % Min space if eqlinespacing too small
\newdimen\eqlineskiplimit \eqlineskiplimit=2pt  % Threshold for switching to eqlineskip
```

The value of `\eqbinoffset` should include a negative shrink component that cancels the shrink component of medmuskip, otherwise there can be a noticeable variation in the indent of adjacent lines if one is shrunken a lot and the other isn't.

```
\newmuskip \eqbinoffset \eqbinoffset=15mu minus-3mu % Offset from mathrel alignment pt for mathbins
\newmuskip\eqdelimoffset \eqdelimoffset=2mu    % Additional offset for break inside delims
\newdimen\eqindentstep \eqindentstep=8pt     % Indent used when LHS wd is n/a or too large
\newtoks\eqstyle           % Customization hook
\newcount\eqbreakdepth \eqbreakdepth=2       % Allow breaks within delimiters to this depth
\newcount \eqinterlinepenalty \eqinterlinepenalty=10000 % No page breaks between equation lines
\newcount \intereqpenalty \intereqpenalty=1000   % Pagebreak penalty between equations [BRM: Was \@M]
\newlength \intereqskip \intereqskip=3pt plus2pt % Additional vert space between equations
\newcount\prerelpenalty \prerelpenalty=-\@M   % Linebreak penalty before mathrel symbols
\newcount\prebinoppenalty \prebinoppenalty=888  % Linebreak penalty before mathbins
```

When breaking equations we never right-justify, so a stretch component of the muskip is never helpful and sometimes it is definitely undesirable. Note that thick/medmuskips frozen inside a fraction or radical may turn out noticeably larger than neighboring unfrozen ones. Nonetheless I think this way is the best compromise short of a new TEX that can make those built-up objects shrink horizontally in proportion; the alternative is to pretty much eliminate the shrink possibility completely in displays.

```
\newmuskip \Dmedmuskip \Dmedmuskip=4mu minus 3mu % medmuskip in displays
\newmuskip \Dthickmuskip \Dthickmuskip=5mu minus 2mu % thickmuskip in displays
```

And now some internal variables. 1997/10/22: some of these are dead branches that need to be pruned.

MH: Started cleaning up a bit. No more funny loops.

```
\def\eq@number{}          % Internal variable
\newlength\eqleftskip \eqleftskip=\@centering  % Space on the left  [NOT USED?]
\newlength\eqrightskip \eqrightskip=\@centering % Space on the right [NOT USED?]
\newlength\eq@vspan \eq@vspan=\z@skip     % Glue used to vcenter the eq number
\newmuskip\eq@binoffset \eq@binoffset=\eqbinoffset % Roughly, \eqbinoffset + \eqdelimoffset
\newsavebox\EQ@box               % Storage for equation body
\newsavebox\EQ@copy              % For eq body sans vadjust/insert/mark material
\newsavebox\EQ@numbox            % For equation number
\newdimen\eq@wdNum         % width of number + separation [NEW]
\newsavebox\GRP@numbox            % For group number [NEW]
\newdimen\grp@wdNum         % width of number + separation [NEW]
%%B\EQ@vimbox            % Vadjust, insert, or mark material
%%B\EQ@vimcopy           % Spare copy of same
%%B\eq@impinging         % Temporary box for measuring number placement
\newcount \eq@lines          % Internal counter, actual number of lines
\newcount \eq@curline       % Loop counter
```

```
\newcount \eq@badness      % Used in testing for overfull lines
\newcount \EQ@vims          % For bookkeeping
\def\@eq@numbertrue{\let\eq@hasNumber\@True}%
\def\@eq@numberfalse{\let\eq@hasNumber\@False}%
\let\eq@hasNumber\@False
```

Here for the dimens, it would be advisable to do some more careful management to conserve dimen registers. First of all, most of the dimen registers are needed in the measuring phase, which is a tightly contained step that happens after the contents of the equation have been typeset into a box and before any external functions have a chance to regain control—e.g.,, the output routine. Therefore it is possible to make use of the the dimen registers 0–9, reserved by convention for scratch use, without fear of conflict with other macros. But I don't want to use them directly with the available names:

`\dimen@ \dimen@i \dimen@ii \dimen3 \dimen4 ... \dimen9`

. It would be much more useful to have names for these registers indicative of way they are used.

Another source whence dimen registers could be borrowed is the amsmath package, which allocates six registers for equation-measuring purposes. We can reuse them under different names since the amsmath functions and our functions will never be used simultaneously.

`\eqnshift@ \alignsep@ \tagshift@ \tagwidth@ \totwidth@ \lineht@`

```
\newdimen\eq@dp         % Depth of last line
\newdimen\eq@wdL        % Width of the left-hand-side
\newdimen\eq@wdT        % Total width for framing
\newdimen\eq@wdMin      % Width of narrowest line in equation
\newdimen\grp@wdL       % Max width of LHS's in a group
\newdimen\grp@wdR       % Max RHS of all equations in a group
\newdimen\grp@wdT
\newdimen\eq@wdRmax
\newdimen\eq@firstht    % Height of first line
```

BRM: measure the condition too.

```
\newdimen\eq@wdCond
\newdimen\eq@indentstep % Indent amount when LHS is not present
\newdimen\eq@linewidth  % Width actually used for display
\newdimen\grp@linewidth % Max eq@linewidth over a group
```

Maybe `\eq@hshift` could share the same register as `\mathindent` [mjd,1997/10/22].

```
\newdimen\eq@hshift
\let\eq@isIntertext\@False
```

Init `\eq@indentstep` to a nonzero value so that we can detect and refrain from clobbering a user setting of zero. And `\eq@sidespace` to `\maxdimen` because that is the right init before computing a min.

```
\eq@indentstep=\maxdimen
\newdimen\eq@given@sidespace
```

`\eq@overrun` MH: Appears to be unused.

Not a dimen register; don't need to advance it.

```
\def\eq@overrun{0pt}
```

To initialize `\eqnumside` and `\eqindent` properly, we may need to grub around a bit in `\@filelist`. However, if the amsmath package was used, we can use its option data. More trouble: if a documentclass sends an option of `leqno` to amsmath by default, and it gets overridden by the user with a `reqno` documentclass option, then amsmath believes itself to have received *both* options.

```
\@ifpackagewith{amsmath}{leqno}{%
  \@ifpackagewith{amsmath}{reqno}{}{\def\eqnumside{L}}%
}{%
```

If the amsmath package was not used, the next method for testing the `leqno` option is to see if `leqno.clo` is present in `\@filelist`.

```
  \def\@tempa#1,leqno.clo,#2#3\@nil{%
    \ifx @#2\relax\else \def\eqnumside{L}\fi
  }%
  \@xp\@tempa\@filelist,leqno.clo,@\@nil
```

Even that test may fail in the case of amsart if it does not load amsmath. Then we have to look whether `\iftagsleft@` is defined, and if so whether it is true. This is tricky if you want to be careful about conditional nesting and don't want to put anything in the hash table unnecessarily.

```
  \if L\eqnumside
  \else
    \@ifundefined{iftagsleft@}{}{%
      \edef\eqnumside{%
        \if TT\csname fi\endcsname\csname iftagsleft@\endcsname
          L\else R\fi
      }%
    }
  \fi
}
```

A similar sequence of tests handles the 'fleqn or not fleqn' question for the article and amsart documentclasses.

```
\@ifpackagewith{amsmath}{fleqn}{%
  \def\eqindent{I}%
}{%
  \def\@tempa#1,fleqn.clo,#2#3\@nil{%
    \ifx @#2\relax\else \def\eqindent{I}\fi
  }%
  \@xp\@tempa\@filelist,fleqn.clo,@\@nil
  \if I\eqindent
  \else
    \@ifundefined{if@fleqn}{}{%
      \edef\eqindent{%
        \if TT\csname fi\endcsname\csname if@fleqn\endcsname
          I\else C\fi
      }%
    }%
  \fi
}
```

BRM: This conditional implies we must use ALL indented or ALL centered?

```
%\if I\eqindent
  \@ifundefined{mathindent}{%
    \newdimen\mathindent
  }{%
    \@ifundefined{@mathmargin}{}{%
      \mathindent\@mathmargin
    }%
  }
%\fi
```

25 Measuring equation components

Measure the left-hand side of an equation. This function is called by mathrel symbols. For the first mathrel we want to discourage a line break more than for following mathrels; so `\mark@lhs` gobbles the following `\rel@break` and substitutes a higher penalty.

Maybe the LHS should be kept in a separate box.

`\EQ@hasLHS` Boolean: does this equation have a "left-hand side"?

```
\let\EQ@hasLHS=\@False
```

`\EQ@QED` If nonempty: the qed material that should be incorporated into this equation after the final punctuation.

```
\let\EQ@QED=\@empty
```

`\mark@lhs`

```
\def\mark@lhs#1{%
  \ifnum\lr@level<\@ne
    \let\mark@lhs\relax
    \global\let\EQ@hasLHS=\@True
    \global\let\EQ@prebin@space\EQ@prebin@space@a
    \mark@lhs@a
```

But the penalty for the first mathrel should still be lower than a binoppenalty. If not, when the LHS contains a binop, the split will occur inside the LHS rather than at the mathrel. On the other hand if we end up with a multline sort of equation layout where the RHS is very short, the break before the relation symbol should be made *less* desirable than the breakpoints inside the LHS. Since a lower penalty takes precedence over a higher one, we start by putting in the highest relpenalty; during subsequent measuring if we find that that RHS is not excessively short then we put in an extra "normal" relpenalty when rejoining the LHS and RHS.

```
    \penalty9999 % instead of normal \rel@break
  % else no penalty = forbid break
  \fi
}
```

`\mark@lhs@a` Temporarily add an extra thickmuskip to the LHS; it will be removed later. This is necessary to compensate for the disappearance of the thickmuskip glue preceding a mathrel if a line break is taken at that point. Otherwise we would have to make our definition of mathrel symbols more complicated, like the one for mathbins. The penalty of 2 put in with vadjust is a flag for `\eq@repack` to suggest that the box containing this line should be measured to find the value of `\eq@wdL`. The second vadjust ensures that the normal prerelpenalty and thickmuskip will not get lost at the line break during this preliminary pass.

BRM: I originally thought the `\mskip\thickmuskip` was messing up summation limits in LHS. But I may have fixed that problem by fixing other things...

```
\def\mark@lhs@a{%
  \mskip\thickmuskip \@@vadjust{\penalty\tw@}\penalty-\@Mi\@@vadjust{}%
}
```

`\hiderel` If you want the LHS to extend past the first mathrel symbol to a following one, mark the first one with `\hiderel`:

```
a \hiderel{=} b = c...
```

.

I'm not sure now why I didn't use `\begingroup` `\endgroup` here

mjd,1999/01/21

.

```
\newcommand\hiderel[1]{\mathrel{\advance\lr@level\@ne#1}}
```

`\m@@Bin` `\m@@Rel` `\bin@break` `\rel@break` `\bin@mark` `\rel@mark` `\d@@Bin` `\d@@Rel`

cf. flexisym handling of mathbins and mathrels. These are alternate definitions of `\m@Bin` and `\m@Rel`, activated by `\display@setup`.

```
%%%%\let\m@@Bin\m@Bin
%%%%%\let\m@@Rel\m@Rel
\let\EQ@prebin@space\relax
\def\EQ@prebin@space@a{\mskip-\eq@binoffset \keep@glue \mskip\eq@binoffset}
\def\bin@break{\ifnum\lastpenalty=\z@\penalty\prebinoppenalty\fi
  \EQ@prebin@space}
\def\rel@break{%
  \ifnum\abs@num\lastpenalty <\abs@num\prerelpenalty
    \penalty\prerelpenalty
  \fi
}
\ExplSyntaxOn
%%%\def\d@@Bin{\bin@break \m@@Bin}
%%%%\def\d@@Rel{\mark@lhs \rel@break \m@@Rel}
\cs_set:Npn \math_dsym_Bin:Nn {\bin@break\math_bsym_Bin:Nn}
\cs_set:Npn \math_dsym_Rel:Nn {\mark@lhs \rel@break \math_bsym_Rel:Nn }
\ExplSyntaxOff
```

The difficulty of dealing properly with the subscripts and superscripts sometimes appended to mathbins and mathrels is one of the reasons that we do not attempt to handle the mathrels as a separate 'column' a la `eqnarray`.

`\m@@symRel` More of the same.

`\d@@symRel` `\m@@symBin` `\d@@symBin` `\m@@symDel` `\d@@symDel` `\m@@symDeR` `\d@@symDeR` `\m@@symDeB` `\d@@symDeB` `\m@@symDeA` `\d@@symDeA`

```
\ExplSyntaxOn
%%\let\m@@symRel\@symRel
%%%\def\d@@symRel{\mark@lhs \rel@break \m@@symRel}

\cs_set_protected:Npn \math_dcsym_Bin:Nn {\bin@break \math_bcsym_Bin:Nn}
\cs_set_protected:Npn \math_dcsym_Rel:Nn { \mark@lhs \rel@break \math_bcsym_Rel:Nn}


%%\let\m@@symBin\@symBin \def\d@@symBin{\bin@break \m@@symBin}
%%\let\m@@symDel\@symDel
%%\let\m@@symDeR\@symDeR
%%\let\m@@symDeB\@symDeB
%%\let\m@@symDeA\@symDeA
```

`\display@setup` `\everydisplay`

Setup. Note that LaTeX reserves the primitive `\everydisplay` under the name `\frozen@everydisplay`. BRM: Disable this! It also affects non-breqn math!!!!

```
%\global\everydisplay\expandafter{\the\everydisplay \display@setup}
```

Change some math symbol function calls.

```
\def\display@setup{%
  \medmuskip\Dmedmuskip \thickmuskip\Dthickmuskip
   \math_setup_display_symbols:
  %%\let\m@Bin\d@@Bin \let\m@Rel\d@@Rel
  %%\let\@symRel\d@@symRel \let\@symBin\d@@symBin
  %%\let\m@DeL\d@@DeL \let\m@DeR\d@@DeR \let\m@DeB\d@@DeB
  %%\let\m@DeA\d@@DeA
  %%\let\@symDeL\d@@symDeL \let\@symDeR\d@@symDeR
  %%\let\@symDeB\d@@symDeB \let\@symDeA\d@@symDeA
  \let\left\eq@left \let\right\eq@right \global\lr@level\z@
  \global\eq@wdCond\z@          %BRM: new
```

If we have an embedded array environment (for example), we don't want to have each math cell within the array resetting `\lr@level` globally to 0—not good! And in general I think it is safe to say that whenever we have a subordinate level of boxing we want to revert to a normal math setup.

```
  \everyhbox{\everyhbox\@emptytoks
    \let\display@setup\relax \textmath@setup \let\textmath@setup\relax
  }%
  \everyvbox{\everyvbox\@emptytoks
    \let\display@setup\relax \textmath@setup \let\textmath@setup\relax
  }%
}
```

The `\textmath@setup` function is needed for embedded inline math inside text inside a display.

BRM: DS Experiment: Variant of `\display@setup` for use within dseries environmnents

```
\def\dseries@display@setup{%
  \medmuskip\Dmedmuskip \thickmuskip\Dthickmuskip
  \math_setup_display_symbols:
%%%%  \let\m@Bin\d@@Bin
%%%\let\m@Rel\d@@Rel
```

```
%%%  \let\@symRel\d@@symRel
%%% \let\@symBin\d@@symBin
%%%  \let\m@DeL\d@@DeL \let\m@DeR\d@@DeR \let\m@DeB\d@@DeB
%%%  \let\m@DeA\d@@DeA
%%%  \let\@symDeL\d@@symDeL \let\@symDeR\d@@symDeR
%%%  \let\@symDeB\d@@symDeB \let\@symDeA\d@@symDeA
  \let\left\eq@left \let\right\eq@right \global\lr@level\z@
  \everyhbox{\everyhbox\@emptytoks
    \let\display@setup\relax \textmath@setup \let\textmath@setup\relax
  }%
  \everyvbox{\everyvbox\@emptytoks
    \let\display@setup\relax \textmath@setup \let\textmath@setup\relax
  }%
 \displaystyle
}
```

```
\def\textmath@setup{%
   \math_setup_inline_symbols:
%%%%  \let\m@Bin\m@@Bin \let\m@Rel\m@@Rel
%%%%  \let\@symRel\m@@symRel \let\@symBin\m@@symBin
%%%%  \let\m@DeL\m@@DeL \let\m@DeR\m@@DeR \let\m@DeB\m@@DeB
%%%%  \let\m@DeA\m@@DeA
%%%%  \let\@symDeL\m@@symDeL \let\@symDeR\m@@symDeR
%%%%  \let\@symDeB\m@@symDeB \let\@symDeA\m@@symDeA
  \let\left\@@left \let\right\@@right
}

\ExplSyntaxOff
```

`\if@display` `\everydisplay`

The test `\ifinner` is unreliable for distinguishing whether we are in a displayed formula or an inline formula: any display more complex than a simple one-line equation typically involves the use of `$ \displaystyle ... $` instead of `$$...$$`. So we provide a more reliable test. But it might have been provided already by the amsmath package.

```
\@ifundefined{@displaytrue}{%
  \@xp\newif\csname if@display\endcsname
  \everydisplay\@xp{\the\everydisplay \@displaytrue}%
}{}
```

Is there any reason to maintain separate `\everydisplay` and `\eqstyle`?

26 The `dmath` and `dmath*` environments

Options for the `dmath` and `dmath*` environments.

```
\begin{dmath}[label={eq:xyz}]
\begin{dmath}[labelprefix={eq:},label={xyz}]
```

WSPR: added the option for a label prefix, designed to be used in the preamble like so:

```
\setkeys{breqn}{labelprefix={eq:}}
```

```
\define@key{breqn}{label}{%
  \edef\next@label{\noexpand\label{\next@label@pre#1}}%
  \let\next@label@pre\@empty}
\define@key{breqn}{labelprefix}{\def\next@label@pre{#1}}
\global\let\next@label\@empty
\global\let\next@label@pre\@empty
```

Allow a variant number.

```
\begin{dmath}[number={\nref{foo}\textprime}]
```

```
\define@key{breqn}{number}{\def\eq@number{#1}%
  \let\@currentlabel\eq@number
}
```

```
\begin{dmath}[shiftnumber]
\begin{dmath}[holdnumber]
```

Holding or shifting the number.

```
\define@key{breqn}{shiftnumber}{\let\eq@shiftnumber\@True}
\define@key{breqn}{holdnumber}{\let\eq@holdnumber\@True}
```

```
\begin{dmath}[density={.5}]
```

```
\define@key{breqn}{density}{\def\eq@density@factor{#1}}
```

```
\begin{dmath}[indentstep={1em}]
```

To change the amount of indent for post-initial lines. Note: for lines that begin with a mathbin symbol there is a fixed amount of indent already built in (`\eqbinoffset`) and it cannot be reduced through this option. The indentstep amount is the indent used for lines that begin with a mathrel symbol.

```
\define@key{breqn}{indentstep}{\eqindentstep#1\relax}
```

```
\begin{dmath}[compact]
\begin{dmath}[compact=-2000]
```

To make mathrels stay inline to the extent possible, use the compact option. Can give a numeric value in the range $-10000 \ldots 10000$ to adjust the behavior. -10000: always break at a rel symbol; 10000: never break at a rel symbol.

```
\define@key{breqn}{compact}[-99]{\prerelpenalty=#1\relax}
```

```
\begin{dmath}[layout={S}]%
```

Specify a particular layout. We take care to ensure that `\eq@layout` ends up containing one and only one letter.

```
\define@key{breqn}{layout}[?]{%
  \edef\eq@layout{\@car#1?\@nil}%
}
```

```
\begin{dmath}[spread={1pt}]
```

To change the interline spacing in a particular equation.

```
\define@key{breqn}{spread}{%
  \addtolength\eqlinespacing{#1}%
  \addtolength\eqlineskip{#1}%
  \eqlineskiplimit\eqlineskip
}
```

To change the amount of space on the side for "multline" layout.

```
\define@key{breqn}{sidespace}{%
  \setlength\eq@given@sidespace{#1}%
}
```

`\begin{dmath}[style={\small}]`

The `style` option is mainly intended for changing the type size of an equation but as a matter of fact you could put arbitrary LaTeX code here—thus the option name is 'style' rather than just 'typesize'. In order for this option to work when setting options globally, we need to put the code in `\eqstyle` rather than execute it directly.

```
\define@key{breqn}{style}{\eqstyle\@xp{\the\eqstyle #1}}
```

`\begin{dmath}[shortskiplimit={1em}]`

If the line immediately preceeding a display has length l, the first line of the display is indented i, and a shortskip limit s is set, then the spacing above the display is equal to `\abovedisplayshortskip` if $l + s < i$ and `\abovedisplayskip` otherwise. The default shortskip limit is 2 em which is what TeX hardcodes but this parameter overrides that.

