Plain TEX's \ooalign

Udo Wermuth

Abstract

This article looks at a single macro defined in the file `plain.tex`: `\ooalign`. Its definition is analyzed and several examples for its use are presented.

1 Introduction

On page 9 of *The TEXbook* [1] the author, Donald E. Knuth, makes a promise: "TEX understands about 900 control sequences as part of its built-in vocabulary, and all of them are explained in this manual somewhere." And on the next page the number 900 is split into two parts: "In the following chapters we shall frequently discuss 'plain TEX' format, which is a set of about 600 basic control sequences that are defined in Appendix B. These control sequences, together with the 300 or so primitives, are usually present when TEX begins to process a manuscript; that is why TEX claims to know roughly 900 control sequences when it starts." So the `plain` format is covered in the book at least by Appendix B.

It's necessary to write "at least" in the previous sentence since we read in Appendix B on page 343, "When we come to macros whose usage has not yet been explained — for example, somehow `\vglue` and `\beginsection` never made it into Chapters 1 through 27 — we shall consider them from a user's viewpoint; but most of the time we shall be addressing the issues from the standpoint of a macro designer." (Note however, in Chapter 23, page 259f., a different, more complex `\beginsection` from a real-world book design is discussed.)

Another macro of the `plain` format that never made it into the numbered chapters of the book is `\ooalign`. It is defined in Appendix B on page 356. Well, there is an important difference between the two macros `\ooalign` and `\beginsection`. The first is definitely something for macro writers, the second serves users who want to enter text. But the latter is only used in a test file of Appendix B, not in *The TEXbook* itself. (As mentioned above, a more complex macro is shown in Chapter 23 and later in Appendix E, page 418f., Knuth explains the non-`plain` macro `\beginchapter` that is used in the creation of *The TEXbook*.) On the other hand the macro `\ooalign` is used several times in the `plain` format.

Nevertheless it seems that the macro `\ooalign` is a little bit hidden through this treatment. Like the simple `\beginsection`, other authors of books that concentrate on TEX's primitives and the `plain` format hardly mention the macro `\ooalign`. I surmise `\ooalign` is seen by most people as a support macro that Knuth uses to define an accent (`\c`: ¸) and the copyright symbol (`\copyright`: ©) although it appears in macros for math modes too.

But I assume that most "support macros" in the `plain` format are written with the '`@`' symbol in their names to make them private to the format. As `\ooalign` doesn't contain the '`@`' in its name, it might indicate that Knuth wants people other than TEX experts to consider `\ooalign` a useful tool.

Contents. Section 2 shows and explains the definition of the macro `\ooalign`. Then we look at simple examples of its use in section 3. The constructed symbols are meant to be entered in normal text. Symbols that should be used in a math mode need more complex code as section 4 shows. Section 5 increases the complexity and handles a weird use case.

2 \ooalign's definition

As the macro name implies the macro has something to do with alignment. Here is the definition as it appears in `plain.tex`:

```
\def\ooalign{\lineskiplimit-\maxdimen \oalign}
```

This definition is followed by a comment: "chars over each other". Okay, one of the two 'o's vanishes in the replacement text to leave us with another unknown macro of the `plain` format: `\oalign`. As we will see in a moment, `\oalign` calls another unknown macro, `\ialign`, so I assume it's best to study the whole chain of macro calls in one code block.

Note that the macro `\ooalign` has no parameters. Of course, we expect that "characters" follow so that the comment makes sense. The truth is that it's `\oalign` that has a parameter but for the users of `\ooalign` it looks as if this macro takes an argument. (Some people call this a *pseudo-parameter* for `\ooalign`.) This technique is used if certain parameter settings must be done before the argument is digested by TEX. Here the `\lineskiplimit` is set.

This assignment is a bit of a surprise in the above definition. The dimension `\lineskiplimit` occurs usually in a triumvirate with the two glue parameters `\baselineskip` and `\lineskip`. Together they control the *interline glue* design, i.e., the vertical line spacing; see [1], p. 78f.

Interline glue. When TEX typesets text in paragraphs it observes that the distances between adjacent lines have a certain value stored in the glue parameter `\baselineskip`. To get this distance TEX looks for every pair of lines at the depth of the first

doi.org/10.47397/tb/44-3/tb138wermuth-ooalign

line and the height of the second. Both values are subtracted from the \baselineskip to get the value for the interline glue. TEX adds this glue value between the two lines to keep the baselines exactly \baselineskip apart. (Usually this glue parameter has only a natural width and no stretch or shrink component. Nevertheless, TEX allows this kind of flexibility here.)