```
\define@key{breqn}{shortskiplimit}{\def\eq@shortskiplimit{#1}}
\def\eq@shortskiplimit{2em}
```

`\begin{dmath}[frame]`

The `frame` option merely puts a framebox around the body of the equation. To change the thickness of the frame, give the thickness as the argument of the option. For greater control, you can change the appearance of the frame by redefining `\eqframe`. It must be a command taking two arguments, the width and height of the equation body. The top left corner of the box produced by `\eqframe` will be pinned to the top-left corner of the equation body.

```
\define@key{breqn}{frame}[\fboxrule]{\def\eq@frame{T}%
  \dim@a#1\relax\edef\eq@framewd{\the\dim@a}%
```

Until such time as we provide a frame implementation that allows the frame to stretch and shrink, we'd better remove any stretch/shrink from the interline glue in this case.

```
  \freeze@glue\eqlinespacing \freeze@glue\eqlineskip
}
\define@key{breqn}{fullframe}[]{\def\eq@frame{U}%
  \freeze@glue\eqlinespacing \freeze@glue\eqlineskip
}
\def\eq@frame{F} % no frame
\def\eq@framewd{\fboxrule}
```

Wishful thinking?

`\begin{dmath}[frame={width={2pt},color={blue},sep={2pt}}]`

To change the space between the frame and the equation there is a framesep option.

```
\define@key{breqn}{framesep}[\fboxsep]{%
  \if\eq@frame F\def\eq@frame{T}\fi
  \dim@a#1\relax \edef\eq@framesep{\the\dim@a}%
  \freeze@glue\eqlinespacing \freeze@glue\eqlineskip
}
\def\eq@framesep{\fboxsep}
```

`\begin{dmath}[background={red}]`

Foreground and background colors for the equation. By default the background area that is colored is the size of the equation, plus fboxsep. If you need anything fancier for the background, you'd better do it by defining `\eqframe` in terms of `\colorbox` or `\fcolorbox`.

```
\define@key{breqn}{background}{\def\eq@background{#1}%
  \freeze@glue\eqlinespacing \freeze@glue\eqlineskip
}
%   \end{macrocode}
% \begin{literalcode}
% \begin{dmath}[color={purple}]
% \end{literalcode}
%    \begin{macrocode}
\define@key{breqn}{color}{\def\eq@foreground{#1}}
```

`\begin{dmath}[center]`
`\begin{dmath}[nocenter]`

The `center` option means add leftskip stretch to make the individual lines be centered; this is the default for `dseries`.

```
\define@key{breqn}{center}[]{\let\eq@centerlines\@True}
\define@key{breqn}{nocenter}[]{\let\eq@centerlines\@False}
\let\eq@centerlines\@False
```

`\begin{dgroup}[noalign]`

Equation groups normally have alignment of the primary relation symbols across the whole group. The `noalign` option switches that behavior.

```
\define@key{breqn}{noalign}[]{\let\grp@aligned\@False}
\let\grp@aligned\@True % default
```

`\begin{dgroup}[breakdepth={2}]`

Break depth of 2 means that breaks are allowed at mathbin symbols inside two pairs of delimiters, but not three.

```
\define@key{breqn}{breakdepth}{\eqbreakdepth#1\relax}
```

`\begin{darray}[cols={lcrlcr}]`

The `cols` option only makes sense for the `darray` environment but we liberally allow all the options to be used with all the environments and just ignore any unsensible ones that happen to come along.

```
\define@key{breqn}{cols}{\global\let\@preamble\@empty
```

```
  \darray@mkpream#1\@percentchar
}
```

FORMAT STATUS

```
\def\eq@frame{T}%
CLM works tolerably
 \def\eqindent{C}\def\eqnumside{L}\def\eqnumplace{M}
CLT works tolerably
 \def\eqindent{C}\def\eqnumside{L}\def\eqnumplace{T}
ILM
 \def\eqindent{I}\def\eqnumside{L}\def\eqnumplace{M}\mathindent40\p@
ILT
 \def\eqindent{I}\def\eqnumside{L}\def\eqnumplace{T}\mathindent40\p@
Indended w/left number
   work ok if mathindent is larger than number width,
   but then equations must fit into smaller space.
   Is shiftnumber allowed to put eqn at left, instead of indent?
CRM
 \def\eqindent{C}\def\eqnumside{R}\def\eqnumplace{M}
CRB
 \def\eqindent{C}\def\eqnumside{R}\def\eqnumplace{B}
IRM
 \def\eqindent{I}\def\eqnumside{R}\def\eqnumplace{M}\mathindent10\p@
IRB
 \def\eqindent{I}\def\eqnumside{R}\def\eqnumplace{B}\mathindent10\p@
```

The main environments.

BRM: The following incorporates several changes: 1) modifications supplied by MJD to fix the eaten `\paragraph` problem. 2) Added `\display@setup` here, rather than globally.

`\@dmath@start@hook`
`\@dgroup@start@hook`

```
\let\@dmath@start@hook\@empty
\let\@dgroup@start@hook\@empty
```

`\dmath`
`\enddmath`

For the `dmath` environment we don't want the standard optional arg processing because of the way it skips over whitespace, including newline, while looking for the `[` char; which is not good for math material. So we call `\@optarg` instead.

```
\newenvironment{dmath}{%
 \@dmath@start@hook
 \let\eq@hasNumber\@True \@optarg\@dmath{}}{}
\def\@dmath[#1]{%
⟨trace⟩  \breqn@debugmsg{=== DMATH ==================================================}%
  \everydisplay\expandafter{\the\everydisplay \display@setup}%
  \if@noskipsec \leavevmode \fi
  \if@inlabel \leavevmode \global\@inlabelfalse \fi
  \if\eq@group\else\eq@prelim\fi
  \setkeys{breqn}{#1}%
  \the\eqstyle
```

The equation number might have been overridden in `#1`.

```
  \eq@setnumber
```

Start up the displayed equation by reading the contents into a box register. Enclose this phase in an extra group so that modified `\hsize` and other params will be auto-restored afterwards.

```
  \begingroup
  \eq@setup@a
  \eq@startup
}
```

Before it finishes off the box holding the equation body, `\enddmath` needs to look ahead for punctuation (and `\qed`?).

```
\def\enddmath#1{%
  \check@punct@or@qed
}
\def\end@dmath{%
  \gdef\EQ@setwdL{}% Occasionally undefined ???
  \eq@capture
  \endgroup
  \EQ@setwdL
```

Measure (a copy of) the equation body to find the minimum width required to get acceptable line breaks, how many lines will be required at that width, and whether the equation number needs to be shifted to avoid overlapping. This information will then be used by `\eq@finish` to do the typesetting of the real equation body.

```
  \eq@measure
```

Piece together the equation from its constituents, recognizing current constraints. If we are in an equation group, this might just save the material on a stack for later processing.

```
  \if\eq@group \grp@push \else \eq@finish\fi
}
```

`\dmath*` `\enddmath*`

Ah yes, now the lovely `dmath*` environment.

```
\newenvironment{dmath*}{%
  \@dmath@start@hook
  \let\eq@hasNumber\@False \@optarg\@dmath{}%
}{}
\@namedef{end@dmath*}{\end@dmath}
\@namedef{enddmath*}#1{\check@punct@or@qed}
```

`\eq@prelim`

If `\everypar` has a non-null value, it's probably some code from `\@afterheading` that sets `\clubpenalty` and/or removes the parindent box. Both of those actions are irrelevant and interfering for our purposes and need to be deflected for the time being. If an equation appears at the very beginning of a list item (possibly from a trivlist such as `proof`), we need to trigger the item label.

```
\def\eq@prelim{%
  \if@inlabel \indent \par \fi
  \if@nobreak \global\@nobreakfalse \predisplaypenalty\@M \fi
  \everypar\@emptytoks
```

If for some reason dmath is called between paragraphs, \noindent is better than \leavevmode, which would produce an indent box and an empty line to hold it. If we are in a list environment, \par is defined as {\@@par} to preserve \parshape.

```
  \noindent
  \eq@nulldisplay
  \par %% \eq@saveparinfo %% needs work
  \let\intertext\breqn@intertext
}
```

breqn@parshape@warning Warning message extracted to a separate function to streamline the calling function.

```
\def\breqn@parshape@warning{%
  \PackageWarning{breqn}{%
    Complex paragraph shape cannot be followed by this equation}%
}
```

\eq@prevshape Storage; see \eq@saveparinfo.

```
\let\eq@prevshape\@empty
```

\eq@saveparinfo Save the number of lines and parshape info for the text preceding the equation.

```
\def\eq@saveparinfo{%
  \count@\prevgraf \advance\count@-\thr@@ % for the null display
  \edef\eq@prevshape{\prevgraf\the\count@\space}%
  \ifcase\parshape
    % case 0: no action required
  \or \edef\eq@prevshape{\eq@prevshape
        \parshape\@ne\displayindent\displaywidth\relax
      }%
```

Maybe best to set \eq@prevshape the same in the else case also. Better than nothing.

```
  \else
    \breqn@parshape@warning
  \fi
}
```

\eq@setnumber If the current equation number is not explicitly given, then use an auto-generated number, unless the no-number switch has been thrown (dmath*). \theequation is the number form to be used for all equations, \eq@number is the actual value for the current equation (might be an exception to the usual sequence).

```
\def\eq@setnumber{%
  \eq@wdNum\z@
  \if\eq@hasNumber
    \ifx\eq@number\@empty
      \stepcounter{equation}\let\eq@number\theequation
    \fi
%  \fi
```

This sets up numbox, etc, even if unnumbered?????

```
    \ifx\eq@number\@empty
    \else
```

Put the number in a box so we can use its measurements in our number-placement calculations. The extra braces around `\eqnumform` make it possible for `\eqnumfont` to have either an `\itshape` (recommended) or a `\textit` value.

```
⟨trace⟩       \breqn@debugmsg{Number \eq@number}%
        \set@label{equation}\eq@number
        \global\sbox\EQ@numbox{%
          \next@label \global\let\next@label\@empty
          \eqnumcolor\eqnumsize\eqnumfont{\eqnumform{\eq@number}}%
        }%
        \global\eq@wdNum\wd\EQ@numbox\global\advance\eq@wdNum\eqnumsep
%     \let\eq@hasNumber\@True % locally true
      \fi
    \fi
}
```

`\eq@finish` The information available at this point from preliminary measuring includes the number of lines required, the width of the equation number, the total height of the equation body, and (most important) the parshape spec that was used in determining height and number of lines.

Invoke the equation formatter for the requested centering/indentation having worked out the best parshape. BRM: This portion is extensively refactored to get common operations together (so corrections get consistently applied).

MH: I've destroyed Bruce's nice refactoring a bit to get the abovedisplayskips correct for both groups of equations and single `dmath` environments. I will have to redo that later.

```
\newcount\eq@final@linecount
\let\eq@GRP@first@dmath\@True
\def\eq@finish{%
  \begingroup
⟨trace⟩     \breqn@debugmsg{Formatting equation}%
⟨trace⟩     \debug@showmeasurements
    \if F\eq@frame\else
      \freeze@glue\eqlinespacing \freeze@glue\eqlineskip
    \fi
%     \eq@topspace{\vskip\parskip}% Set top spacing
    \csname eq@\eqindent @setsides\endcsname % Compute \leftskip,\rightskip
    \adjust@parshape\eq@parshape% Final adjustment of parshape for left|right skips
```

If we are in an a group of equations we don't want to calculate the top space for the first one as that will be delayed until later when the space for the group is calculated. However, we do need to store the leftskip used here as that will be used later on for calculating the top space.

```
    \if\eq@group
      \if\eq@GRP@first@dmath
        \global\let\eq@GRP@first@dmath\@False
        \xdef\dmath@first@leftskip{\leftskip=\the\leftskip\relax}%
⟨trace⟩ \breqn@debugmsg{Stored\space\dmath@first@leftskip}
      \else
        \eq@topspace{\vskip\parskip}% Set top spacing
      \fi
    \else
      \eq@topspace{\vskip\parskip}% Set top spacing
```

```
    \fi
⟨trace⟩    \debug@showformat
```

We now know the final line count of the display. If it is a single-line display, we want to know as that greatly simplifies the equation tag placement (until such a time where this algorithm has been straightened out).

```
    \afterassignment\remove@to@nnil
    \eq@final@linecount=\expandafter\@gobble\eq@parshape\@nnil
```

Now, invoke the appropriate typesetter according to number placement

```
    \if\eq@hasNumber
      \if\eq@shiftnumber
        \csname eq@typeset@\eqnumside Shifted\endcsname
      \else
```

If there is only one line and the tag doesn't have to be shifted, we call a special procedure to put the tag correctly.

```
        \ifnum\eq@final@linecount=\@ne
          \csname eq@typeset@\eqnumside @single\endcsname
        \else
          \csname eq@typeset@\eqnumside\eqnumplace\endcsname
        \fi
      \fi
    \else
      \eq@typeset@Unnumbered
    \fi
  \endgroup
  \eq@botspace
}
```

These are temporary until the tag position algorithm gets rewritten. At least the tag is positioned correctly for single-line displays. The horizontal frame position is not correct but the problem lies elsewhere.

```
\def\eq@typeset@L@single{%
  \nobreak
  \eq@params\eq@parshape
  \nointerlineskip\noindent
  \add@grp@label
  \rlap{\kern-\leftskip\box\EQ@numbox}%
  \if F\eq@frame
  \else
    \rlap{\raise\eq@firstht\hbox to\z@{\eq@addframe\hss}}%
  \fi
  \eq@dump@box\unhbox\EQ@box \@@par
}
\def\eq@typeset@R@single{%
  \nobreak
  \eq@params\eq@parshape
  \nointerlineskip\noindent
  \add@grp@label
  \if F\eq@frame
```

```
  \else
    \rlap{\raise\eq@firstht\hbox to\z@{\eq@addframe\hss}}%
  \fi
  \rlap{\kern-\leftskip\kern\linewidth\kern-\wd\EQ@numbox\copy\EQ@numbox}%
  \eq@dump@box\unhbox\EQ@box
  \@@par
}
```

27 Special processing for end-of-equation

At the end of a displayed equation environment we need to peek ahead for two things: following punction such as period or command that should be pulled in for inclusion at the end of the equation; and possibly also an `\end{proof}` with an implied "qed" symbol that is traditionally included at the end of the display rather than typeset on a separate line. We could require that the users type `\qed` explicitly at the end of the display when they want to have the display take notice of it. But the reason for doing that would only be to save work for the programmer; the most natural document markup would allow an inline equation and a displayed equation at the end of a proof to differ only in the environment name:

```
... \begin{math} ... \end{math}.
\end{proof}
```

versus

```
...
\begin{dmath}
 ...
\end{dmath}.
\end{proof}
```

. The technical difficulties involved in supporting this markup within LaTeX 2e are, admittedly, nontrivial. Nonetheless, let's see how far we can go.

The variations that we will support are only the most straightforward ones:

```
\end{dmath}.
\end{proof}
```

or

```
\end{dmath}.
Perhaps a comment
\end{proof}
```

. If there is anything more complicated than a space after the period we will not attempt to scan any further for a possible `\end{proof}`. This includes material such as:

```
\begin{figure}...\end{figure}%
\footnote{...}
\renewcommand{\foo}{...}
\par
```

or even a blank line—because in LaTeX a blank line is equivalent to `\par` and the meaning of `\par` is “end-paragraph”; in my opinion if explicit end-of-paragraph markup is given before the end of an element, it has to be respected, and the preceding paragraph has to be fully finished off before proceeding further, even inside an element like “proof” whose end-element formatting requires integration with the end of the paragraph text. And TeX nically speaking, a `\par` token that comes from a blank line and one that comes from the sequence of characters `\ p a r` are equally explicit. I hope to add support for `\footnote` in the future, as it seems to be a legitimate markup possibility in that context from a purely logical point of view, but there are additional technical complications if one wants to handle it in full generality

mjd,1999/02/08

.

`\peek@branch` This is a generalized “look at next token and choose some action based on it” function.

```
\def\peek@branch#1#2{%
  \let\peek@b#1\let\peek@space#2\futurelet\@let@token\peek@a
}
\def\peek@skipping@spaces#1{\peek@branch#1\peek@skip@space}
\def\peek@a{%
  \ifx\@let@token\@sptoken \expandafter\peek@space
  \else \expandafter\peek@b\fi
}
\lowercase{\def\peek@skip@space} {\futurelet\@let@token\peek@a}%
```

`\check@punct` For this one we need to recognize and grab for inclusion any of the following tokens: `,;.!?`, both catcode 12 (standard LaTeX value) and catcode 13 (as might hold when the Babel package is being used). We do not support a space preceding the punctuation since that would be considered simply invalid markup if a display-math environment were demoted to in-line math; and we want to keep their markup as parallel as possible. If punctuation does not follow, then the `\check@qed` branch is not applicable.

```
\def\check@punct{\futurelet\@let@token\check@punct@a}
\def\check@punct@a{%
  \edef\@tempa{%
    \ifx\@let@token\@sptoken\@nx\finish@end
    \else\ifx\@let@token ,\@nx\check@qed
    \else\ifx\@let@token .\@nx\check@qed
    \else\check@punct@b % check the less common possibilities
    \fi\fi\fi
  }%
  \@tempa
}
\begingroup
\toks@a{%
  \ifx\@let@token ;\@nx\check@qed
  \else\ifx\@let@token ?\@nx\check@qed
  \else\ifx\@let@token !\@nx\check@qed
}
\toks@c{\fi\fi\fi}% matching with \toks@a
```

```
\catcode`\.=\active \catcode`\,=\active \catcode`\;=\active
\catcode`\?=\active \catcode`\!=\active
\toks@b{%
  \else\ifx\@let@token ,\@nx\check@qed
  \else\ifx\@let@token .\@nx\check@qed
  \else\ifx\@let@token ;\@nx\check@qed
  \else\ifx\@let@token ?\@nx\check@qed
  \else\ifx\@let@token !\@nx\check@qed
  \else\@nx\finish@end
  \fi\fi\fi\fi\fi
}
\xdef\check@punct@b{%
  \the\toks@a\the\toks@b\the\toks@c
}
\endgroup
```

```
\let\found@punct\@empty
\def\check@qed#1{%
  \gdef\found@punct{#1}%
  \peek@skipping@spaces\check@qed@a
}
\def\check@qed@a{%
  \ifx\end\@let@token \@xp\check@qed@b
  \else \@xp\finish@end
  \fi
}
```

For each environment ENV that takes an implied qed at the end, the control sequence ENVqed must be defined; and it must include suitable code to yield the desired results in a displayed equation.

```
\def\check@qed@b#1#2{%
  \@ifundefined{#2qed}{}{%
    \toks@\@xp{\found@punct\csname#2qed\endcsname}%
    \xdef\found@punct{\the\toks@}%
  }%
  \finish@end
  \end{#2}%
}
```

`\latex@end` `\finish@end`

The lookahead for punctuation following a display requires mucking about with the normal operation of `\end`. Although this is not exactly something to be done lightly, on the other hand this whole package is so over-the-top anyway, what's a little more going to hurt? And rationalizing this aspect of equation markup is a worthy cause. Here is the usual definition of `\end`.

```
\def\end#1{
  \csname end#1\endcsname \@checkend{#1}%
  \expandafter\endgroup\if@endpe\@doendpe\fi
  \if@ignore \global\@ignorefalse \ignorespaces \fi
}
```

We can improve the chances of this code surviving through future minor changes in the fundamental definition of `\end` by taking a little care in saving the original meaning.

```
\def\@tempa#1\endcsname#2\@nil{\def\latex@end##1{#2}}
\expandafter\@tempa\end{#1}\@nil
\def\end#1{\csname end#1\endcsname \latex@end{#1}}%
```

Why don't we call `\CheckCommand` here? Because that doesn't help end users much; it works better to use it during package testing by the maintainer.

If a particular environment needs to call a different end action, the end command of the environment should be defined to gobble two args and then call a function like `\check@punct@or@qed`.

```
\def\check@punct@or@qed#1{%
  \xdef\found@punct{\@empty}% BRM: punctuation was being remembered past this eqn.
  % WSPR: err, why isn't that just \global\let\found@punct\@empty ?
  \def\finish@end{\csname end@#1\endcsname\latex@end{#1}}%
  \check@punct
}
```

`\eqpunct` User-settable function for handling the punctuation at the end of an equation. You could, for example, define it to just discard the punctuation.

```
\newcommand\eqpunct[1]{\thinspace#1}
```

`\set@label` `\set@label` just sets `\@currentlabel` but it takes the counter as an argument, in the hope that LaTeX will some day provide an improved labeling system that includes type info on the labels.

```
\providecommand\set@label[2]{\protected@edef\@currentlabel{#2}}
```

`\eq@topspace` `\eq@botspace` The action of `\eq@topspace` is complicated by the need to test whether the 'short' versions of the display skips should be used. This can be done only after the final parshape and indent have been determined, so the calls of this function are buried relatively deeply in the code by comparison to the calls of `\eq@botspace`. This also allows us to optimize slightly by setting the above-skip with `\parskip` instead of `\vskip`. `#1` is either `\noindent` or `\vskip\parskip`.

BRM: Hmm; we need to do *@setspace BEFORE this for small skips to work!

```
\def\eq@topspace#1{%
  \begingroup
    \global\let\EQ@shortskips\@False
```

If we are in `dgroup` or `dgroup*` and not before the top one, we just insert `\intereqskip`. Otherwise we must check for shortskip.

```
    \if\@And{\eq@group}{\@Not\eq@GRP@first@dmath}%
⟨trace⟩\breqn@debugmsg{Between lines}%
      \parskip\intereqskip \penalty\intereqpenalty
⟨trace⟩\breqn@debugmsg{parskip=\the\parskip}%
    \else
      \eq@check@shortskip
      \if\EQ@shortskips
        \parskip\abovedisplayshortskip
        \aftergroup\belowdisplayskip\aftergroup\belowdisplayshortskip
```

BRM: Not exactly TeX's approach, but seems right…

```
        \ifdim\predisplaysize>\z@\nointerlineskip\fi
      \else
        \parskip\abovedisplayskip
      \fi
    \fi
    \if F\eq@frame
    \else
      \addtolength\parskip{\eq@framesep+\eq@framewd}%
    \fi
⟨*trace⟩
    \breqn@debugmsg{Topspace: \theb@@le\EQ@shortskips, \parskip=\the\parskip,
      \predisplaysize=\the\predisplaysize}%
⟨/trace⟩
    #1%
  \endgroup
}
```

\eq@check@shortskip

```
\def\eq@check@shortskip {%
  \global\let\EQ@shortskips\@False
  \setlength\dim@a{\abovedisplayskip+\ht\EQ@numbox}%
```

Here we work around the hardwired standard TeX value and use the designer parameter instead.

```
  \ifdim\leftskip<\predisplaysize
  \else
```

If the display was preceeded by a blank line, `\predisplaysize` is −`\maxdimen` and so we should insert a fairly large skip to separate paragraphs, i.e., no short skip. Perhaps this should be a third parameter `\abovedisplayparskip`.

```
    \ifdim -\maxdimen=\predisplaysize
    \else
      \if R\eqnumside
        \global\let\EQ@shortskips\@True
      \else
        \if\eq@shiftnumber
        \else
          \if T\eqnumplace
            \ifdim\dim@a<\eq@firstht
              \global\let\EQ@shortskips\@True
            \fi
          \else
            \setlength\dim@b{\eq@vspan/2}%
            \ifdim\dim@a<\dim@b
              \global\let\EQ@shortskips\@True
            \fi
          \fi
        \fi
      \fi
    \fi
```

```
  \fi
}
```

At the end of an equation, need to put in a pagebreak penalty and some vertical space. Also set some flags to remove parindent and extra word space if the current paragraph text continues without an intervening `\par`.

```
\def\eq@botspace{%
  \penalty\postdisplaypenalty
```

Earlier calculations will have set `\belowdisplayskip` locally to `\belowdisplayshortskip` if applicable. So we can just use it here.

```
  \if F\eq@frame
  \else
    \addtolength\belowdisplayskip{\eq@framesep+\eq@framewd}%
  \fi
  \vskip\belowdisplayskip
  \@endpetrue % kill parindent if current paragraph continues
  \global\@ignoretrue % ignore following spaces
  \eq@resume@parshape
}
```

`\eq@resume@parshape` This should calculate the total height of the equation, including space above and below, and set prevgraf to the number it would be if that height were taken up by normally-spaced normal-height lines. We also need to restore parshape if it had a non-null value before the equation. Not implemented yet.

```
\def\eq@resume@parshape{}
```

28 Preprocessing the equation body

`\eq@startup` Here is the function that initially collects the equation material in a box.

```
\def\eq@startup{%
  \global\let\EQ@hasLHS\@False
  \setbox\z@\vbox\bgroup
    \noindent \@@math \displaystyle
    \penalty-\@Mi
}
```

This setup defines the environment for the first typesetting pass, note the `\hsize` value for example.

```
\def\eq@setup@a{%
  \everymath\everydisplay
  %\let\@newline\eq@newline % future possibility?
  \let\\\eq@newline
  \let\insert\eq@insert \let\mark\eq@mark \let\vadjust\eq@vadjust
  \hsize\maxdimen \pretolerance\@M
```

Here it is better not to use `\@flushglue` (0pt plus1fil) for `\rightskip`, or else a negative penalty (such as −99 for `\prerelpenalty`) will tempt TEX to use more line breaks than necessary in the first typesetting pass. Ideal values for `\rightskip` and `\linepenalty` are

The contents of an equation after the initial typesetting pass, as shown by `\showlists`. This is the material on which the `\eq@repack` function operates. The equation was

```
a=b +\left(\frac{c\sp 2}{2} -d\right) +(e -f) +g
```

. The contents are shown in four parts in this figure and the next three. The first part contains two line boxes, one for the mathon node and one for the LHS.

```
\hbox(0.0+0.0)x16383.99998, glue set 1.6384
.\mathon
.\penalty -10000
.\glue(\rightskip) 0.0 plus 10000.0
\penalty 1
\glue(\baselineskip) 7.69446
\hbox(4.30554+0.0)x16383.99998, glue set 1.63759
.\OML/cmm/m/it/10 a
.\glue 2.77771 minus 1.11108
.\penalty -10001
.\glue(\rightskip) 0.0 plus 10000.0
\penalty 2
\glue(\lineskip) 1.0
...
```

Figure 1: Preliminary equation contents, part 1

unclear to me, but they are rather sensitively interdependent. Choice of 10000 pt for rightskip is derived by saying, let's use a value smaller than 1 fil and smaller than `\hsize`, but more than half of `\hsize` so that if a line is nearly empty, the glue stretch factor will always be less than 2.0 and so the badness will be less than 100 and so TEX will not issue badness warnings.

```
  \linepenalty\@m
  \rightskip\z@\@plus\@M\p@ \leftskip\z@skip \parfillskip\z@skip
  \clubpenalty\@ne \widowpenalty\z@ \interlinepenalty\z@
```

After a relation symbol is discovered, binop symbols should start including a special offset space. But until then `\EQ@prebin@space` is a no-op.

```
  \global\let\EQ@prebin@space\relax
```

Set binoppenalty and relpenalty high to prohibit line breaks after mathbins and mathrels. As a matter of fact, the penalties are then omitted by TEX, since bare glue without a penalty is *not* a valid breakpoint if it occurs within mathon–mathoff items.

```
  \binoppenalty\@M \relpenalty\@M
}
```

`\eq@capture` `\eq@punct` If an equation ends with a `\right` delim, the last thing on the math list will be a force-break penalty. Then don't redundantly add another forcing penalty. (question: when does a penalty after a linebreak not disappear? Answer: when you have two forced break penalties in a row). Ending punctuation, if any, goes into the last box with the mathoff kern. If the math list ends

This is the first part of the RHS, up to the `\right`, where a line break has been forced so that we can break open the left-right box.

```
...
\penalty 2
\glue(\lineskip) 1.0
\hbox(14.9051+9.50012)x16383.99998, glue set 1.63107
.\penalty -99
.\glue(\thickmuskip) 2.77771 minus 1.11108
.\OT1/cmr/m/n/10 =
.\glue(\thickmuskip) 2.77771 minus 1.11108
.\OML/cmm/m/it/10 b
.\penalty 888
.\glue -10.5553
.\rule(*+*)x0.0
.\penalty 10000
.\glue 10.5553
.\glue(\medmuskip) 2.22217 minus 1.66663
.\OT1/cmr/m/n/10 +
.\glue(\medmuskip) 2.22217 minus 1.66663
.\hbox(14.9051+9.50012)x43.36298
..\hbox(0.39998+23.60025)x7.36115, shifted -14.10013
...\OMX/cmex/m/n/5 \hat \hat R
..\hbox(14.9051+6.85951)x11.21368
...\hbox(14.9051+6.85951)x11.21368
  ... [fraction contents, elided]
..\penalty 5332
..\glue -10.5553
..\rule(*+*)x0.0
..\penalty 10000
..\glue 10.5553
..\glue(\medmuskip) 2.22217 minus 1.66663
..\OMS/cmsy/m/n/10 \hat \hat @
..\glue(\medmuskip) 2.22217 minus 1.66663
..\OML/cmm/m/it/10 d
..\hbox(0.39998+23.60025)x7.36115, shifted -14.10013
...\OMX/cmex/m/n/5 \hat \hat S
.\penalty -10000
.\glue(\rightskip) 0.0 plus 10000.0
\penalty 3
\glue(\lineskip) 1.0
...
```

Figure 2: Preliminary equation contents, part 2

This is the remainder of the RHS after the post-`\right` split.

```
...
\penalty 3
\glue(\lineskip) 1.0
\hbox(7.5+2.5)x16383.99998, glue set 1.63239
.\penalty 888
.\glue -10.5553
.\rule(*+*)x0.0
.\penalty 10000
.\glue 10.5553
.\glue(\medmuskip) 2.22217 minus 1.66663
.\OT1/cmr/m/n/10 +
.\glue(\medmuskip) 2.22217 minus 1.66663
.\OT1/cmr/m/n/10 (
.\OML/cmm/m/it/10 e
.\penalty 5332
.\glue -10.5553
.\rule(*+*)x0.0
.\penalty 10000
.\glue 10.5553
.\glue(\medmuskip) 2.22217 minus 1.66663
.\OMS/cmsy/m/n/10 \hat \hat @
.\glue(\medmuskip) 2.22217 minus 1.66663
.\OML/cmm/m/it/10 f
.\kern1.0764
.\OT1/cmr/m/n/10 )
.\penalty 888
.\glue -10.5553
.\rule(*+*)x0.0
.\penalty 10000
.\glue 10.5553
.\glue(\medmuskip) 2.22217 minus 1.66663
.\OT1/cmr/m/n/10 +
.\glue(\medmuskip) 2.22217 minus 1.66663
.\OML/cmm/m/it/10 g
.\kern0.35878
.\penalty -10000
.\glue(\rightskip) 0.0 plus 10000.0
\glue(\baselineskip) 9.5
...
```

Figure 3: Preliminary equation contents, part 3

This is the mathoff fragment.

```
...
\glue(\baselineskip) 9.5
\hbox(0.0+0.0)x16383.99998, glue set 1.6384
.\mathoff
.\penalty 10000
.\glue(\parfillskip) 0.0
.\glue(\rightskip) 0.0 plus 10000.0
```

Figure 4: Preliminary equation contents, part 4

with a slanted letter, then there will be an italic correction added after it by TEX. Should we remove it? I guess so.

28.1 Capturing the equation

BRM: There's a problem here (or with `\ss@scan`). If the LHS has `\left \right` pairs, ß@scan gets involved. It seems to produce a separate box marked w/`\penalty` 3. But it appears that `\eq@repack` is only expecting a single box for the LHS; when it measures that box it's missing the (typically larger) bracketted section, so the LHS is measured =¿ 0pt (or very small). I'm not entirely clear what Michael had in mind for this case; whether it's an oversight, or whether I've introduced some other bug. At any rate, my solution is to measure the RHS (accumulated in `\EQ@box`), at the time of the relation, and subtract that from the total size.

```
\newdimen\eq@wdR\eq@wdR\z@%BRM
\def\eq@capture{%
  \ifnum\lastpenalty>-\@M \penalty-\@Mi \fi
```

We want to keep the mathoff kern from vanishing at the line break, so that we can reuse it later.

```
  \keep@glue\@@endmath
  \eq@addpunct
  \@@par
  \eq@wdL\z@
```

First snip the last box, which contains the mathoff node, and put it into `\EQ@box`. Then when we call `\eq@repack` it will recurse properly.

```
  \setbox\tw@\lastbox
  \global\setbox\EQ@box\hbox{\unhbox\tw@\unskip\unskip\unpenalty}%
  \unskip\unpenalty
  \global\setbox\EQ@copy\copy\EQ@box
%%  \global\setbox\EQ@vimcopy\copy\EQ@vimbox
  \clubpenalty\z@
%\batchmode\showboxbreadth\maxdimen\showboxdepth99\showlists\errorstopmode
  \eq@wdR\z@%BRM: eq@wdL patch
  \eq@repack % recursive
```

Finally, add the mathon item to `\EQ@box` and `\EQ@copy`.

```
  \setbox\tw@\lastbox
```

```
  \global\setbox\EQ@box\hbox{\unhcopy\tw@\unskip\unpenalty \unhbox\EQ@box}%
  \global\setbox\EQ@copy\hbox{\unhbox\tw@\unskip\unpenalty \unhbox\EQ@copy}%
%\batchmode\showbox\EQ@copy \showthe\eq@wdL\errorstopmode
  \ifdim\eq@wdR>\z@% BRM:  eq@wdL patch
    \setlength\dim@a{\wd\EQ@box-\eq@wdR
    % Apparently missing a \thickmuskip = 5mu = 5/18em=0.27777777777.. ?
       + 0.2777777777777em}% FUDGE??!?!?!
    \ifdim\dim@a>\eq@wdL
⟨*trace⟩
      \breqn@debugmsg{Correcting LHS from \the\eq@wdL\space to
                 \the\dim@a = \the\wd\EQ@box - \the\eq@wdR}%
⟨/trace⟩
      \eq@wdL\dim@a
      \xdef\EQ@setwdL{\eq@wdL\the\eq@wdL\relax}%
    \fi
  \fi
⟨*trace⟩
  \breqn@debugmsg{Capture: total length=\the\wd\EQ@box \MessageBreak
           ==== has LHS=\theb@@le\EQ@hasLHS, \eq@wdL=\the\eq@wdL, \eq@wdR=\the\eq@wdR,
           \MessageBreak
           ==== \eq@wdCond=\the\eq@wdCond}%
⟨/trace⟩
  \egroup % end vbox started earlier
⟨*trace⟩
%\debugwr{EQ@box}\debug@box\EQ@box
%\debugwr{EQ@copy}\debug@box\EQ@copy
⟨/trace⟩
}
```

Now we have two copies of the equation, one in `\EQ@box`, and one in `\EQ@copy` with inconvenient stuff like inserts and marks omitted.

`\eq@addpunct` is for tacking on text punctuation at the end of a display, if any was captured by the 'gp' lookahead.

```
\def\eq@addpunct{%
  \ifx\found@punct\@empty
  \else \eqpunct{\found@punct}%
  \fi
  % BRM: Added; the punctuation kept  getting carried to following environs
  \xdef\found@punct{\@empty}%
  \EQ@afterspace
}
```

Needed for the `dseries` environment, among other things.

```
\global\let\EQ@afterspace\@empty
```

`\eq@repack` The `\eq@repack` function looks at the information at hand and proceeds accordingly.

TeX Note: this scans BACKWARDS from the end of the math.

```
\def\eq@repack{%
% A previous penalty of 3 on the vertical list means that we need
% to break open a left-right box.
%    \begin{macrocode}
```

```
  \ifcase\lastpenalty
     % case 0: normal case
    \setbox\tw@\lastbox
    \eq@repacka\EQ@copy \eq@repacka\EQ@box
    \unskip
  \or % case 1: finished recursing
```

Grab the mathon object since we need it to inhibit line breaking at bare glue nodes later.

```
    \unpenalty
    \setbox\tw@\lastbox
    \eq@repacka\EQ@copy \eq@repacka\EQ@box
    \@xp\@gobble
  \or % case 2: save box width = LHS width
```

Don't need to set `\EQ@hasLHS` here because it was set earlier if applicable.

```
    \unpenalty
    \setbox\tw@\lastbox
    \setbox\z@\copy\tw@ \setbox\z@\hbox{\unhbox\z@\unskip\unpenalty}%
    \addtolength\eq@wdL{\wd\z@}
    \setlength\eq@wdR{\wd\EQ@box}% BRM:  eq@wdL patch
    \xdef\EQ@setwdL{\eq@wdL\the\eq@wdL\relax}%
```

At this point, box 2 typically ends with

```
.\mi10 a
.\glue 2.77771 plus 2.77771
.\penalty -10001
.\glue(\rightskip) 0.0 plus 10000.0
```

and we want to ensure that the thickmuskip glue gets removed. And we now arrange for `\EQ@copy` and `\EQ@box` to keep the LHS in a separate subbox; this is so that we can introduce a different penalty before the first relation symbol if necessary, depending on the layout decisions that are made later.

```
    \global\setbox\EQ@copy\hbox{%
      \hbox{\unhcopy\tw@\unskip\unpenalty\unskip}%
      \box\EQ@copy
    }%
    \global\setbox\EQ@box\hbox{%
      \hbox{\unhbox\tw@\unskip\unpenalty\unskip}%
      \box\EQ@box
    }%
    \unskip
  \or % case 3: unpack left-right box
    \unpenalty
    \eq@lrunpack
  \else
    \breqn@repack@err
  \fi
  \eq@repack % RECURSE
}
```

Error message extracted to streamline calling function.

```
\def\breqn@repack@err{%
  \PackageError{breqn}{eq@repack penalty neq 0,1,2,3}\relax
}
```

`\eq@repacka` We need to transfer each line into two separate boxes, one containing everything and one that omits stuff like `\inserts` that would interfere with measuring.

```
\def\eq@repacka#1{%
  \global\setbox#1\hbox{\unhcopy\tw@ \unskip
    \count@-\lastpenalty
    \ifnum\count@<\@M \else \advance\count@-\@M \fi
    \unpenalty
```

If creating the measure copy, ignore all cases above case 3 by folding them into case 1.

```
    \ifx\EQ@copy#1\ifnum\count@>\thr@@ \count@\@ne\fi\fi
    \ifcase\count@
        % case 0, normal line break
      \penalty-\@M % put back the linebreak penalty
    \or % case 1, do nothing (end of equation)
      \relax
    \or % case 2, no-op (obsolete case)
    \or % case 3, transfer vspace and/or penalty
      \ifx#1\EQ@box \eq@revspace \else \eq@revspaceb \fi
    \or % case 4, put back an insert
      \eq@reinsert
    \or % case 5, put back a mark
      \eq@remark
    \or % case 6, put back a vadjust
      \eq@readjust
    \else % some other break penalty
      \penalty-\count@
    \fi
    \unhbox#1}%
}
```

`\eq@nulldisplay` Throw in a null display in order to get predisplaysize etc.. My original approach here was to start the null display, then measure the equation, and set a phantom of the equation's first line before ending the null display. That would allow finding out if TEX used the short displayskips instead of the normal ones. But because of some complications with grouping and the desirability of omitting unnecessary invisible material on the vertical list, it seems better to just collect information about the display (getting `\prevdepth` requires `\halign`) and manually perform our own version of TEX's shortskip calculations. This approach also gives greater control, e.g.,, the threshold amount of horizontal space between predisplaysize and the equation's left edge that determines when the short skips kick in becomes a designer-settable parameter rather than hardwired into TEX.

```
\def\eq@nulldisplay{%
  \begingroup \frozen@everydisplay\@emptytoks
  \@@display
  \predisplaypenalty\@M \postdisplaypenalty\@M
  \abovedisplayskip\z@skip \abovedisplayshortskip\z@skip
  \belowdisplayskip\z@skip \belowdisplayshortskip\z@skip
```

```
  \xdef\EQ@displayinfo{%
    \prevgraf\the\prevgraf \predisplaysize\the\predisplaysize
    \displaywidth\the\displaywidth \displayindent\the\displayindent
    \listwidth\the\linewidth
```

Not sure how best to test whether leftmargin should be added. Let's do this for now [mjd,1997/10/08].

```
    \ifdim\displayindent>\z@
      \advance\listwidth\the\leftmargin
      \advance\listwidth\the\rightmargin
    \fi
    \relax}%
```

An `\halign` containing only one `\cr` (for the preamble) puts no box on the vertical list, which means that no `\baselineskip` will be added (so we didn't need to set it to zero) and the previous value of prevdepth carries through. Those properties do not hold for an empty simple equation without `\halign`.

```
  \halign{##\cr}%
  \@@enddisplay
  \par
  \endgroup
  \EQ@displayinfo
}
```

`\eq@newline` `\eq@newlinea` `\eq@newlineb`

Here we use `\@ifnext` so that in a sequence like

```
...\\
[a,b]
```

LaTeX does not attempt to interpret the `[a,b]` as a vertical space amount. We would have used `\eq@break` in the definition of `\eq@newlineb` except that it puts in a `\keep@glue` object which is not such a good idea if a mathbin symbol follows—the indent of the mathbin will be wrong because the leading negative glue will not disappear as it should at the line break.