But there is an exception. If the natural width of the computed interline glue is smaller than the dimension \lineskiplimit, TEX ignores the computed value and inserts glue with the value of the parameter \lineskip between the lines.

The format plain sets \baselineskip to 12 pt, \lineskip to 1 pt, and \lineskiplimit to 0 pt. So, in plain TEX lines are kept 1 pt apart if the natural width of the computed interline glue becomes negative, i.e., if the lines might overprint each other. ("Might" because the places with large depth in the first line and large height in the second might not be at the same position inside the lines.)

The macro chain. Let's return to the definition of \ooalign and describe it in detail with comments.

```
\def\ooalign{% ``chars over each other''
 \lineskiplimit=-\maxdimen %-\maxdimen: smallest
 \oalign}%                       dimension in TeX
```

From the discussion about interline glue we expect that the macro \oalign sets the two missing parameters. The following code from plain.tex — extended by my comments — meets our expectation.

```
\def\oalign#1{% #1: cell entries for \ialign
 \leavevmode   % in horizontal mode: do nothing;
          % otherwise switch to horizontal mode
 \vtop{%   put the alignment into an unbreakable
               % box with height of the 1st row
    \baselineskip=0pt % switch off \baselineskip
         % and set a default value to \lineskip:
    \lineskip=.25ex %  a quarter of the x-height
    \ialign{##\crcr   % preamble of a one-column
        #1\crcr}}%        initialized alignment
```

Before we discuss the settings of the three parameters for the interline glue, let's look at the macro that is called at the end of \oalign.

```
\def\ialign{% initialized horizontal alignment
 \everycr={}\tabskip=0pt \halign}
```

The primitive \halign reacts to two TEX parameters with their current settings: First, the token list \everycr is applied directly after the preamble and at every \cr (or non-redundant \crcr); see [1], p. 275. Second, TEX applies the current value of the \tabskip glue from outside of an \halign to the left of the alignment; see [1], p. 238. So it makes sense to first reset both values to control the appearance of the horizontal alignment.

Now we can also describe the pseudo-parameter of \ooalign: It consists of a sequence of rows for a horizontal alignment. Each row entry except the last must end with \cr (or \crcr).

Why \lineskip? If you paid attention to the discussion about interline glue then you might wonder why \oalign sets \lineskip to a nonzero value. The macro \ooalign assigns the smallest acceptable dimension of TEX to \lineskiplimit, so no computation can give a smaller value: TEX never discards its computed value for the interline glue and thus it never applies the value of \lineskip.

Of course, there is a reason that \lineskip is set by \oalign. The macro \ooalign has a sibling \o@lign that's private to the plain format.

```
\def\o@lign{\lineskiplimit=0pt \oalign}
```

Now we can see why \lineskiplimit is set outside of \oalign. In \o@lign, \lineskip is always applied if the interline glue is negative; and this is always the case, as \baselineskip is 0 pt. Thus, every nonzero depth and height of the pair of lines makes TEX insert glue with a natural height of 0.25 ex.

The macro \o@lign is private to plain as it has fewer use cases than \ooalign. The macro only occurs in the definition of two accents. As with the above-mentioned cedilla accent, where \ooalign is used, these accents are set below the baseline.

So, what does \ooalign do? It takes material provided as cell entries for an alignment and TEX typesets these entries as one column with baselines that are 0 pt apart. Or in other words: TEX prints the second row on top of the first, then the third on top of the combination of the first and second, etc. As the entries are characters, we get "chars over each other". The material in the first row determines the height of the result, that in the last row its depth.

But recall that TEX determines in a horizontal alignment the widest cell entry in all the rows of a column and uses this width for the other rows in this column too. Users of \ooalign must make sure that TEX finds enough glue in the cell entries to accomplish this.

Next, \ooalign sets a lot of parameters, sometimes to unusual values. For example, we certainly want to return to the current interline glue parameters as soon as \ooalign finishes its job. Thus, users of the macro should call it only inside a group.