```
\def\eq@newline{%
  \@ifstar{\eq@newlinea\@M}{\eq@newlinea\eqinterlinepenalty}}
\def\eq@newlinea#1{%
  \@ifnext[{\eq@newlineb{#1}}{\eq@newlineb{#1}[\maxdimen]}}
\def\eq@newlineb#1[#2]{\penalty-\@M}
```

`\eq@revspace` `\eq@revspaceb`

When `\eq@revspace` (re-vspace) is called, we are the end of an equation line; we need to remove the existing penalty of -10002 in order to put a vadjust object in front of it, then put back the penalty so that the line break will still take place in the final result.

```
\def\eq@revspace{%
  \global\setbox\EQ@vimbox\vbox{\unvbox\EQ@vimbox
    \unpenalty
    \global\setbox\@ne\lastbox}%
  \@@vadjust{\unvbox\@ne}%
  \penalty-\@M
}
```

needs work

Figure 5: first-approximation parshape for equations

The b version is used for the `\EQ@copy` box.

```
\def\eq@revspaceb{%
  \global\setbox\EQ@vimcopy\vbox{\unvbox\EQ@vimcopy
    \unpenalty
    \global\setbox\@ne\lastbox}%
  \@@vadjust{\unvbox\@ne}%
  \penalty-\@M
}
```

`\eq@break` The function `\eq@break` does a preliminary linebreak with a flag penalty.

```
\def\eq@break#1{\penalty-1000#1 \keep@glue}
```

29 Choosing optimal line breaks

The question of what line width to use when breaking an equation into several lines is best examined in the light of an extreme example. Suppose we have a two-column layout and a displayed equation falls inside a second-level list with nonzero leftmargin and rightmargin. Then we want to try in succession a number of different possibilities. In each case if the next possibility is no wider than the previous one, skip ahead to the one after.

1. First try linewidth(2), the linewidth for the current level-2 list.
2. If we cannot find adequate linebreaks at that width, next try listwidth(2), the sum of leftmargin, linewidth, and rightmargin for the current list.
3. If we cannot find linebreaks at that width, next try linewidth (1) (skipping this step if it is no larger then listwidth(2)).
4. If we cannot find linebreaks at that width, next try listwidth(1).
5. If we cannot find linebreaks at that width, next try column width.
6. If we cannot find linebreaks at that width, next try text width.
7. If we cannot find linebreaks at that width, next try equation width, if it exceeds text width (i.e.,, if the style allows equations to extend into the margins).

At any given line width, we run through a series of parshape trials and, essentially, use the first one that gives decent line breaks. But the process is a bit more complicated in fact. In order to do a really good job of setting up the parshapes, we need to know how many lines the equation will require. And of course the number of lines needed depends on the parshape! So as our very first trial we run a simple first-approximation parshape (Figure 5) whose main purpose is to get an estimate on the number of lines that will be needed; it chooses a uniform indent for all lines after the first one and does not take any account of the equation number. A substantial majority of equations only require one line anyway, and for them this first trial will succeed. In the one-line case if there is an equation number and it doesn't fit on the same line as the equation body, we don't go on to other trials because breaking up the equation body will not gain us anything—we know that we'll have to use two lines in any case—so we might as well keep the equation body together on one line and shift the number to a separate line.

If we learn from the first trial that the equation body requires more than one line, the next parshape trial involves adjusting the previous parshape to leave room for the equation number, if present. If no number is present, again no further trials are needed.

Some remarks about parshape handling. The TeX primitive doesn't store the line specs anywhere, `\the\parshape` only returns the number of line specs. This makes it well nigh impossible for different packages that use `\parshape` to work together. Not that it would be terribly easy for the package authors to make inter-package collaboration work, if it were possible. If we optimistically conjecture that someone some day may take on such a task, then the thing to do, obviously, is provide a parshape interface that includes a record of all the line specs. For that we designate a macro `\@parshape` which includes not only the line specs, but also the line count and even the leading `\parshape` token. This allows it to be directly executed without an auxiliary if-empty test. It should include a trailing `\relax` when it has a nonempty value.

```
\let\@parshape\@empty
```

The function `\eq@measure` runs line-breaking trials on the copy of the equation body that is stored in the box register `\EQ@copy`, trying various possible layouts in order of preference until we get successful line breaks, where 'successful' means there were no overfull lines. The result of the trials is, first, a parshape spec that can be used for typesetting the real equation body in `\EQ@box`, and second, some information that depends on the line breaks such as the depth of the last line, the height of the first line, and positioning information for the equation number. The two main variables in the equation layout are the line width and the placement of the equation number, if one is present.

`\eq@measure` Run linebreak trials on the equation contents and measure the results.

```
\def\eq@measure{%
```

If an override value is given for indentstep in the env options, use it.

```
  \ifdim\eq@indentstep=\maxdimen \eq@indentstep\eqindentstep \fi
```

If `\eq@linewidth` is nonzero at this point, it means that the user specified a particular target width for this equation. In that case we override the normal list of trial widths.

```
  \ifdim\eq@linewidth=\z@ \else \edef\eq@linewidths{{\the\eq@linewidth}}\fi
  \begingroup \eq@params
  \leftskip\z@skip
```

Even if `\hfuzz` is greater than zero a box whose contents exceed the target width by less then hfuzz still has a reported badness value of 1000000 (infinitely bad). Because we use inf-bad to test whether a particular trial succeeds or fails, we want to make such boxes return a smaller badness. To this end we include an `\hfuzz` allowance in `\rightskip`. In fact, `\eq@params` ensures that `\hfuzz` for equations is at least 1pt.

```
  \rightskip\z@\@plus\columnwidth\@minus\hfuzz
%  \eqinfo
  \global\EQ@continue{\eq@trial}%
```

```
  \eq@trial % uses \eq@linewidths
  \eq@failout % will be a no-op if the trial succeeded
  \endgroup
```

'local' parameter settings are passed outside the endgroup through `\EQ@trial`.

```
  \EQ@trial
}
```

```
⟨*trace⟩
\def\debug@showmeasurements{%
  \breqn@debugmsg{=> \number\eq@lines\space lines}%
  \begingroup
  \def\@elt##1X##2{\MessageBreak==== \space\space##1/##2}%
  \let\@endelt\@empty
  \breqn@debugmsg{=> trial info:\eq@measurements}%
  \breqn@debugmsg{=> bounding box: \the\eq@wdT x\the\eq@vspan, badness=\the\eq@badness}%
  \let\@elt\relax \let\@endelt\relax
  \endgroup
}
\def\debug@showmeasurements{%
  \begingroup
  \def\@elt##1X##2{\MessageBreak====   ##1/##2}%
  \let\@endelt\@empty
  \breqn@debugmsg{===> Measurements: \number\eq@lines\space lines
            \eq@measurements
            \MessageBreak
            ==== bounding box: \the\eq@wdT x\the\eq@vspan, badness=\the\eq@badness
            \MessageBreak
            ==== \leftskip=\the\leftskip, \rightskip=\the\rightskip}%
 \endgroup
}
⟨/trace⟩
```

Layout Trials Driver Basically, trying different sequences of parshapes.

`\EQ@trial` Init.

```
\let\EQ@trial\@empty
```

`\EQ@continue` This is a token register used to carry trial info past a group boundary with only one global assignment.

```
\newtoks\EQ@continue
```

`\EQ@widths` This is used for storing the actual line-width info of the equation contents after breaking.

```
\let\EQ@widths\@empty
```

`\EQ@fallback`

```
\let\EQ@fallback\@empty
```

`\eq@linewidths` This is the list of target widths for line breaking.

=== BRM: Odd; I don't think I've seen this use anything but `\displaywidth`...

```
\def\eq@linewidths{\displaywidth\linewidth\columnwidth}
```

`\eq@trial` The `\eq@trial` function tries each candidate line width in `\eq@linewidths` until an equation layout is found that yields satisfactory line breaks.

```
\def\eq@trial{%
  \ifx\@empty\eq@linewidths
    \global\EQ@continue{}%
  \else
    \iffalse{\fi \@xp\eq@trial@a \eq@linewidths}%
  \fi
  \the\EQ@continue
}
```

`\eq@trial@a` The `\eq@trial@a` function reads the leading line width from `\eq@linewidths`; if the new line width is greater than the previous one, start running trials with it; otherwise do nothing with it. Finally, run a peculiar `\edef` that leaves `\eq@linewidths` redefined to be the tail of the list. If we succeed in finding satisfactory line breaks for the equation, we will reset `\EQ@continue` in such a way that it will terminate the current trials. An obvious branch here would be to check whether the width of `\EQ@copy` is less than `\eq@linewidth` and go immediately to the one-line case if so. However, if the equation contains more than one RHS, by default each additional RHS starts on a new line—i.e.,, we want the ladder layout anyway. So we choose the initial trial on an assumption of multiple lines and leave the one-line case to fall out naturally at a later point.

```
\def\eq@trial@a#1{%
  \dim@c#1\relax
  \if T\eq@frame \eq@frame@adjust\dim@c \fi
  \ifdim\dim@c>\eq@linewidth
    \eq@linewidth\dim@c
⟨trace⟩    \breqn@debugmsg{Choose Shape for width(#1)=\the\eq@linewidth}%
    \let\eq@trial@b\eq@trial@d
    \csname eq@try@layout@\eq@layout\endcsname
⟨trace⟩  \else
⟨trace⟩    \breqn@debugmsg{Next width (#1) is shorter; skip it}%
  \fi
  \edef\eq@linewidths{\iffalse}\fi
}
\def\eq@frame@adjust#1{%
  %\addtolength#1{-2\eq@framewd-2\eq@framesep}%
  \dim@a\eq@framewd \advance\dim@a\eq@framesep
  \advance#1-2\dim@a
}
```

== Note curious control structure. Try to understand interaction of `\EQ@fallback`, `\EQ@continue`, `\eq@failout`

```
\def\eq@trial@succeed{%
  \aftergroup\@gobbletwo % cancel the \EQ@fallback code; see \eq@trial@c (?)
  \global\EQ@continue{\eq@trial@done}%
}
```

`\eq@trial@done` Success.

```
\def\eq@trial@done{%
```

```
⟨trace⟩  \breqn@debugmsg{End trial: Success!}%
  \let\eq@failout\relax
}
```

`\eq@trial@init` This is called from `\eq@trial@b` to initialize or re-initialize certain variables as needed when running one or more trials at a given line width. By default assume success, skip the fallback code.

```
\def\eq@trial@init{\global\let\EQ@fallback\eq@nextlayout}
```

`\eq@nextlayout` In the fallback case cancel the current group to avoid unnecessary group nesting (with associated save-stack cost, etc.).

```
\def\eq@nextlayout#1{%
  \endgroup
⟨trace⟩  \breqn@debugmsg{Nope ... that ain't gonna work.}%
  \begingroup #1%
}
```

`\eq@failout` .

```
\def\eq@failout{%
⟨trace⟩\breqn@debugmsg{End trial: failout}%
  \global\let\EQ@trial\EQ@last@trial
}
```

`\eq@trial@save` Save the parameters of the current trial.

```
\def\eq@trial@save#1{%
⟨*trace⟩
%  \begingroup \def\@elt##1X##2{\MessageBreak==== \space\space##1/##2}\let\@endelt\@empty\breqn@debugmsg
%         \breqn@debugmsg{=> bounding box: \the\eq@wdT x\the\eq@vspan, badness=\the\eq@badness\MessageBr
%         \let\@elt\relax \let\@endelt\relax
%  \endgroup
⟨/trace⟩
  \xdef#1{%
    \eq@linewidth\the\eq@linewidth
    % save info about the fit
    \eq@lines\the\eq@lines \eq@badness\the\eq@badness \def\@nx\eq@badline{\eq@badline}%
    % save size info
    \eq@wdT\the\eq@wdT \eq@wdMin\the\eq@wdMin
    \eq@vspan\the\eq@vspan \eq@dp\the\eq@dp \eq@firstht\the\eq@firstht
    % save info about the LHS
    \eq@wdL\the\eq@wdL \def\@nx\EQ@hasLHS{\EQ@hasLHS}%
    % save info about the numbering
    \def\@nx\eq@hasNumber{\eq@hasNumber}%
    % save info about the chosen layout
    \def\@nx\eq@layout{\eq@layout}%
    \def\@nx\eq@parshape{\@parshape}%
    \def\@nx\eq@measurements{\eq@measurements}%
    \def\@nx\adjust@rel@penalty{\adjust@rel@penalty}%
    \def\@nx\eq@shiftnumber{\eq@shiftnumber}%
    \def\@nx\eq@isIntertext{\@False}%
  }%
```

```
}
```

\eq@trial@b By default this just runs \eq@trial@c; cf. \eq@trial@d.

```
\def\eq@trial@b{\eq@trial@c}
```

\eq@trial@c Run the equation contents through the current parshape.

```
\def\eq@trial@c#1#2{%
⟨trace⟩  \breqn@debugmsg{Trying layout "#1" with\MessageBreak==== parshape\space\@xp\@gobble\@parshape}%
  \begingroup
  \eq@trial@init
  \def\eq@layout{#1}%
  \setbox\z@\vbox{%
    \hfuzz\maxdimen
    \eq@trial@p % run the given parshape
    \if\@Not{\eq@badline}%
      \eq@trial@save\EQ@trial
```

If there is a number, try the same parshape again with adjustments to make room for the number.

This is an awkward place for this: It only allows trying to fit the number w/the SAME layout shape!

```
      \if\eq@hasNumber\eq@retry@with@number\fi
      \if L\eq@layout \eq@check@density
      \else
        \if\@Not{\eq@badline}%
           \eq@trial@succeed
        \fi
      \fi
    \else
      \eq@trial@save\EQ@last@trial
    \fi
  }%
  \EQ@fallback{#2}%
  \endgroup
}
```

\eq@trial@d

```
\def\eq@trial@d#1#2{\eq@trial@c{#1}{}}
```

\eq@check@density

```
\def\eq@check@density{%
⟨trace⟩  \breqn@debugmsg{Checking density for layout L}%
  \if\@Or{\@Not\EQ@hasLHS}{\eq@shortLHS}%
⟨trace⟩    \breqn@debugmsg{Density check: No LHS, or is short; OK}%
    \eq@trial@succeed
  \else\if\eq@dense@enough
    \eq@trial@succeed
  \fi\fi
}
```

`\eq@shortLHS` Test to see if we need to apply the `\eq@dense@enough` test.

```
\def\eq@shortLHS{\ifdim\eq@wdL>.44\eq@wdT 1\else 0\fi 0}
```

`\def\eq@shortLHS{\@False}` ==

`\eq@trial@p` Run a trial with the current `\@parshape` and measure it.

```
\def\eq@trial@p{%
  \@parshape %
  \eq@dump@box\unhcopy\EQ@copy
  {\@@par}% leave \parshape readable
  \eq@lines\prevgraf
  \eq@fix@lastline
  \let\eq@badline\@False
  \if i\eq@layout \ifnum\eq@lines>\@ne \let\eq@badline\@True \fi\fi
  \eq@curline\eq@lines % loop counter for eq@measure@lines
  \let\eq@measurements\@empty
  \eq@ml@record@indents
  \eq@measure@lines
  \eq@recalc
⟨trace⟩  \debug@showmeasurements
}
```

`\adjust@rel@penalty` Normally this is a no-op.

```
\let\adjust@rel@penalty\@empty
```

`\eq@fix@lastline` Remove parfillskip from the last line box.

```
\def\eq@fix@lastline{%
  \setbox\tw@\lastbox \dim@b\wd\tw@
  \eq@dp\dp\tw@
```

Remove `\parfillskip` but retain `\rightskip`. Need to keep the original line width for later shrink testing.

```
  \nointerlineskip\hbox to\dim@b{\unhbox\tw@
    \skip@c\lastskip \unskip\unskip\hskip\skip@c
  }%
}
```

`\eq@recalc` Calculate `\eq@wdT` et cetera.

```
\def\eq@recalc{%
  \eq@wdT\z@ \eq@wdMin\maxdimen \eq@vspan\z@skip \eq@badness\z@
  \let\@elt\eq@recalc@a \eq@measurements \let\@elt\relax
}
```

`\eq@recalc@a`

```
\def\eq@recalc@a#1x#2+#3\@endelt{%
  \eq@firstht#2\relax
  \let\@elt\eq@recalc@b
  \@elt#1x#2+#3\@endelt
}
```

\eq@recalc@b

```
\def\eq@recalc@b#1X#2,#3x#4+#5@#6\@endelt{%
  \setlength\dim@a{#2+#3}%
  \ifdim\dim@a>\eq@wdT \eq@wdT\dim@a \fi
  \ifdim\dim@a<\eq@wdMin \eq@wdMin\dim@a \fi
  \eq@dp#5\relax
  \addtolength\eq@vspan{#1+#4+\eq@dp}%
```

Record the max badness of all the lines in `\eq@badness`.

```
  \ifnum#6>\eq@badness \eq@badness#6\relax\fi
}
```

\eq@layout

A value of `?` for `\eq@layout` means that we should deduce which layout to use by looking at the size of the components. Any other value means we have a user-specified override on the layout.

Layout Definitions. Based on initial equation measurements, we can choose a sequence of candidate parshapes that the equation might fit into. We accept the first shape that 'works', else fall to next one. [The sequence is hardcoded in the `\eq@try@layout@`¡shape¿ Would it be useful be more flexible? (eg. try layouts LDA, in order...)]

```
\def\eq@layout{?}
```

\eq@try@layout@?

This is a branching function used to choose a suitable layout if the user didn't specify one in particular.

Default layout: Try Single line layout first, else try Multiline layouts

```
\@namedef{eq@try@layout@?}{%
  \let\eq@trial@b\eq@trial@c
  \edef\@parshape{\parshape 1 0pt \the\eq@linewidth\relax}%
%  \eq@trial@b{i}{\eq@try@layout@multi}%
  \setlength\dim@a{\wd\EQ@copy-2em}% Fudge; can't shrink more than this?
  % if we're in a numbered group, try hard to fit within the numbers
  \dim@b\eq@linewidth
  \if\eq@shiftnumber\else\if\eq@group
    \if\eq@hasNumber\addtolength\dim@b{-\wd\EQ@numbox-\eqnumsep}%
    \else\if\grp@hasNumber\addtolength\dim@b{-\wd\GRP@numbox-\eqnumsep}%
  \fi\fi\fi\fi
  \ifdim\dim@a<\dim@b% Do we even have a chance of fitting to one line?
⟨trace⟩    \breqn@debugmsg{Choose Shape: (\the\wd\EQ@copy) may fit in \the\dim@b}%
```

BRM: assuming it might fit, don't push too hard

```
    \setlength\dim@b{\columnwidth-\dim@a+\eq@wdCond}%
    \rightskip\z@\@plus\dim@b\@minus\hfuzz
    \eq@trial@b{i}{\eq@try@layout@multi}%
  \else
⟨*trace⟩
  \breqn@debugmsg{Choose Shape: Too long (\the\wd\EQ@copy) for one line
            (free width=\the\dim@b)}%
⟨/trace⟩
    \eq@try@layout@multi
  \fi
}
```

Layout Multiline layout: If no LHS, try Stepped(S) layout Else try Stepped(S), Ladder(L), Drop-ladder(D) or Stepladder(l), depending on LHS length.

```
\def\eq@try@layout@multi{%
  \if\EQ@hasLHS
    \ifdim\eq@wdL>\eq@linewidth
⟨trace⟩       \breqn@debugmsg{Choose Shape: LHS \the\eq@wdL > linewidth}%
```

Find the total width of the RHS. If it is relatively short, a step layout is the thing to try.

```
      \setlength\dim@a{\wd\EQ@copy-\eq@wdL}%
      \ifdim\dim@a<.25\eq@linewidth \eq@try@layout@S
      \else \eq@try@layout@l
      \fi
    % BRM: Originally .7: Extreme for L since rhs has to wrap within the remaining 30+%!
    \else\ifdim\eq@wdL>.50\eq@linewidth
⟨*trace⟩
      \breqn@debugmsg{Choose Shape: LHS (\the\eq@wdL) > .50 linewidth (linewidth=\the\eq@linewidth)}%
⟨/trace⟩
      \eq@try@layout@D
    \else
⟨trace⟩      \breqn@debugmsg{Choose Shape: LHS (\the\eq@wdL) not extraordinarily wide}%
      \eq@try@layout@L
    \fi\fi
  \else
⟨trace⟩    \breqn@debugmsg{Choose Shape: No LHS here}%
```

Try one-line layout first, then step layout.

```
    \eq@try@layout@S % (already checked case i)
  \fi
}
```

`\eq@try@layout@D` Change the penalty before the first mathrel symbol to encourage a break there.

Layout D=Drop-Ladder Layout, for wide LHS.

```
LOOOOOOOONG LHS
 = RHS
 = ...
```

If fails, try Almost-Columnar layout

```
\def\eq@try@layout@D{%
  \setlength\dim@a{\eq@linewidth -\eq@indentstep}%
  \edef\@parshape{\parshape 2
    0pt \the\eq@wdL\space \the\eq@indentstep\space \the\dim@a\relax
  }%
  \def\adjust@rel@penalty{\penalty-99 }%
  \eq@trial@b{D}{\eq@try@layout@A}%
}
```

`\eq@try@layout@L` Try a straight ladder layout. Preliminary filtering ensures that `\eq@wdL` is less than 70 of the current line width.

```
Layout L=Ladder layout
 LHS = RHS
```

```
 = RHS
 ...
```

If fails, try Drop-ladder layout. NOTE: This is great for some cases (multi relations?), but tends to break really badly when it fails....

```
\def\eq@try@layout@L{%
  \setlength\dim@b{\eq@linewidth-\eq@wdL}%
  \edef\@parshape{\parshape 2 0pt \the\eq@linewidth\space
    \the\eq@wdL\space \the\dim@b\relax
  }%
  \eq@trial@b{L}{\eq@try@layout@D}%
}
```

`\eq@try@layout@S`

In the "stepped" layout there is no LHS, or LHS is greater than the line width and RHS is small. Then we want to split up the equation into lines of roughly equal width and stagger them downwards to the right, leaving a small amount of whitespace on both sides. But also, if there is an equation number, we want to try first a layout that leaves room for the number. Otherwise it would nearly always be the case that the number would get thrown on a separate line.

Layout S=Stepped layout, typically no LHS or very long, variations on

```
STUFF ....
  + MORE STUFF ...
    + MORE STUFF ...
```

If fails, try Almost-Columnar layout

```
\def\eq@try@layout@S{%
  \setlength\dim@b{\eq@linewidth-2\eqmargin}% \advance\dim@b-1em%
```

About how many lines will we need if dim@b is the line width?

```
  \int@a\wd\EQ@copy \divide\int@a\dim@b
```

Adjust the target width by number of lines times indentstep. We don't need to decrement `\int@a` because TEX division is integer division with truncation.

```
  \addtolength\dim@b{-\int@a\eq@indentstep}%
```

Adjust for equation number. But try not to leave too little room for the equation body.

```
  \if\eq@hasNumber
    \ifdim\dim@b>15em%
%      \advance\dim@b-\eqnumsep \advance\dim@b-\wd\EQ@numbox
      \addtolength\dim@b{-\eq@wdNum}%
    \fi
  \fi
```

Now some hand-waving to set up the parshape.

```
  \int@b\z@
  \def\@tempa{\dim}%
  \edef\@parshape{\parshape 2 0pt \the\dim@b\space
    \the\eqmargin\space\the\dim@b\relax}%
  \eq@trial@b{S}{\eq@try@layout@A}%
}
```

\eq@try@layout@l

This is the “step-ladder” layout: similar to the drop-ladder layout but the LHS is too wide and needs to be broken up.

Layout l = Stepladder Similar to Drop-Ladder, but LHS is long and needs to be broken up. If fails, try Almost-Columnar layout

```
\def\eq@try@layout@l{%
  \setlength\dim@a{\eq@linewidth -\eq@indentstep}%
  \int@a\eq@wdL \divide\int@a\dim@a
  \advance\int@a\tw@
  \edef\@parshape{\parshape \number\int@a\space
    0pt \the\eq@linewidth
  }%
  \advance\int@a-\tw@
  \setlength\dim@b{2\eq@indentstep}%
  \setlength\dim@c{\eq@linewidth -\dim@b}%
  \edef\@parshape{\@parshape
    \replicate{\int@a}{\space\the\eq@indentstep\space\the\dim@a}%
    \space\the\dim@b\space\the\dim@c\relax
  }%
  \eq@trial@b{l}{\eq@try@layout@A}%
}
```

\eq@try@layout@A

In the “almost-columnar” layout, which is the layout of last resort, we let all lines run to the full width and leave the adjusting of the indents to later.

Layout A = Almost-Columnar layout. Pretty much straight full width, more of a last-resort. If fails, give up.

```
\def\eq@try@layout@A{%
  \edef\@parshape{\parshape 1 0pt \the\eq@linewidth\relax}%
  \if\EQ@hasLHS \def\adjust@rel@penalty{\penalty-99 }\fi
  \eq@trial@b{A}{}%
}
```

\eq@shiftnumber

MH: Should be moved to a section where all keys are set to defaults.

```
\let\eq@shiftnumber\@False
```

eq@retry@with@number@a

Number placement adjustments

```
\def\eq@retry@with@number{%
 \if\eq@shiftnumber
⟨trace⟩    \breqn@debugmsg{Place number: Shifted number requested}%
 \else
```

Condition and right numbers? We’re just going to have to shift.

```
    \ifdim\eq@wdCond>\z@\if R\eqnumside
⟨trace⟩       \breqn@debugmsg{Place number: Condition w/Right number => Shift number}%
      \let\eq@shiftnumber\@True
    \fi\fi
```

Compute free space.

```
%     \dim@b\eqnumsep\advance\dim@b\wd\EQ@numbox
    \dim@b\eq@wdNum
    \if L\eqnumside
```

```
      \ifdim\@totalleftmargin>\dim@b\dim@b\@totalleftmargin\fi
    \else
      \addtolength\dim@b{\@totalleftmargin}%
    \fi
    \setlength\dim@a{\eq@linewidth-\dim@b}%\advance\dim@a1em\relax% Allowance for shrink?
```

Set up test against 1-line case only if not in a group

```
    \int@a\@ne\if\eq@group\int@a\maxint\fi
```

Now check for cases.

```
    \if\eq@shiftnumber               % Already know we need to shift
    \else\ifdim\eq@wdT<\dim@a % Fits!
```

left & right skips will be done later, and parshape adjusted if needed.

```
⟨trace⟩      \breqn@debugmsg{Place number: eqn and number fit together}%
%    \else\ifnum\eq@lines=\int@a %  Shift, if single line, unless inside a dgroup.
```

NOTE: this is too strong for dgroup!

```
⟨*trace⟩
%      \breqn@debugmsg{Place number: single line too long with number => Shift number \the\int@a}%
⟨/trace⟩
%      \let\eq@shiftnumber\@True
    \else
```

Retry: use leftskip for space for number(for now; whether right/left) & adjust parshape

```
%      \leftskip\wd\EQ@numbox\advance\leftskip\eqnumsep
      \setlength\leftskip{\eq@wdNum}%
      \setlength\rightskip{\z@\@plus\dim@a}%
      \adjust@parshape\@parshape
⟨*trace⟩
      \breqn@debugmsg{Place number: Try with \leftskip=\the\leftskip, \rightskip=\the\rightskip,
                \MessageBreak==== parshape\space\@xp\@gobble\@parshape}%
⟨/trace⟩
      \nointerlineskip
      \edef\eq@prev@lines{\the\eq@lines}%
      \edef\eq@prev@badness{\the\eq@badness}% BRM
      \eq@trial@p
      \int@a\eq@prev@badness\relax\advance\int@a 50\relax%?
      \int@b\eq@prev@lines  \if\eq@group\advance\int@b\@ne\fi% Allow 1 extra line in group
      \ifnum\eq@lines>\int@b % \eq@prev@lines
⟨trace⟩        \breqn@debugmsg{Adjustment causes more breaks => Shift number}%
        \let\eq@shiftnumber\@True
      \else\if\eq@badline
⟨trace⟩        \breqn@debugmsg{Adjustment causes bad lines (\the\eq@badness) => Shift}%
        \let\eq@shiftnumber\@True
      \else\ifnum\eq@badness>\int@a % BRM: New case
⟨*trace⟩
        \breqn@debugmsg{Adjustment is badder than previous
                  (\the\eq@badness >> \eq@prev@badness) => Shift}%
⟨/trace⟩
        \let\eq@shiftnumber\@True
      \else
```

```
⟨trace⟩         \breqn@debugmsg{Adjustment succeeded}%
        \fi\fi%\fi
    \fi\fi\fi
```

If we got shifted, restore parshape, etc,

```
    \if\eq@shiftnumber
      \EQ@trial% Restore parshape & other params,
      \leftskip\z@\let\eq@shiftnumber\@True % But set shift & leftskip
      \edef\@parshape{\eq@parshape}% And copy saved parshape back to ‘working copy’ !?!?
    \fi
    \eq@trial@save\EQ@trial      % Either way, save the trial state.
  \fi
}
```

`\adjust@parshape` Varies depending on the layout.

Adjust a parshape variable for a given set of left|right skips. Note that the fixed part of the left|right skips effectively comes out of the parshape widths (NOT in addition to it). We also must trim the widths so that the sum of skips, indents and widths add up to no more than the `\eq@linewidth`.

```
\def\adjust@parshape#1{%
  \@xp\adjust@parshape@a#1\relax
  \edef#1{\temp@a}%
}
```

`\adjust@parshape@a`
`\adjust@parshape@b`

```
\def\adjust@parshape@a#1 #2\relax{%
  \setlength\dim@a{\leftskip+\rightskip}%
  \edef\temp@a{#1}%
  \adjust@parshape@b#2 @ @ \relax
}
\def\adjust@parshape@b#1 #2 {%
  \ifx @#1\edef\temp@a{\temp@a\relax}%
    \@xp\@gobble
  \else
    \dim@b#1\relax
    \dim@c#2\relax
    \addtolength\dim@c{\dim@a+\dim@b}%
    \ifdim\dim@c>\eq@linewidth\setlength\dim@c{\eq@linewidth}\fi
     \addtolength\dim@c{-\dim@b}%
    \edef\temp@a{\temp@a\space\the\dim@b\space\the\dim@c}%
  \fi
  \adjust@parshape@b
}
```

`\eq@ml@record@indents` Plunk the parshape’s indent values into an array for easy access when constructing `\eq@measurements`.

```
\def\eq@ml@record@indents{%
  \int@a\z@
  \def\@tempa{%
    \advance\int@a\@ne
    \@xp\edef\csname eq@i\number\int@a\endcsname{\the\dim@a}%
```

```
    \ifnum\int@a<\int@b \afterassignment\@tempb \fi
    \dim@a
  }%
  \def\@tempb{\afterassignment\@tempa \dim@a}%
  \def\@tempc##1##2 {\int@b##2\afterassignment\@tempa\dim@a}%
  \@xp\@tempc\@parshape
}
```

`\@endelt` This is a scan marker. It should get a non-expandable definition. It could be `\relax`, but let's try a chardef instead.

```
\chardef\@endelt=`\?
```

`\eq@measurements` This is similar to a parshape spec but for each line we record more info: space above, indent, width x height + dp, and badness.

```
\def\eq@measurements{%
  \@elt 4.5pt/5.0pt,66.0ptx6.8pt+2.4pt@27\@endelt
  ...
}
```

`\eq@measure@lines` Loop through the list of boxes to measure things like total height (including interline stretch), etc.. We check the actual width of the current line against the natural width—after removing rightskip—in case the former is *less* than the latter because of shrinkage. In that case we do not want to use the natural width for RHS-max-width because it might unnecessarily exceed the right margin.

```
\def\eq@measure@lines{%
  \let\eq@ml@continue\eq@measure@lines
  \setbox\tw@\lastbox \dim@b\wd\tw@ % find target width of line
  \setbox\z@\hbox to\dim@b{\unhbox\tw@}% check for overfull
  \eq@badness\badness
  \ifnum\eq@badness<\inf@bad \else \let\eq@badline\@True \fi
  \eq@ml@a \eq@ml@continue
}
```

`\eq@ml@a`

```
\def\eq@ml@a{%
  \setbox\tw@\hbox{\unhbox\z@ \unskip}% find natural width
⟨*trace⟩
  \ifnum\eq@badness<\inf@bad\else\breqn@debugmsg{!?! Overfull: \the\wd\tw@ >\the\dim@b}\fi
⟨/trace⟩
```

Is actual width less than natural width?

```
  \ifdim\dim@b<\wd\tw@ \setlength\dim@a{\dim@b}% shrunken line
  \else                \setlength\dim@a{\wd\tw@}% OK to use natural width
  \fi
  \addtolength\dim@a{-\leftskip}% BRM: Deduct the skip if we're retrying w/number
```

If there's no aboveskip, assume we've reached the top of the equation.

```
  \skip@a\lastskip \unskip \unpenalty
  \ifdim\skip@a=\z@
    \let\eq@ml@continue\relax % end the recursion
```

```
  \else
    % Sum repeated vskips if present
    \def\@tempa{%
      \ifdim \lastskip=\z@
      \else \addtolength\skip@a{\lastskip}\unskip\unpenalty \@xp\@tempa
      \fi
    }%
  \fi
  \edef\eq@measurements{\@elt
    \the\skip@a\space X% extra space to facilitate extracting only the
                        % dimen part later
    \csname eq@i%
      \ifnum\eq@curline<\parshape \number\eq@curline
      \else\number\parshape
      \fi
    \endcsname,\the\dim@a x\the\ht\tw@+\the\dp\tw@ @\the\eq@badness\@endelt
    \eq@measurements
  }%
  \advance\eq@curline\m@ne
  \ifnum\eq@curline=\z@ \let\eq@ml@continue\relax\fi
}
```

`\eq@ml@vspace` Handle an embedded vspace.

```
\def\eq@ml@vspace{%
  \global\advance\eq@vspan\lastskip \unskip\unpenalty
  \ifdim\lastskip=\z@ \else \@xp\eq@ml@vspace \fi
}
```

`\eq@dense@enough`

```
\def\eq@dense@enough{%
  \ifnum\eq@lines<\thr@@
⟨trace⟩    \breqn@debugmsg{Density check: less than 3 lines; OK}%
    \@True
  \else
    \ifdim\eq@wdL >.7\eq@wdT
⟨trace⟩      \breqn@debugmsg{Density check: LHS too long; NOT OK}%
      \@False
    \else \@xp\@xp\@xp\eq@dense@enough@a
    \fi
  \fi
}
```

`\true@true@true`
`\true@false@true`
`\false@true@false`
`\false@false@false`

```
\def\true@true@true   {\fi\fi\iftrue \iftrue \iftrue }
\def\true@false@true  {\fi\fi\iftrue \iffalse\iftrue }
\def\false@true@false {\fi\fi\iffalse\iftrue \iffalse}
\def\false@false@false{\fi\fi\iffalse\iffalse\iffalse}
```

`\eq@density@factor` This number specifies, for the ladder layout, how much of the equation's bounding box should contain visible material rather than whitespace. If the amount of visible material drops below this value, then we switch to the drop-ladder layout. The optimality of this factor is highly dependent on the equation contents; .475 was chosen as the default just because it worked well with the sample equation, designed to be as average as possible, that I used for testing.

```
\def\eq@density@factor{.475}
```

`\eq@dense@enough@a` Calculate whether there is more visible material than whitespace within the equation's bounding box. Sum up the actual line widths and compare to the total "area" of the bounding box. But if we have an extremely large number of lines, fall back to an approximate calculation that is more conservative about the danger of exceeding `\maxdimen`.

```
\def\eq@dense@enough@a{%
  \@True \fi
  \ifnum\eq@lines>\sixt@@n
    \eq@dense@enough@b
  \else
    \dim@b\z@ \let\@elt\eq@delt \eq@measurements
    \dim@c\eq@density@factor\eq@wdT \multiply\dim@c\eq@lines
⟨trace⟩    \breqn@debugmsg{Density check: black \the\dim@b/\eq@density@factor total \the\dim@c}%
    \ifdim\dim@b>\dim@c \true@false@true \else \false@false@false \fi
  \fi
}
```

`\eq@delt` Args are space-above, indent, width, height, depth, badness.

```
\def\eq@delt#1X#2,#3x#4+#5@#6\@endelt{\addtolength\dim@b{#3}}%
```

`\eq@dense@enough@b` This is an approximate calculation used to keep from going over `\maxdimen` if the number of lines in our trial break is large enough to make that a threat. If l, t, n represent left-side-width, total-width, and number of lines, the formula is

```
l/t < .4n/(.9n-1)
```

or equivalently, since rational arithmetic is awkward in TeX: b

```
l/t < 4n/(9n-10)
```

.

```
\def\eq@dense@enough@b{%
  \int@b\eq@wdT \divide\int@b\p@
  \dim@b\eq@wdL \divide\dim@b\int@b
  \dim@c\eq@lines\p@ \multiply\dim@c\f@ur
  \int@b\eq@lines \multiply\int@b 9 \advance\int@b -10%
  \divide\dim@c\int@b
⟨trace⟩  \breqn@debugmsg{Density check: l/t \the\dim@b\space< \the\dim@c\space 4n/(9n-10)?}%
  \ifdim\dim@b<\dim@c \true@true@true \else \false@true@false \fi
}
```

`\eq@parshape`

```
\let\eq@parshape\@empty
```

`\eq@params` The interline spacing and penalties in `\eq@params` are used during both preliminary line breaking and final typesetting.

```
\def\eq@params{%
  \baselineskip\eqlinespacing
  \lineskip\eqlineskip \lineskiplimit\eqlineskiplimit
```

Forbid absolutely a pagebreak that separates the first line or last line of a multiline equation from the rest of it. Or in other words: no equation of three lines or less will be broken at the bottom of a page; instead it will be moved whole to the top of the next page. If you really really need a page break that splits the first or last line from the rest of the equation, you can always fall back to`\pagebreak`, I suppose.

```
  \clubpenalty\@M \widowpenalty\@M \interlinepenalty\eqinterlinepenalty
  \linepenalty199 \exhyphenpenalty5000 % was 9999: make breaks at, eg. \* a bit easier.
```

For equations, hfuzz should be at least 1pt. But we have to fake it a little because we are running the equation through TEX's paragrapher. In our trials we use minus 1pt in the rightskip rather than hfuzz; and we must do the same during final breaking of the equation, otherwise in borderline cases TEX will use two lines instead of one when our trial indicated that one line would be enough.

```
  \ifdim\hfuzz<\p@ \hfuzz\p@ \fi
%\hfuzz=2pt
%  \ifdim\hfuzz<2pt\relax \hfuzz2pt \fi
  \parfillskip\z@skip
%  \hfuzz\z@
```

Make sure we skip TEX's preliminary line-breaking pass to save processing time.

```
  \tolerance9999 \pretolerance\m@ne
}
```

30 Equation layout options

Using the notation C centered, I indented (applied to the equation body), T top, B bottom, M middle, L left, R right (applied to the equation number), the commonly used equation types are C, CRM, CRB, CLM, CLT, I, IRM, IRB, ILM, ILT. In other words, CLM stands for Centered equation body with Left-hand Middle-placed equation number, and IRB stands for Indented equation with Right-hand Bottom-placed equation number.

Here are some general thoughts on how to place an equation tag. Currently it does not work as desired: the L option positions the tag app. 10 lines below the math expression, the RM doesn't position the tag on the baseline for single-line math expressions. Therefore I am going to first write what I think is supposed to happen and then implement it.

Below is a small list where especially the two three specifications should be quite obvious, I just don't want to forget anything and it is important to the implementation.

Definition 1 If a display consists of exactly one line, the tag should always be placed on the same baseline as the math expression.

The remaining comments refer to multi-line displays.

Definition 2 If a tag is to be positioned at the top (T), it should be placed such that the baseline of the tag aligns with the baseline of the top line of the display.

Definition 3 If a tag is to be positioned at the bottom (B), it should be placed such that the baseline of the tag aligns with the baseline of the bottom line of the display.

Definition 4 If a tag is to be positioned vertically centered (M), it should be placed such that the baseline of the tag is positioned exactly halfway between the baseline of the top line of the display and the baseline of the bottom line of the display.

Definitions 1–3 are almost axiomatic in their simplicity. Definition 4 is different because I saw at least two possibilities for which area to span:

- Calculate distance from top of top line to the bottom of the bottom line, position the vertical center of the tag exactly halfway between those two extremes.
- Calculate the distance from the baseline of the top line to the baseline of the bottom line, position the baseline of the tag exactly halfway between these two extremes.

Additional combinations of these methods are possible but make little sense in my opinion. I have two reasons for choosing the latter of these possibilities: Firstly, two expressions looking completely identical with the exception of a superscript in the first line or a subscript in the last line will have the tag positioned identically. Secondly, then M means halfway between T and B positions which makes good sense and then also automatically fulfills Definition 1.

From an implementation perspective, these definitions should also make it possible to fix a deficiency in the current implementation, namely that the tag does not influence the height of a display, even if the display is a single line. This means that two single-line expressions in a `dgroup` can be closer together than `\intereqskip` if the math expressions are (vertically) smaller than the tag.

31 Centered Right-Number Equations

`\eq@dump@box` #1 might be `\unhbox` or `\unhcopy`; #2 is the box name.

```
\def\eq@dump@box#1#2{%
%\debug@box#1%
  \noindent #1#2\setbox\f@ur\lastbox \setbox\tw@\lastbox
```

If the LHS contains shrinkable glue, in an L layout the alignment could be thrown off if the first line is shrunk noticeably. For the time being, disable shrinking on the left-hand side. The proper solution requires more work

mjd,1999/03/17

.