3 Simple examples

The Introduction mentions that \ooalign is used to create the copyright symbol. So, for our first example, let's see how to generate this symbol. It's

based on two characters of Computer Modern Roman. First, a circle is needed, called \Orb: ◯. Second, we need the letter 'c' from the default font cmr10. (I present the code in my style.)

```
\def\copyright{% typeset a 'c' inside a circle
 {\ooalign{%      call \ooalign inside a group
   \hfil  % start 1st row; center its contents
     \raise0.07ex\hbox{c}% output 'c' (raised)
     \hfil\crcr            % end the first row
   \Orb\crcr}}%   the 2nd row with the circle
```

The \crcr after \Orb is not required as the code of \oalign shows. But it doesn't hurt to add it and to make the code for the symbol easier to understand.

The value 0.07 ex that occurs in the code of the macro might have been found by trial and error or by insight into the definitions of the involved characters. So either you have to experiment to find the correct value or look up the definitions of the involved characters in [2] and do some computations.

We could have coded the symbol a little bit different but with the same output:

```
\def\Copyright{% typeset a 'c' inside a circle
 {\ooalign{%      call \ooalign inside a group
   \Orb\crcr     % the 1st row with the circle
   \hfil  % start 2nd row; center its contents
     \raise0.07ex\hbox{c}% output 'c' (raised)
     \hfil\crcr}}%       % end the second row
```

Here is a direct comparison: © © entered as \copyright and \Copyright. But remember what was stated in section 2: The symbols aren't identical. Let's add a superscript and a subscript to both symbols: a) ©* ©* b) ©+ ©+. Thus the sequence of the rows is important if the created symbol is later used in certain situations.

Many symbols are based on a circle and another character. Let's create some textile care labels as defined by the organization GINETEX. We want to write a macro that produces the symbol Ⓟ (dry clean with tetrachloroethylene (PCE) only): A sans-serif font is combined with the circle called \Orb.

```
\font\DCfont=cmss8 % sans-serif font for the 'P'
\def\PCE{{%           open a group for \ooalign
 \ooalign{%  horizontal alignment with two rows
  \hidewidth\hbox{\DCfont\kern0.06em P}%
    \hidewidth\cr % 1st row, centered contents
  \raise0.1ex\hbox{\Orb}\cr % 2nd row: a circle
}}%     end \ooalign, the group, and the macro
```

As mentioned above, the symbol is created using row entries of an alignment. Hence the usual techniques known from horizontal alignments can be used. Here we suppress the width of the letter with \hidewidth and apply \cr instead of \crcr.

And we can apply other aspects from horizontal alignments too. The above symbol has a variant if the dry cleaning must be done in a more "gentle" way. This is indicated by a horizontal rule below the circle: Ⓟ. We can realize this rule with a \noalign.

```
\def\gentlePCE{{% open a group for \ooalign
 \ooalign{%      the rows are known from \PCE
    \hidewidth\hbox{\DCfont\kern0.06em P}%
       \hidewidth\cr
    \raise0.1ex\hbox{\Orb}\cr
    \noalign{\vskip1pt \hrule}}}% add a rule
```

Using symbols from math fonts. Of course the cell entries can also contain inline math. But there is a new aspect to consider if the defined macros should work in all situations.

In [6], p. 444, the following macro is shown to print the symbol for acid-free paper.

```
\def\goodpaper{{%
  \ooalign{\hfil
    \raise.25ex\hbox{%
      $\scriptstyle\mathchar"231$}%
    \hfil\crcr
    \mathhexbox20D}}}
```

As \mathchar"231 stands for the symbol '∞' and \mathhexbox20D for the \Orb, i.e., '◯', the result is: ⊗. As stated in the article, the macro works well at the site of the author.

But the macro isn't as universal as it could be; it makes an assumption that a certain parameter has more or less its default value. TEX adds space around inline math as specified by the value of the dimension \mathsurround; [1], p. 162. The default value in plain is 0 pt. But at a site that changes this default value, say, setting \mathsurround=2pt, the above definition gives a distorted symbol: ⊗ .

The macros that define \ooalign assign special values to a lot of parameters and we use therefore a group when \ooalign is called. Thus, it's easy to reset inside this group one more parameter to avoid the described problem. Let's set \mathsurround to 0 pt if inline math is used in the \ooalign. (Note, \scriptspace should be set to 0 pt too if you use super- or subscripts in the math.)