```
  \if L\eq@layout \box\tw@ \else\unhbox\tw@\fi
  \adjust@rel@penalty \unhbox\f@ur
}
```

Various typesetting bits, invoked from `\eq@finish` BRM: This has been extensively refactored from the original breqn, initially to get left|right skips and parshape used consistently, ultimately to get most things handled the same way, in the same order.

Given that left and right skips have been set, typeset the frame, number and equation with the given number side and placement

```
\def\eq@typeset@Unnumbered{%
  \eq@typeset@frame
  \eq@typeset@equation
}
\def\eq@typeset@LM{%
  \setlength\dim@a{(\eq@vspan+\ht\EQ@numbox-\dp\EQ@numbox)/2}%
  \eq@typeset@leftnumber
  \eq@typeset@frame
  \eq@typeset@equation
}
```

Typeset equation and left-top number (and shifted)

```
\def\eq@typeset@LT{%
  \dim@a\eq@firstht
  \eq@typeset@leftnumber
  \eq@typeset@frame
  \eq@typeset@equation
}
```

Typeset equation and left shifted number

```
\def\eq@typeset@LShifted{%
  % place number
  \copy\EQ@numbox \penalty\@M
  \dim@a\eqlineskip
  \if F\eq@frame\else
     \setlength\dim@a{\eq@framesep+\eq@framewd}%
  \fi
  \kern\dim@a
  \eq@typeset@frame
  \eq@typeset@equation
}
```

Typeset equation and right middle number

```
\def\eq@typeset@RM{%
  \setlength\dim@a{(\eq@vspan+\ht\EQ@numbox-\dp\EQ@numbox)/2}%
  \eq@typeset@rightnumber
  \eq@typeset@frame
  \eq@typeset@equation
}
```

Typeset equation and right bottom number

```
\def\eq@typeset@RB{%
  % NOTE: is \eq@dp useful here
  \setlength\dim@a{\eq@vspan-\ht\EQ@numbox-\dp\EQ@numbox}%
  \eq@typeset@rightnumber
  \eq@typeset@frame
  \eq@typeset@equation
}
```

Typeset equation and right shifted number

```
\def\eq@typeset@RShifted{%
  % place number
  \eq@typeset@frame
  \eq@typeset@equation
  \penalty\@M
  \dim@a\eqlineskip
  \if F\eq@frame\else
    \addtolength\dim@a{\eq@framesep+\eq@framewd}%
  \fi
  \parskip\dim@a
  \hbox to\hsize{\hfil\copy\EQ@numbox}\@@par%
}
```

Debugging aid to show all relevant formatting info for a given eqn.

```
⟨*trace⟩
\def\debug@showformat{%
  \breqn@debugmsg{Formatting Layout:\eq@layout\space Center/indent: \eqindent\space
    Number placement \eqnumside\eqnumplace:
    \MessageBreak==== \eq@linewidth=\the\eq@linewidth, \@totalleftmargin=\the\@totalleftmargin,
    \MessageBreak==== Centered Lines=\theb@@le\eq@centerlines, Shift Number=\theb@@le\eq@shiftnumber,
    \MessageBreak==== \eq@wdT=\the\eq@wdT, \eq@wdMin=\the\eq@wdMin,
    \MessageBreak==== LHS=\theb@@le\EQ@hasLHS: \eq@wdL=\the\eq@wdL,
    \MessageBreak==== \eq@firstht=\the\eq@firstht, \eq@vspan=\the\eq@vspan
    \MessageBreak==== \eq@wdNum=\the\eq@wdNum
    \MessageBreak==== \eq@wdCond=\the\eq@wdCond, \conditionsep=\the\conditionsep,
    \MessageBreak==== \leftskip=\the\leftskip, \rightskip=\the\rightskip,
    \MessageBreak==== \abovedisplayskip=\the\abovedisplayskip,
    \MessageBreak==== \belowdisplayskip=\the\belowdisplayskip
    \MessageBreak==== parshape=\eq@parshape}%
}
⟨/trace⟩
```

Set left & right skips for centered equations, making allowances for numbers (if any, right, left) and constraint.

Amazingly, I've managed to collect all the positioning logic for centered equations in one place, so it's more manageable. Unfortunately, by the time it does all it needs to do, it has evolved I'm (re)using so many temp variables, it's becoming unmanageble!

```
\def\eq@C@setsides{%
  % \dim@c = space for number, if any, and not shifted.
  \dim@c\z@
  \if\eq@hasNumber\if\eq@shiftnumber\else
    \dim@c\eq@wdNum
  \fi\fi
  % \dim@e = space for condition(on right), if any and formula is only a single line.(to center nicely)
  % but only count it as being right-aligned if we're not framing, since the frame must enclose it.
  \dim@e\z@
  \if F\eq@frame
    \ifnum\eq@lines=\@ne\ifdim\eq@wdCond>\z@
      \setlength\dim@e{\eq@wdCond+\conditionsep}%
```

```
  \fi\fi\fi
  % \dim@b = minimum needed on left max(totalleftmargin, left number space)
  \dim@b\z@
  \if L\eqnumside\ifdim\dim@b<\dim@c
    \dim@b\dim@c
  \fi\fi
  \ifdim\dim@b<\@totalleftmargin
    \dim@b\z@
  \else
    \addtolength\dim@b{-\@totalleftmargin}%
  \fi
  % \dim@d = minimum needed on right max(condition, right number space)
  \dim@d\dim@e
  \if R\eqnumside\ifdim\dim@d<\dim@c
    \dim@d\dim@c
  \fi\fi
  % \dim@a = left margin; initially half available space
  % \dim@c = right margin;  ditto
  \setlength\dim@a{(\eq@linewidth-\eq@wdT+\dim@e+\@totalleftmargin)/2}%
  \dim@c=\dim@a
  % If too far to the left
  \ifdim\dim@a<\dim@b
     \addtolength\dim@c{\dim@a-\dim@b}%
     \ifdim\dim@c<\z@\dim@c=\z@\fi
     \dim@a=\dim@b
  % Or if too far to the right
  \else\ifdim\dim@c<\dim@d
     \addtolength\dim@a{\dim@c-\dim@d}%
     \ifdim\dim@a<\z@\dim@a=\z@\fi
     \dim@c=\dim@d
  \fi\fi
  % Now, \dim@d,\dim@e is the left & right glue to center each line for centerlines
  \setlength\dim@e{\eq@wdT-\eq@wdMin}\dim@d=\z@
```

NOTE: Need some work here centering when there's a condition

```
%  \advance\dim@e-\eq@wdT\multiply\dim@e-1\relax
%  \if\eq@wdMin<\dim@e\dim@e\eq@wdMin\fi
%  \multiply\dim@e-1\relax\advance\dim@e\eq@wdT
  \dim@d\z@
  \if\eq@centerlines
    \divide\dim@e2\relax
    \dim@d=\dim@e
  \fi
  \setlength\leftskip{\dim@a\@plus\dim@d}%
  \addtolength\dim@e{\dim@c}%
  \setlength\rightskip{\z@\@plus\dim@e}%\@minus5\p@
  % Special case: if framing, reduce the stretchiness of the formula (eg. condition)
  % Or if we have a right number, FORCE space for it
  \dim@b\z@
  \if F\eq@frame\else
    \dim@b\dim@c
```

```
  \fi
  \if\@And{\eq@hasNumber}{\@Not{\eq@shiftnumber}}%
    \if R\eqnumside
      \dim@c\eq@wdNum
      \ifdim\dim@c>\dim@b
        \dim@b\dim@c
      \fi
    \fi
  \fi
  % If either of those cases requires hard rightskip, move that part from glue.
  \ifdim\dim@b>\z@
    \addtolength\dim@e{-\dim@c}%
    \rightskip\dim@b\@plus\dim@e%\@minus5\p@
  \fi
  % And peculiar further special case: in indented environs, width isn't where it would seem
  \ifdim\eq@wdCond>\z@
    \addtolength\rightskip{-\@totalleftmargin}%
  \fi
  \parfillskip\z@skip
}
```

Set the left and right side spacing for indented equations Some things handled by eq@C@setsides that probably apply here????

- centerlines
- `\@totalleftmargin`: SHOULD we move farther right?

Leftskip is normally just the requested indentation

```
\def\eq@I@setsides{%
  \leftskip\mathindent
```

But move left, if shifted number presumably because of clashed w/ number?

```
  \if\eq@shiftnumber
    \setlength\dim@a{\eq@linewidth-\eq@wdT-\mathindent}%
    \ifdim\dim@a<\z@
      \leftskip=\z@ % Or something minimal?
    \fi
  \fi
```

Push gently from right.

```
  \dim@a=\z@
  \setlength\dim@b{\eq@linewidth-\leftskip-\eq@wdMin}%
```

Special case: if framing be much more rigid(?)

```
  \if F\eq@frame
  \else
    \setlength\dim@a{\eq@linewidth-\leftskip-\eq@wdT}
    \addtolength\dim@b{-\dim@a}%
  \fi
  % Or force the space for right number, if needed
%    \begin{macrocode}
```

```
  \if\@And{\eq@hasNumber}{\@Not{\eq@shiftnumber}}%
    \if R\eqnumside
      \dim@c=\eq@wdNum
      \if\dim@c>\dim@a
        \addtolength\dim@b{-\dim@c}%
        \dim@a=\dim@c
      \fi
    \fi
  \fi
  \setlength\rightskip{\dim@a\@plus\dim@b \@minus\hfuzz }%\hfuzz\z@
  \parfillskip\z@skip
}
```

Typesetting pieces: frame, equation and number (if any) `\dim@a` should contain the downward displacement of number's baseline

```
\def\eq@typeset@leftnumber{%
  \setlength\skip@c{\dim@a-\ht\EQ@numbox}%
  \vglue\skip@c% NON discardable
  \copy\EQ@numbox \penalty\@M
  \kern-\dim@a
}
\def\eq@typeset@rightnumber{%
  \setlength\skip@c{\dim@a-\ht\EQ@numbox}%
  \vglue\skip@c% NON discardable
  \hbox to \hsize{\hfil\copy\EQ@numbox}\penalty\@M
  \kern-\dim@a
}
\def\eq@typeset@equation{%
  \nobreak
  \eq@params\eq@parshape
  \nointerlineskip\noindent
  \add@grp@label
  \eq@dump@box\unhbox\EQ@box\@@par
}
```

32 Framing an equation

`\eqframe` The `\eqframe` function is called in vertical mode with the reference point at the top left corner of the equation, including any allowance for `\fboxsep`. Its arguments are the width and height of the equation body, plus fboxsep.

```
\newcommand\eqframe[2]{%
  \begingroup
  \fboxrule=\eq@framewd\relax\fboxsep=\eq@framesep\relax
  \framebox{\z@rule\@height#2\kern#1}%
  \endgroup
}
```

The frame is not typeset at the correct horizontal position. Will fix later.

```
\def\eq@addframe{%
  \hbox to\z@{%
    \setlength\dim@a{\eq@framesep+\eq@framewd}%
    \kern-\dim@a
    \vbox to\z@{\kern-\dim@a
      \hbox{\eqframe{\eq@wdT}{\eq@vspan}}%
      \vss
    }%
    \hss
  }%
}
\def\eq@typeset@frame{%
  \if F\eq@frame\else
   % Tricky: put before \noindent, so it's not affected by glue in \leftskip
   \nobreak\nointerlineskip
   \vbox to\eq@firstht{\moveright\leftskip\hbox to\z@{\eq@addframe\hss}\vss}%
   \kern-\eq@firstht
  \fi
}
```

33 Delimiter handling

The special handling of delimiters is rather complex, but everything is driven by two motives: to mark line breaks inside delimiters as less desirable than line breaks elsewhere, and to make it possible to break open left-right boxes so that line breaks between `\left` and `\right` delimiters are not absolutely prohibited. To control the extent to which line breaks will be allowed inside delimiters, set `\eqbreakdepth` to the maximum nesting depth. Depth 0 means never break inside delimiters.

Note: `\eqbreakdepth` is not implemented as a LaTeX counter because changes done by `\setcounter` etc. are always global.

It would be natural to use grouping in the implementation—at an open delimiter, start a group and increase mathbin penalties; at a close delimiter, close the group. But this gives us trouble in situations like the `array` environment, where a close delimiter might fall in a different cell of the `\halign` than the open delimiter. Ok then, here's what we want the various possibilities to expand to. Note that `\right` and `\biggr` are being unnaturally applied to a naturally open-type delimiter.

```
( -> \delimiter"4... \after@open
\left( ->
  \@@left \delimiter"4... \after@open
\right( ->
  \@@right \delimiter"4... \after@close
\biggl( ->
  \mathopen{\@@left \delimiter... \vrule...\@@right.}
  \after@open
\biggr( ->
  \mathclose{\@@left \delimiter... \vrule...\@@right.}
  \after@close
```

```
\bigg\vert ->
  \mathord{\@@left \delimiter... \vrule...\@@right.}
\biggm\vert ->
  \mathrel{\@@left \delimiter... \vrule...\@@right.}
```

First save the primitive meanings of `\left` and `\right`.

```
\@saveprimitive\left\@@left
\@saveprimitive\right\@@right
```

The variable `\lr@level` is used by the first mathrel in an equation to tell whether it is at top level: yes? break and measure the LHS, no? keep going.

```
\newcount\lr@level
```

It would be nice to have better error checking here if the argument is not a delimiter symbol at all.

Ah, a small problem when renaming commands. In the original version, `\delimiter` is hijacked in order to remove the `\after@bidir` (or open or close) instruction following the delimiter declaration.

```
\ExplSyntaxOn
\def\eq@left{%
  \@ifnext .{\eq@nullleft}{\begingroup \let\math_delimiter:NNnNn \eq@left@a}%
}
\def\eq@right{%
  \@ifnext .{\eq@nullright}{\begingroup \let \math_delimiter:NNnNn \eq@right@a}%
}
```

The arguments are: #1 delim symbol, #2 .

```
%\def\eq@left@a#1 #2{\endgroup\@@left\delimiter#1\space \after@open}
\def\eq@left@a#1#2#3#4#5#6{\endgroup
  \@@left \math_delimiter:NNnNn #1#2{#3}#4{#5}\after@open}
\def\eq@right@a#1#2#3#4#5#6{\endgroup
  \@@right \math_delimiter:NNnNn #1#2{#3}#4{#5}\after@close \ss@scan{#1#2{#3}#4{#5}}%
}
\ExplSyntaxOff
```

The null versions.

```
\def\eq@nullleft#1{\@@left#1\after@open}
\def\eq@nullright#1{\@@right#1\after@close}
```

Here is the normal operation of `\biggl`, for example.

```
\biggl ->\mathopen \bigg
{\mathopen}

\bigg #1->{\hbox {$\left #1\vbox to14.5\p@ {}\right .\n@space $}}
#1<-(
```

^^AFor paren matching:) Like `\left`, `\biggl` coerces its delimiter to be of mathopen type even if its natural inclination is towards closing.

The function `\delim@reset` makes delimiter characters work just about the same as they would in normal LaTeX.

```
\def\delim@reset{%
```

```
  \let\after@open\relax \let\after@close\relax
  \let\left\@@left \let\right\@@right
}
```

If the amsmath or exscale package is loaded, it will have defined `\bBigg@`; if not, the macros `\big` and variants will have hard-coded point sizes as inherited through the ages from `plain.tex`. In this case we can kluge a little by setting `\big@size` to `\p@`, so that our definition of `\bBigg@` will work equally well with the different multipliers.

```
\@ifundefined{bBigg@}{% not defined
  \let\big@size\p@
  \def\big{\bBigg@{8.5}}\def\Big{\bBigg@{11.5}}%
  \def\bigg{\bBigg@{14.5}}\def\Bigg{\bBigg@{17.5}}%
  \def\biggg{\bBigg@{20.5}}\def\Biggg{\bBigg@{23.5}}%
}{}
\def\bBigg@#1#2{%
  {\delim@reset
   \left#2%
   \vrule\@height#1\big@size\@width-\nulldelimiterspace
   \right.
  }%
}
```

.

```
\def\bigl#1{\mathopen\big{#1}\after@open}
\def\Bigl#1{\mathopen\Big{#1}\after@open}
\def\biggl#1{\mathopen\bigg{#1}\after@open}
\def\Biggl#1{\mathopen\Bigg{#1}\after@open}
\def\bigggl#1{\mathopen\biggg{#1}\after@open}
\def\Bigggl#1{\mathopen\Biggg{#1}\after@open}

\def\bigr#1{\mathclose\big{#1}\after@close}
\def\Bigr#1{\mathclose\Big{#1}\after@close}
\def\biggr#1{\mathclose\bigg{#1}\after@close}
\def\Biggr#1{\mathclose\Bigg{#1}\after@close}
\def\bigggr#1{\mathclose\biggg{#1}\after@close}
\def\Bigggr#1{\mathclose\Biggg{#1}\after@close}

%% No change needed, I think. [mjd,1998/12/04]
%%\def\bigm{\mathrel\big}
%%\def\Bigm{\mathrel\Big}
%%\def\biggm{\mathrel\bigg}
%%\def\Biggm{\mathrel\Bigg}
%%\def\bigggm{\mathrel\biggg}
%%\def\Bigggm{\mathrel\Biggg}
```

`\m@@DeL` `\d@@DeL` `\m@@DeR` `\d@@DeR` `\m@@DeB` `\d@@DeB`

Original definition of `\m@DeL` from flexisym is as follows. `\m@DeR` and `\m@DeB` are the same except for the math class number.

```
\def\m@DeL#1#2#3{%
  \delimiter"4\@xp\delim@a\csname sd@#1#2#3\endcsname #1#2#3 }
```

Save the existing meanings of `\m@De[LRB]`.

Define display variants of DeL, DeR, DeB

```
\ExplSyntaxOn
\cs_set:Npn \math_dsym_DeL:Nn #1#2{\math_bsym_DeL:Nn #1{#2}\after@open}
\cs_set:Npn \math_dsym_DeR:Nn #1#2{\math_bsym_DeR:Nn #1{#2}\after@close}
\cs_set:Npn \math_dsym_DeB:Nn #1#2{\math_bsym_DeB:Nn #1{#2}\after@bidir}

%%%%%
%%%%%\let\m@@DeL\m@DeL \let\m@@DeR\m@DeR \let\m@@DeB\m@DeB
%%%%%\def\d@@DeL#1#2#3{%
%%%%%  \delimiter"4\@xp\delim@a\csname sd@#1#2#3\endcsname #1#2#3 \after@open
%%%%%}
%%%%%\def\d@@DeR#1#2#3{%
%%%%%  \delimiter"5\@xp\delim@a\csname sd@#1#2#3\endcsname #1#2#3 \after@close
%%%%%}
%%%%%\def\d@@DeB#1#2#3{%
%%%%%  \delimiter"0\@xp\delim@a\csname sd@#1#2#3\endcsname #1#2#3 \after@bidir
%%%%%}
```

BRM: These weren't defined, but apparently should be. Are these the right values???

```
%%%%%%\let\m@@DeA\m@DeA\let\d@@DeA\m@DeA%
```

`\after@open` `\after@close` `\after@bidir` `\zero@bop` `\bop@incr`

`\after@open` and `\after@close` are carefully written to avoid the use of grouping and to run as fast as possible. `\zero@bop` is the value used for `\prebinoppenalty` at delimiter level 0, while `\bop@incr` is added for each level of nesting. The standard values provide that breaks will be prohibited within delimiters below nesting level 2.

```
\let\after@bidir\@empty
\mathchardef\zero@bop=888 \relax
\mathchardef\bop@incr=4444 \relax
\def\after@open{%
  \global\advance\lr@level\@ne
  \prebinoppenalty\bop@incr \multiply\prebinoppenalty\lr@level
  \advance\prebinoppenalty\zero@bop
  \ifnum\eqbreakdepth<\lr@level
     \cs_set_eq:NN \math_sym_Bin:Nn \math_isym_Bin:Nn %%%%%%\let\m@Bin\m@@Bin
```

Inside delimiters, add some fillglue before binops so that a broken off portion will get thrown flush right. Also shift it slightly further to the right to ensure that it clears the opening delimiter.

```
  \else
    \eq@binoffset=\eqbinoffset
    \advance\eq@binoffset\lr@level\eqdelimoffset plus1fill\relax
    \def\dt@fill@cancel{\hskip\z@ minus1fill\relax}%
  \fi
  \penalty\@M % BRM: discourage break after an open fence?
}
\def\after@close{%
  \global\advance\lr@level\m@ne
  \prebinoppenalty\bop@incr \multiply\prebinoppenalty\lr@level
  \advance\prebinoppenalty\zero@bop
  \ifnum\eqbreakdepth<\lr@level
```

```
  \else \cs_set_eq:NN \math_sym_Bin:Nn \math_dsym_Bin:Nn %%%%%\let\m@Bin\d@@Bin
  \fi
```

When we get back to level 0, no delimiters, remove the stretch component of `\eqbinoffset`.

```
  \ifnum\lr@level<\@ne \eq@binoffset=\eqbinoffset\relax \fi
}

\ExplSyntaxOff

```

`\subsup@flag` `\ss@scan`

`\ss@scan` is called after a `\right` delimiter and looks ahead for sub and superscript tokens. If sub and/or superscripts are present, we adjust the line-ending penalty to distinguish the various cases (sub, sup, or both). This facilitates the later work of excising the sub/sup box and reattaching it with proper shifting.

Sub/Superscript measurement

BRM: There's possibly a problem here. When `\ss@scan` gets invoked after a `\left`...`\right` pair in the LHS during `\eq@measure`, it produces an extra box (marked with `\penalty` 3); Apparently `\eq@repack` expects only one for the LHS. The end result is `\eq@wdL` =¿ 0.0pt !!! (or at least very small)

```
\let\subsup@flag=\count@
\def\ss@delim@a@new#1#2#3#4#5{\xdef\right@delim@code{\number"#4#5}}
```

The argument of `\ss@scan` is an expanded form of a right-delimiter macro. We want to use the last three digits in the expansion to define `\right@delim@code`. The assignment to a temp register is just a way to scan away the leading digits that we don't care about.

```
\def\ss@scan#1{%
```

This part of the code.

```
  \begingroup
    \ss@delim@a@new #1%
  \endgroup
  \subsup@flag\@M \afterassignment\ss@scan@a \let\@let@token=}
\def\ss@scan@a{%
  \let\breqn@next\ss@scan@b
  \ifx\@let@token\sb \advance\subsup@flag\@ne\else
  \ifx\@let@token\@@subscript \advance\subsup@flag\@ne\else
  \ifx\@let@token\@@subscript@other \advance\subsup@flag\@ne\else
  \ifx\@let@token\sp \advance\subsup@flag\tw@\else
  \ifx\@let@token\@@superscript \advance\subsup@flag\tw@\else
  \ifx\@let@token\@@superscript@other \advance\subsup@flag\tw@\else
    \ss@finish
    \let\breqn@next\relax
  \fi\fi\fi\fi\fi\fi
  \breqn@next\@let@token
}
```

```
\ExplSyntaxOn
\def\ss@scan@b#1#2{#1{%
```

hack! coff!

```
  %%%%%\let\m@Bin\m@@Bin  \let\m@Rel\m@@Rel
```

```
  \cs_set_eq:NN \math_sym_Bin:Nn \math_isym_Bin:Nn
  \cs_set_eq:NN \math_sym_Rel:Nn \math_isym_Rel:Nn
  #2}\afterassignment\ss@scan@a \let\@let@token=}%
\ExplSyntaxOff
```

We need to keep following glue from disappearing—e.g.,, a thickmuskip or medmuskip from a following mathrel or mathbin symbol.

```
\def\ss@finish{%
  \@@vadjust{\penalty\thr@@}%
  \penalty\right@delim@code \penalty-\subsup@flag \keep@glue
}
```

`\eq@lrunpack` For `\eq@lrunpack` we need to break open a left-right box and reset it just in case it contains any more special breaks. After it is unpacked the recursion of `\eq@repack` will continue, acting on the newly created lines.

```
\def\eq@lrunpack{\setbox\z@\lastbox
```

We remove the preceding glue item and deactivate baselineskip for the next line, otherwise we would end up with three items of glue (counting parskip) at this point instead of the single one expected by our recursive repacking procedure.

```
  \unskip \nointerlineskip
```

Then we open box 0, take the left-right box at the right end of it, and break that open. If the line-ending penalty is greater than 10000, it means a sub and/or superscript is present on the right delimiter and the box containing them must be taken off first.

```
  \noindent\unhbox\z@ \unskip
  \subsup@flag-\lastpenalty \unpenalty
  \xdef\right@delim@code{\number\lastpenalty}%
  \unpenalty
  \ifnum\subsup@flag>\@M
    \advance\subsup@flag-\@M
    \setbox\tw@\lastbox
  \else \setbox\tw@\box\voidb@x
  \fi
  \setbox\z@\lastbox
  \ifvoid\tw@ \unhbox\z@
  \else \lrss@reattach % uses \subsup@flag, box\z@, box\tw@
  \fi
```

The reason for adding a null last line here is that the last line will contain parfillskip in addition to rightskip, and a final penalty of 10000 instead of $-1000N$ ($1 \leq N \leq 9$), which would interfere with the usual processing. Setting a null last line and discarding it dodges this complication. The penalty value -10001 is a no-op case in the case statement of `\eq@repacka`.

```
  \penalty-\@Mi\z@rule\@@par
  \setbox\z@\lastbox \unskip\unpenalty
%%{\showboxbreadth\maxdimen\showboxdepth99\showlists}%
}
```

`\lrss@reattach` Well, for a small self-contained computation, carefully hand-allocated dimens should be safe enough. But let the maintainer beware! This code cannot be arbitrarily transplanted or shaken up without regard to grouping and interaction with other hand-allocated dimens.

```
\dimendef\sub@depth=8 \dimendef\sup@base=6
\dimendef\prelim@sub@depth=4 \dimendef\prelim@sup@base=2
\def\sym@xheight{\fontdimen5\textfont\tw@}
\def\sup@base@one{\fontdimen13\textfont\tw@}
\def\sub@base@one{\fontdimen16\textfont\tw@}
\def\sub@base@two{\fontdimen17\textfont\tw@}
```

Note that only `\sup@drop` and `\sub@drop` come from the next smaller math style.

```
\def\sup@drop{\fontdimen18\scriptfont\tw@}
\def\sub@drop{\fontdimen19\scriptfont\tw@}
```

Provide a mnemonic name for the math axis fontdimen, if it's not already defined.

```
\providecommand{\mathaxis}{\fontdimen22\textfont\tw@}
```

Assumes box 2 contains the sub/sup and box 0 contains the left-right box. This is just a repeat of the algorithm in `tex.web`, with some modest simplifications from knowing that this is only going to be called at top level in a displayed equation, thus always mathstyle = uncramped displaystyle.

```
\def\lrss@reattach{%
  \begingroup
  % "The TeXbook" Appendix G step 18:
  \setlength\prelim@sup@base{\ht\z@-\sup@drop}%
  \setlength\prelim@sub@depth{\dp\z@ +\sub@drop}%
  \unhbox\z@
  \ifcase\subsup@flag      % case 0: this can't happen
  \or \lr@subscript   % case 1: subscript only
  \or \lr@superscript % case 2: superscript only
  \else \lr@subsup    % case 3: sub and superscript both
  \fi
  \endgroup
}
\def\lr@subscript{%
  \sub@depth\sub@base@one
  \ifdim\prelim@sub@depth>\sub@depth \sub@depth\prelim@sub@depth\fi
  \setlength\dim@a{\ht\tw@  -.8\sym@xheight}%
  \ifdim\dim@a>\sub@depth \sub@depth=\dim@a \fi
  \twang@adjust\sub@depth
  \lower\sub@depth\box\tw@
}
\def\lr@superscript{%
  \sup@base\sup@base@one
  \ifdim\prelim@sup@base>\sup@base \sup@base\prelim@sup@base\fi
  \setlength\dim@a{\dp\tw@ -.25\sym@xheight}%
  \ifdim\dim@a>\sup@base \sup@base\dim@a \fi
  \twang@adjust\sup@base
  \raise\sup@base\box\tw@
}
\def\lr@subsup{%
  \sub@depth\sub@base@two
  \ifdim\prelim@sub@depth>\sub@depth \sub@depth\prelim@sub@depth \fi
```

```
  \twang@adjust\sub@depth
  \lower\sub@depth\box\tw@
}
```

For delimiters that curve top and bottom, the twang factor allows horizontal shifting of the sub and superscripts so they don't fall too far away (or too close for that matter). This is accomplished by arranging for (e.g.,) `\right\rangle` to leave a penalty N in the math list before the subsup penalty that triggers `\lrss@reattach`, where N is the mathcode of `\rangle` (ignoring "small" variant).

```
\def\twang@adjust#1{%
  \begingroup
    \@ifundefined{twang@\right@delim@code}{}{%
      \setlength\dim@d{#1-\mathaxis}%
      % put an upper limit on the adjustment
      \ifdim\dim@d>1em \dim@d 1em \fi
      \kern\csname twang@\right@delim@code\endcsname\dim@d
    }%
  \endgroup
}
```

The method used to apply a "twang" adjustment is just an approximate solution to a complicated problem. We make the following assumptions that hold true, approximately, for the most common kinds of delimiters:

1. The right delimiter is symmetrical top to bottom.
2. There is an upper limit on the size of the adjustment.
3. When we have a superscript, the amount of left-skew that we want to apply is linearly proportional to the distance of the bottom left corner of the superscript from the math axis, with the ratio depending on the shape of the delimiter symbol.

. By symmetry, Assumption 3 is true also for subscripts (upper left corner). Assumption 2 is more obviously true for parens and braces, where the largest super-extended versions consist of truly vertical parts with slight bending on the ends, than it is for a `\rangle`. But suppose for the sake of expediency that it is approximately true for rangle symbols also.

Here are some passable twang factors for the most common types of delimiters in `cmex10`, as determined by rough measurements from magnified printouts.

```
vert bar, double vert: 0
       square bracket: -.1
          curly brace: -.25
          parenthesis: -.33
               rangle: -.4
```

Let's provide a non-private command for changing the twang factor of a given symbol.

```
\newcommand{\DeclareTwang}[2]{%
  \ifcat.\@nx#1\begingroup
    \lccode`\~=`#1\lowercase{\endgroup \DeclareTwang{~}}{#2}%
  \else
    \@xp\decl@twang#1?\@nil{#2}%
```

```
  \fi
}
```

Note that this is dependent on a fixed interpretation of the mathgroup number #4 .

```
\def\decl@twang#1#2#3#4#5#6#7\@nil#8{%
  \@namedef{twang@\number"#4#5#6}{#8}%
}
\DeclareTwang{\rangle}{-.4}
\DeclareTwang{)}{-.33}
\DeclareTwang{\rbrace}{-.25}
```

34 Series of expressions

The `dseries` environment is for a display containing a series of expressions of the form ‘A, B’ or ‘A and B’ or ‘A, B, and C’ and so on. Typically the expressions are separated by a double quad of space. If the expressions in a series don’t all fit in a single line, they are continued onto extra lines in a ragged-center format.

```
\newenvironment{dseries}{\let\eq@hasNumber\@True \@optarg\@dseries{}}{}%
\def\enddseries#1{\check@punct@or@qed}%
```

And the unnumbered version of same.

```
\newenvironment{dseries*}{\let\eq@hasNumber\@False \@optarg\@dseries{}}{}%
\@namedef{enddseries*}#1{\check@punct@or@qed}%
\@namedef{end@dseries*}{\end@dseries}%
\def\@dseries[#1]{%
```

Turn off the special breaking behavior of mathrels etc. for math formulas embedded in a `dseries` environment.

BRM: DS Expermient: Use alternative display setup.

```
%  \def\display@setup{\displaystyle}%
  \let\display@setup\dseries@display@setup
  % Question: should this be the default for dseries???
%  \let\eq@centerlines\@True
  \global\eq@wdCond\z@
```

BRM: use special layout for dseries

```
%  \@dmath[#1]%
  \@dmath[layout={M},#1]%
  \mathsurround\z@\@@math \penalty\@Mi
  \let\endmath\ends@math
  \def\premath{%
```

BRM: Tricky to cleanup space OR add space ONLY BETWEEN math!

```
    \ifdim\lastskip<.3em \unskip
    \else\ifnum\lastpenalty<\@M \dquad\fi\fi
}%
```

BRM: Tricky; if a subformula breaks, we’d like to start the next on new line!

```
  \def\postmath{\unpenalty\eq@addpunct \penalty\intermath@penalty \dquad \@ignoretrue}%
\ignorespaces
}
```

```
\def\end@dseries{%
  \unskip\unpenalty
  \@@endmath \mathsurround\z@ \end@dmath
}
```

BRM: Try this layout for dseries: Essentially layout i, but w/o limit to 1 line. And no fallback!

```
\def\eq@try@layout@M{%
  \edef\@parshape{\parshape 1 0pt \the\eq@linewidth\relax}%
  \eq@trial@b{M}{}%
}
```

BRM: Tricky to get right value here. Prefer breaks between formula if we've got to break at all.

```
%\def\intermath@penalty{-201}%
\def\intermath@penalty{-221}%
```

BRM: A bit tighter than it was (1em minus.25em)

```
%\newcommand\dquad{\hskip0.4em}
\newcommand\dquad{\hskip0.6em minus.3em}
\newcommand\premath{}\newcommand\postmath{}
```

Change the `math` environment to add `\premath` and `\postmath`. They are no-ops except inside a `dseries` environment.

Redefinition of math environment to take advantage of dseries env.

```
\renewenvironment{math}{%
  \leavevmode \premath
  \ifmmode\@badmath\else\@@math\fi
}{%
  \ifmmode\@@endmath\else\@badmath\fi
}
\def\ends@math#1{\check@punct@or@qed}
\def\end@math{%
  \ifmmode\@@endmath\else\@badmath\fi
  \postmath
}
```

35 Equation groups

For many equation groups the strategy is easy: just center each equation individually following the normal rules for a single equation. In some groups, each equation gets its own number; in others, a single number applies to the whole group (and may need to be vertically centered on the height of the group). In still other groups, the equations share a parent number but get individual equation numbers consisting of parent number plus a letter.

If the main relation symbols in a group of equations are to be aligned, then the final alignment computations cannot be done until the end of the group—i.e.,, the horizontal positioning of the first $n-1$ equations cannot be done immediately. Yet because of the automatic line breaking, we cannot calculate an initial value of RHS-max over the whole group unless we do a trial run on each equation first to find an RHS-max for that equation. Once we know RHS-group-max and LHS-group-max we must redo the trial set of each equation because they may affect the line breaks. If the second trial for an equation fails (one of its lines exceeds the available width), but the first one succeeded, fall back to the first trial, i.e., let that equation fall out of alignment with the rest of the group.

All right then, here is the general idea of the whole algorithm for group alignment. To start with, ignore the possibility of equation numbers so that our equation group has the form:

```
LHS[1] RHS[1,1] RHS[1,2] ... RHS[1,n[1]]
LHS[2] RHS[2,1] RHS[2,2] ... RHS[2,n[2]]
  ...
LHS[3] RHS[3,1] RHS[3,2] ... RHS[3,n[3]]
```

The number of RHS's might not be the same for all of the equations. First, accumulate all of the equation contents in a queue, checking along the way to find the maximum width of all the LHS's and the maximum width of all the RHS's. Call these widths maxwd_L and maxwd_R. Clearly if maxwd_L + maxwd_R is less than or equal to the available equation width then aligning all of the equations is going to be simple.

Otherwise we are going to have to break at least one of the RHS's and/or at least one of the LHS's. The first thing to try is using maxwd_L for the LHS's and breaking all the RHS's as needed to fit in the remaining space. However, this might be a really dumb strategy if one or more of the LHS's is extraordinarily wide. So before trying that we check whether maxwd_L exceeds some threshold width beyond which it would be unsensible not to break the LHS. Such as, max(one-third of the available width; six ems), or something like that. Or how about this? Compare the average LHS width and RHS width and divide up the available width in the same ratio for line breaking purposes.

BRM: Fairly broad changes; it mostly didn't work before (for me).

`\begin{dgroup}` produces a 'numbered' group The number is the next equation number. There are 2 cases:

- If ANY contained equations are numbered (`\begin{dmath}`), then they will be subnumbered: eg 1.1a and the group number is not otherwise displayed.
- If ALL contained equations are unnumbered (`\begin{dmath*}`) then the group, as a whole, gets a number displayed, using the same number placement as for equations.

`\begin{dgroup*}` produces an unnumbered group. Contained equations are numbered, or not, as normal. But note that in the mixed case, it's too late to force the unnumbered eqns to `\retry@with@number` We'll just do a simple check of dimensions, after the fact, and force a shiftnumber if we're stuck.

NOTE: Does this work for dseries, as well? (alignment?)

NOTE: Does `\label` attach to the expected thing?

For number placement We use shiftnumber placement on ALL equations if ANY equations need it, or if an unnumbered equation is too wide to be aligned, given that the group or other eqns are numbered. [does this latter case interract with the chosen alignment?]

For Alignment As currently coded, it tries to align on relations, by default. If LHS's are not all present, or too long, it switches to left-justify. Maybe there are other cases that should switch? Should there be a case for centered?

NOTE: Should there be some options to choose alignment?

\eq@group
\GRP@top

```
\let\eq@group\@False
\let\grp@shiftnumber\@False
\let\grp@hasNumber\@False
\let\grp@eqs@numbered\@False
\let\grp@aligned\@True
```

Definition of the dgroup environment.

```
\newenvironment{dgroup}{%
  \@dgroup@start@hook
  \let\grp@hasNumber\@True\@optarg\@dgroup{}%
}{%
  \end@dgroup
}
```

And the.

```
\newtoks\GRP@queue
\newenvironment{dgroup*}{%
  \let\grp@hasNumber\@False\@optarg\@dgroup{}%
}{%
  \end@dgroup
}
\def\@dgroup[#1]{%
⟨trace⟩   \breqn@debugmsg{=== DGROUP ===================================================}%
  \let\eq@group\@True \global\let\eq@GRP@first@dmath\@True
  \global\GRP@queue\@emptytoks \global\setbox\GRP@box\box\voidb@x
  \global\let\GRP@label\@empty
  \global\grp@wdL\z@\global\grp@wdR\z@\global\grp@wdT\z@
  \global\grp@linewidth\z@\global\grp@wdNum\z@
  \global\let\grp@eqs@numbered\@False
  \global\let\grp@aligned\@True
  \global\let\grp@shiftnumber\@False
  \eq@prelim
  \setkeys{breqn}{#1}%
  \if\grp@hasNumber \grp@setnumber \fi
}
\def\end@dgroup{%
  \EQ@displayinfo \grp@finish
  \if\grp@hasNumber\grp@resetnumber\fi
}
```

If the amsmath package is not loaded the parentequation counter will not be defined.

```
\@ifundefined{c@parentequation}{\newcounter{parentequation}}{}
```

Init.

```
\global\let\GRP@label\@empty
\def\add@grp@label{%
  \ifx\@empty\GRP@label
  \else \GRP@label \global\let\GRP@label\@empty
  \fi
}
```

Before sending down the 'equation' counter to the subordinate level, set the current number in `\EQ@numbox`. The `\eq@setnumber` function does everything we need here. If the child equations are unnumbered, `\EQ@numbox` will retain the group number at the end of the group.

```
\def\grp@setnumber{%
  \global\let\GRP@label\next@label \global\let\next@label\@empty
  % Trick \eq@setnumber to doing our work for us.
  \let\eq@hasNumber\@True
  \eq@setnumber
```

Define `\theparentequation` equivalent to current `\theequation`. `\edef` is necessary to expand the current value of the equation counter. This might in rare cases cause something to blow up, in which case the user needs to add `\protect`.

```
  \global\sbox\GRP@numbox{\unhbox\EQ@numbox}%
  \grp@wdNum\eq@wdNum
  \let\eq@hasNumber\@False
  \let\eq@number\@empty
  \eq@wdNum\z@
%
  \protected@edef\theparentequation{\theequation}%
  \setcounter{parentequation}{\value{equation}}%
```

And set the equation counter to 0, so that the normal incrementing processes will produce the desired results if the child equations are numbered.

```
  \setcounter{equation}{0}%
  \def\theequation{\theparentequation\alph{equation}}%
⟨trace⟩  \breqn@debugmsg{Group Number \theequation}%
}
```

At the end of a group, need to reset the equation counter.

```
\def\grp@resetnumber{%
  \setcounter{equation}{\value{parentequation}}%
}
\newbox\GRP@box
\newbox\GRP@wholebox
```

Save data for this equation in the group

- push the trial data onto end of `\GRP@queue`.
- push an hbox onto the front of `\GRP@box` containing: `\EQ@box`, `\EQ@copy`, `\penalty 1` and `\EQ@numbox`.

`\grp@push` For putting the equation on a queue.

```
\def\grp@push{%
  \global\GRP@queue\@xp\@xp\@xp{\@xp\the\@xp\GRP@queue
    \@xp\@elt\@xp{\EQ@trial}%
  }%
  \global\setbox\GRP@box\vbox{%
    \hbox{\box\EQ@box\box\EQ@copy\penalty\@ne\copy\EQ@numbox}%
    \unvbox\GRP@box
  }%
  \EQ@trial
```

```
  \if\eq@isIntertext\else
    \ifdim\eq@wdL>\grp@wdL \global\grp@wdL\eq@wdL \fi
    \ifdim\eq@wdT>\grp@wdT \global\grp@wdT\eq@wdT \fi
    \setlength\dim@a{\eq@wdT-\eq@wdL}%
    \ifdim\dim@a>\grp@wdR \global\grp@wdR\dim@a \fi
    \ifdim\eq@linewidth>\grp@linewidth \global\grp@linewidth\eq@linewidth\fi
    \if\eq@hasNumber
       \global\let\grp@eqs@numbered\@True
       \ifdim\eq@wdNum>\grp@wdNum\global\grp@wdNum\eq@wdNum\fi
    \fi
    \if\EQ@hasLHS\else\global\let\grp@aligned\@False\fi
    \if D\eq@layout \global\let\grp@aligned\@False\fi % Layout D (usually) puts rel on 2nd line.
    \if\eq@shiftnumber\global\let\grp@shiftnumber\@True\fi % One eq shifted forces all.
  \fi
}
```

`\grp@finish` Set accumulated equations from a `dgroup` environment.

BRM: Questionable patch!! When processing the `\GRP@queue`, put it into a `\vbox`, then `\unvbox` it. This since there's a bizarre problem when the `\output` routine gets invoked at an inopportune moment: All the not-yet-processed `\GRP@queue` ends up in the `\@freelist` and bad name clashes happen. Of course, it could be due to some other problem entirely!!!

```
\def\grp@finish{%
%  \debug@box\GRP@box
%  \breqn@debugmsg{\GRP@queue: \the\GRP@queue}%
```

== Now that we know the collective measurements, make final decision about alignment & shifting. Check if alignment is still possible

```
  \setlength\dim@a{\grp@wdL+\grp@wdR-4em}% Allowance for shrink?
  \if\grp@aligned
    \ifdim\dim@a>\grp@linewidth
      \global\let\grp@aligned\@False
    \fi
  \fi
```

If we're adding an unshifted group number that equations didn't know about, re-check shifting

```
  \addtolength\dim@a{\grp@wdNum }% Effective length
  \if\grp@shiftnumber
  \else
    \if\@And{\grp@hasNumber}{\@Not\grp@eqs@numbered}
      \ifdim\dim@a>\grp@linewidth
        \global\let\grp@shiftnumber\@True
      \fi
    \fi
  \fi
```

If we can still align, total width is sum of maximum LHS & RHS

```
  \if\grp@aligned
     \global\grp@wdT\grp@wdL
     \global\advance\grp@wdT\grp@wdR
  \fi
⟨*trace⟩
```

```
  \breqn@debugmsg{======= DGROUP Formatting
   \MessageBreak==== \grp@wdL=\the\grp@wdL, \grp@wdR=\the\grp@wdR
   \MessageBreak==== Shift Number=\theb@@le\grp@shiftnumber, Eqns. numbered=\theb@@le\grp@eqs@numbered
   \MessageBreak==== Aligned=\theb@@le\grp@aligned
   \MessageBreak==== \grp@wdNum=\the\grp@wdNum}%
⟨/trace⟩
```

BRM: Originally this stuff was dumped directly, without capturing it in a `\vbox`

```
  \setbox\GRP@wholebox\vbox{%
    \let\@elt\eqgrp@elt
    \the\GRP@queue
  }%
```

If we're placing a group number (not individual eqn numbers) NOTE: For now, just code up LM number NOTE: Come back and handle other cases. NOTE: Vertical spacing is off, perhaps because of inter eqn. glue

A bit of a hack to get the top spacing correct. Fix this logic properly some day. Also, we do the calculation in a group for maximum safety.

```
  \global\let\eq@GRP@first@dmath\@True
  \begingroup
  \dmath@first@leftskip
  \eq@topspace{\vskip\parskip}%
  \endgroup
  \if\@And{\grp@hasNumber}{\@Not{\grp@eqs@numbered}}%
%     \eq@topspace{\vskip\parskip}%
    \if\grp@shiftnumber
      \copy\GRP@numbox \penalty\@M
      \kern\eqlineskip
    \else
      \setlength\dim@a{%
        (\ht\GRP@wholebox+\dp\GRP@wholebox+\ht\GRP@numbox-\dp\GRP@numbox)/2}%
      \setlength\skip@c{\dim@a-\ht\GRP@numbox}%
      \vglue\skip@c% NON discardable
      \copy\GRP@numbox \penalty\@M
⟨*trace⟩
\breqn@debugmsg{GROUP NUMBER: preskip:\the\skip@c,  postkern:\the\dim@a, height:\the\ht\GRP@wholebox,
  \MessageBreak==== box height:\the\ht\GRP@numbox, box depth:\the\dp\GRP@numbox}%
⟨/trace⟩
      \kern-\dim@a
      \kern-\abovedisplayskip % To cancel the topspace above the first eqn.
    \fi
  \fi
⟨*trace⟩
%\debug@box\GRP@wholebox
⟨/trace⟩
  \unvbox\GRP@wholebox
  \let\@elt\relax
```

We'd need to handle shifted, right number here, too!!!

```
  \eq@botspace % not needed unless bottom number?
}
```

`\eqgrp@elt` Mission is to typeset the next equation from the group queue.

The arg is an `\EQ@trial`

```
\def\eqgrp@elt#1{%
  \global\setbox\GRP@box\vbox{%
    \unvbox\GRP@box
    \setbox\z@\lastbox
    \setbox\tw@\hbox{\unhbox\z@
      \ifnum\lastpenalty=\@ne
      \else
        \global\setbox\EQ@numbox\lastbox
      \fi
      \unpenalty
      \global\setbox\EQ@copy\lastbox
      \global\setbox\EQ@box\lastbox
    }%
  }%
  \begingroup \let\eq@botspace\relax
  #1%
  \if\eq@isIntertext
    \vskip\belowdisplayskip
    \unvbox\EQ@copy
  \else
    \grp@override
    \eq@finish
  \fi
  \endgroup
}
```

Override the `\eq@trial` data as needed for this equation in this group NOTE: w/ numbering variations (see above), we may need to tell `\eq@finish` to allocate space for a number, but not actually have one

```
\def\grp@override{%
```

For aligned (possibly becomes an option?) For now ASSUMING we started out as CLM!!!

```
  \def\eqindent{I}%
```

compute nominal left for centering the group

```
  \setlength\dim@a{(\grp@linewidth-\grp@wdT)/2}%
```

Make sure L+R not too wide; should already have unset alignment

```
  \ifdim\dim@a<\z@\dim@a=\z@\fi
  \dim@b\if L\eqnumside\grp@wdNum\else\z@\fi
```

make sure room for number on left, if needed.

```
  \if\grp@shiftnumber\else
    \ifdim\dim@b>\dim@a\dim@a\dim@b\fi
  \fi
  \if\grp@aligned
    \addtolength\dim@a{\grp@wdL-\eq@wdL}%
  \fi
  \mathindent\dim@a
```

```
  \ifdim\dim@b>\dim@a
    \let\eq@shiftnumber\@True
  \fi
```

Could set `\def\eqnumplace{T}` (or even (m) if indentation is enough).

NOTE: Work out how this should interact with the various formats!!! NOTE: should recognize the case where the LHS's are a bit Wild, and then do simple left align (not on relation)

```
}
```

36 The `darray` environment

There are two potential applications for darray. One is like eqnarray where the natural structure of the material crosses the table cell boundaries, and math operator spacing needs to be preserved across cell boundaries. And there is also the feature of attaching an equation number to each row. The other application is like a regular array but with automatic displaystyle math in each cell and better interline spacing to accommodate outsize cell contents. In this case it is difficult to keep the vert ruling capabilities of the standard `array` environment without redoing the implementation along the lines of Arseneau's `tabls` package. Because the vert ruling feature is at cross purposes with the feature of allowing interline stretch and page breaks within a multiline array of equations, the `darray` environment is targeted primarily as an alternative to `eqnarray`, and does not support vertical ruling.

Overall strategy for `darray` is to use `\halign` for the body. In the case of a group, use a single halign for the whole group!

What about intertext?

That's the most reliable way to get accurate column widths. Don't spread the halign to the column width, just use the natural width. Then, if we repack the contents of the halign into `\EQ@box` and `\EQ@copy`, as done for dmath, and twiddle a bit with the widths of the first and last cell in each row, we can use the same algorithms for centering and equation number placement as dmath! As well as handling footnotes and vadjust objects the same way.

We can't just use `\arraycolsep` for `darray`, if we want to be able to change it without screwing up interior arrays. So let's make a new colsep variable. The initial value is '2em, but let it shrink if necessary'.

```
\newskip\darraycolsep \darraycolsep 20pt plus1fil minus12pt
```

Let's make a nice big default setup with eighteen columns, split up into six sets of lcr like `eqnarray`.

```
\newcount\cur@row \newcount\cur@col
\def\@tempa#1#2#3{%
  \cur@col#1 \hfil
  \setbox\z@\hbox{$\displaystyle####\m@th$}\@nx\col@box
  \tabskip\z@skip
  &\cur@col#2 \hfil
  \setbox\z@\hbox{$\displaystyle\mathord{}####\mathord{}\m@th$}\@nx\col@box
  \hfil
  &\cur@col#3 \setbox\z@\hbox{$\displaystyle####\m@th$}\@nx\col@box
```

```
  \hfil\tabskip\darraycolsep
}
\xdef\darray@preamble{%
  \@tempa 123&\@tempa 456&\@tempa 789%
  &\@tempa{10}{11}{12}&\@tempa{13}{14}{15}&\@tempa{16}{17}{18}%
  \cr
}
\@ifundefined{Mathstrut@}{\let\Mathstrut@\strut}{}
\def\darray@cr{\Mathstrut@\cr}
\def\col@box{%
⟨*trace⟩
%\breqn@debugmsg{Col \number\cur@row,\number\cur@col: \the\wd\z@\space x \the\ht\z@+\the\dp\z@}%
⟨/trace⟩
  \unhbox\z@
}
\newenvironment{darray}{\@optarg\@darray{}}{}
\def\@darray[#1]{%
⟨trace⟩  \breqn@debugmsg{=== DARRAY ==================================================}%
  \if\eq@group\else\eq@prelim\fi
```

Init the halign preamble to empty, then unless the 'cols' key is used to provide a non-null preamble just use the default darray preamble which is a multiple lcr.

```
  \global\let\@preamble\@empty
  \setkeys{breqn}{#1}%
  \the\eqstyle \eq@setnumber
  \ifx\@preamble\@empty \global\let\@preamble\darray@preamble \fi
  \check@mathfonts
  % \let\check@mathfonts\relax % tempting, but too risky
  \@xp\let\csname\string\ \endcsname\darray@cr
  \setbox\z@\vbox\bgroup
  \everycr{\noalign{\global\advance\cur@row\@ne}}%
  \tabskip\z@skip \cur@col\z@
  \global\cur@row\z@
  \penalty\@ne % flag for \dar@repack
  \halign\@xp\bgroup\@preamble
}
```

Assimilate following punctuation.

```
\def\enddarray#1{\check@punct@or@qed}
\def\end@darray{%
  \ifvmode\else \eq@addpunct \Mathstrut@\fi\crcr \egroup
  \dar@capture
  \egroup
}
```

The `\dar@capture` function steps back through the list of row boxes and grinds them up in the best possible way.

```
\def\dar@capture{%
%% \showboxbreadth\maxdimen\showboxdepth99\showlists
  \eq@wdL\z@ \eq@wdRmax\z@
  \dar@repack
}
```

The \dar@repack function is a variation of \eq@repack.

```
\def\dar@repack{%
  \unpenalty
  \setbox\tw@\lastbox
%\batchmode{\showboxbreadth\maxdimen\showboxdepth99\showbox\tw@}\errorstopmode
  \global\setbox\EQ@box\hbox{%
    \hbox{\unhcopy\tw@\unskip}\penalty-\@M \unhbox\EQ@box}%
  \global\setbox\EQ@copy\hbox{%
    \hbox{\unhbox\tw@\unskip}\penalty-\@M \unhbox\EQ@copy}%
  \unskip
  \ifcase\lastpenalty \else\@xp\@gobble\fi
  \dar@repack
}
```

37 Miscellaneous

The \condition command. With the star form, set the argument in math mode instead of text mode. In a series of conditions, use less space between members of the series than between the conditions and the main equation body.

WSPR: tidied/fixed things up as it made sense to me but might have broken something else!

```
\newskip\conditionsep \conditionsep=10pt minus5pt%
\newcommand{\conditionpunct}{,}
```

\condition

```
\newcommand\condition{%
  \begingroup\@tempswatrue
    \@ifstar{\@tempswafalse \condition@a}{\condition@a}}
```

\condition@a

```
\newcommand\condition@a[2][\conditionpunct]{%
  \unpenalty\unskip\unpenalty\unskip % BRM Added
  \hbox{#1}%
  \penalty -201\relax\hbox{}% Penalty to allow breaks here.
  \hskip\conditionsep
  \setbox\z@\if@tempswa\hbox{#2}\else\hbox{$\textmath@setup #2$}\fi
```

BRM's layout is achieved with this line commented out but it has the nasty side-effect of shifting the equation number to the next line:

```
%  \global\eq@wdCond\wd\z@
  \usebox\z@
  \endgroup}
```

The dsuspend environment. First the old one that didn't work.

```
\newenvironment{XXXXdsuspend}{%
  \global\setbox\EQ@box\vbox\bgroup \@parboxrestore
```

If we are inside a list environment, `\displayindent` and `\displaywidth` give us `\@totalleftmargin` and `\linewidth`.

```
    \parshape 1 \displayindent \displaywidth\relax
    \hsize=\columnwidth \noindent\ignorespaces
}{%
  \par\egroup
```

Let's try giving `\EQ@box` the correct height for the first line and `\EQ@copy` the depth of the last line.

```
  \global\setbox\GRP@box\vbox{%
    \vbox{\copy\EQ@box\vtop{\unvbox\EQ@box}}%
    \unvbox\GRP@box
  }%
```

Need to add a dummy element to `\GRP@queue`.

```
  \global\GRP@queue\@xp{\the\GRP@queue
    \@elt{\gdef\EQ@trial{}}%
  }%
}
```

And then the one that does work.

```
\newenvironment{dsuspend}{%
  \global\setbox\EQ@box\vbox\bgroup \@parboxrestore
    \parshape 1 \displayindent \displaywidth\relax
    \hsize=\columnwidth \noindent\ignorespaces
}{%
  \par\egroup
  \global\setbox\GRP@box\vbox{%
    \hbox{\copy\EQ@box\vtop{\unvbox\EQ@box}}%
    \unvbox\GRP@box
  }%
  \global\GRP@queue\@xp{\the\GRP@queue
%     \@elt{\gdef\EQ@trial{\let\eq@isIntertext\@True}}%
     \@elt{\let\eq@isIntertext\@True}%
  }%
}
```

Allow `\intertext` as a short form of the `dsuspend` environment; it's more convenient to write, but it doesn't support embedded verbatim because it reads the material as a macro argument. To support simultaneous use of amsmath and breqn, the user command `\intertext` is left alone until we enter a breqn environment.

```
\newcommand\breqn@intertext[1]{\dsuspend#1\enddsuspend}
```

`\*`
`\discretionarytimes`

Discretionary times sign. Standard LaTeX definition serves only for inline math. Should the thin space be included? Not sure.

```
\renewcommand{\*}{%
  \if@display
```

Since `\eq@binoffset` is mu-glue, we can't use it directly with `\kern` but have to measure it separately in a box.

```
    \setbox\z@\hbox{\mathsurround\z@$\mkern\eq@binoffset$}%
```

```
    \discretionary{}{%
      \kern\the\wd\z@ \textchar\discretionarytimes
    }{}%
    \thinspace
  \else
    \discretionary{\thinspace\textchar\discretionarytimes}{}{}%
  \fi
}
```

This is only the symbol; it can be changed to some other symbol if desired.

```
\newcommand{\discretionarytimes}{\times}
```

`\nref` This is like `\ref` but doesn't apply font changes or other guff if the reference is undefined. And it is fully expandable for use as a label value.

> **Can break with Babel if author uses active characters in label key; need to address that**
>
> **mjd,1999/01/21**
>
> .

```
\def\nref#1{\@xp\@nref\csname r@#1\endcsname}
\def\@nref#1#2{\ifx\relax#1??\else \@xp\@firstoftwo#1\fi}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

38 Compatibility

lineno (or at the very least, allow documents to compile!)

```
\AtBeginDocument{%
  \@ifpackageloaded{lineno}{%
    \g@addto@macro\@dmath@start@hook{\nolinenumbers}%
    \g@addto@macro\@dgroup@start@hook{\nolinenumbers}%
  }{}%
}
```

39 Wrap-up

The usual endinput.

```
⟨/package⟩
```

40 To do

1. Alignment for equation groups.
2. Use dpc's code for package options in keyval form.
3. Encapsulate "break math" into a subroutine taking suitable arguments.

4. Need a density check for layout S when linewidth is very small.
5. Make `:=` trigger a warning about using `\coloneq` instead.
6. Ill-centered multiline equation (three-line case) in test008.
7. Attaching a single group number.
8. Make sure to dump out box registers after done using them.
9. Do the implementation for `\eq@resume@parshape`.
10. Check on stackrel and buildrel and relbar and ???.
11. Test math symbols at the beginning of array cells.
12. Test \\md in and out of delims.
13. Framing the equation body: the parshape and number placement need adjusting when a frame is present.
14. Cascading line widths in list env.
15. Noalign option for dmath = multline arrangement?
16. Nocompact option, suggested 1998/05/19 by Andrew Swann.
17. `\delbreak` cmd to add discretionary space at a break within delimiters.
18. Reduce above/below skip when the number is shifted.
19. Need a `\middelim` command for marking a delimiter symbol as nondirectional if it has an innate directionality `()[]` etc..
20. `\xrightarrow` from amsmath won't participate in line breaking unless something extra is done. Make `\BreakingRel` and `\BreakingBin` functions?
21. Placement of number in an indented quotation or abstract.
22. If $LHSwd > 2em$, it might be a good idea to try with eq@indentstep = 2em before shifting the number. Currently this doesn't happen if the first trial pass (without the number) succeeds with $indentstep = LHSwd > 2em$.
23. Read past `\end{enumerate}` when checking for `\end{proof}`?
24. Look into using a "qed-list" of environment names instead of checking the existence of `\proofqed`.
25. Pick up the vadjust/footnote/mark handling.
26. Forcing/prohibiting page breaks after/before an equation.
27. Adding a spanner brace on the left and individual numbers on the right (indy-numbered cases).

28. Provide `\shiftnumber`, `\holdnumber` to override the decision.

29. Provide a mechanism for adjusting the vertical position of the number. Here a version-specific selection macro would be useful.

```
\begin{dmath}[
  style={\foredition{1}{\raisenumber{13pt}}}
]
```

30. Add an alignleft option for an equation group to mean, break and align to a ladder layout as usual within the equations, but for the group alignment used the leftmost point (for equations that don't have an LHS, this makes no difference).

31. Test with Arseneau's wrapfig for parshape/everypar interaction.

32. Fix up the macro/def elements.

33. Convert the literal examples in section 'Equation types and forms' to typeset form.

34. Compile comparison-examples: e.g.,, a standard equation env with big left-right objects that don't shrink, versus how shrinking can allow it to fit.

35. Frame the "figures" since they are mostly text.

Possible enhancements:

1. Provide a `pull` option meaning to pull the first and last lines out to the margin, like the `multline` environment of the amsmath package. Maybe this should get an optional argument, actually, to specify the amount of space left at the margin.

2. With the draft option, one would like to see the equation labels in the left margin. Need to check with the showkeys package.

3. Options for break preferences: if there's not enough room, do we first shift the number, or first try to break up the equation body?. In an aligned group, does sticking to the group alignment take precedence over minimizing the number of line breaks needed for individual equations?. And the general preferences probably need to be overridable for individual instances.

4. Extend suppress-breaks-inside-delimiters support to inline math (suggestion of Michael Doob).

5. Use belowdisplayshortskip above a dsuspend fragment if the fragment is only one line and short enough compared to the equation line above it.

6. Add `\eqfuzz` distinct from `\hfuzz`. Make use of it in the measuring phase.

7. Provision for putting in a 'continued' note.

8. Conserve box mem: modify frac, sub, sup, overline, underline, sqrt, to turn off `\bin@break` and (less urgently) `\rel@break`.

9. More explicit support for Russian typesetting conventions (cf Grinchuk article).
10. With package option `refnumbers`, leave unnumbered all uncited equations, even if they are not done with the star form (Bertolazzi's easyeqn idea).
11. In an equation group, use a vertical bracket with the equation number to mark the lines contained in that equation.
12. For a two-line multline thingamabob, try to make sure that the lines overlap in the middle by 2 em or whatever (settable design variable).
13. Provide a separate vertical column for the principal mathrel symbols and center them within the column if they aren't all the same width. Maybe an option for `dmath`: relwidth=x, so that two passes are not required to get the max width of all the mathrels. Or, no, just require it to be an halign or provide a macro to be applied to all the shorter rels:

```
lhs \widerel{19pt}{=} ...
    \xrightarrow{foo} ...
```

14. try to use vadjust for keepglue

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols

K

L

V

W

X

Z