All the above examples are made from two symbols but that isn't a fixed limit. For example, to express that dry cleaning with chemicals must be avoided, the symbol '⊗' was designed. This time three characters are involved as the cross symbol consists of the less-than and greater-than symbols.

```
\def\NOTchemical{{\mathsurround=0pt % init math
 \ooalign{\hidewidth % 1st row with two symbols
   \raise0.1ex\hbox{$>\mkern-3mu<$}%
   \hidewidth\cr          % 1st row is centered
  \raise0.1ex\hbox{%     same \raise as in \PCE
   \kern2pt\Orb\kern2pt}\cr}}% 2nd row: circle
```

Note that inside the math mode I work with \mkern instead of \kern.

Up to now all examples were constructed with two row entries. But of course, this isn't a limit either. Any number of rows can be used. Let's look at an example with inline math and three rows.

```
\def\frowny{{% a group is required for \ooalign
 \mathsurround=0pt %  required with inline math
 \ooalign{\hidewidth \raise.3ex\hbox{% the eyes
    $\cdot\mkern 2mu\cdot$}\hidewidth\cr
  \hidewidth \lower.2ex\hbox{%        the mouth
    $\scriptscriptstyle \frown$}\hidewidth\cr
  \Orb\cr}}}%                          the head
```

The above code typesets this symbol: ☹. It shouldn't be difficult to create the symbol '☺' if you prefer this one.

4 Complex examples

The symbols that we defined in the previous section are designed to be entered in horizontal mode, i.e., inside a paragraph. But \ooalign can also be used to create symbols that occur in math mode. We need to be more careful, as such a symbol must be allowed to appear in a displayed equation, in inline math inside a paragraph, or as a super- or subscript to another symbol. In other words we must code the symbol in a way that TEX scales its size for the different use cases.

The Computer Modern math fonts provide the symbol '∈' through the control word \in. It represents a relation between two entities like the equal sign. Mathematicians write this relation to express the phrase "is a member of". For example, it is used to express that a number is a member of a set of numbers: $8 \in \{0, 2, 4, 6, 8\}$. To state that a number is not in this set the symbol '∈' is crossed out: $7 \notin \{0, 2, 4, 6, 8\}$.

No Computer Modern math font contains this crossed-out symbol. Plain TEX names it "\notin"; a macro builds it from a slash, i.e., '/', and the symbol '∈'. We look at this macro in a moment but first let's see how we would define this macro. I name it \NOTin as I don't want to replace the macro of plain.tex.

```
\def\crossedoutin{% print '/' over \in in math
 {\mathsurround=0pt %  a group, no \mathsurround
  \ooalign{$\hfil\mkern 1mu/\hfil$\crcr% the '/'
    $\in$\crcr}}%    close group and end macro
\let\NOTin=\crossedoutin
```

The indirect approach to define \NOTin will become clear in a moment.

First, let's perform a test of the new symbol. In the following I entered \NOTin every time TEX typesets ∉: "$8 \in \{0, 2, 4, 6, 8\}$ and $7 \notin \{0, 2, 4, 6, 8\}$ and (a little bit silly) as a subscript

$$X_{8\in\{0,2,4,6,8\}} \quad \text{and} \quad X_{7\notin\{0,2,4,6,8\}}"$$

It is obvious that (a) the spacing for the new symbol and (b) the size in the subscript aren't good. Look closely and you see that the axis of the new symbol in the subscript isn't perfect.

TEX doesn't know that the new symbol is a relation. We must inform TEX about this fact. The primitive \mathrel assigns the new symbol to the class that represents relations ([1], p. 155). This fixes problem (a):

```
\def\NOTin{\mathrel\crossedoutin}
```

"$8 \in \{0, 2, 4, 6, 8\}$ and $7 \notin \{0, 2, 4, 6, 8\}$ and (a little bit silly) as a subscript

$$X_{8\in\{0,2,4,6,8\}} \quad \text{and} \quad X_{7\notin\{0,2,4,6,8\}}"$$

Mathematics: classes and styles. TEX assigns to every math character one of eight *classes*. "Relation" is one of these classes. The others are: ordinary, large operator, binary operation, opening, closing, punctuation, and variable family; see [1], p. 154. Any symbol can be assigned to one of these classes (except the last): Precede the symbol by one of these primitives \mathrel, \mathord, \mathop, \mathbin, \mathopen, \mathclose, or \mathpunct, respectively. The class of an object determines how TEX treats it in certain situations. For example, TEX looks at the class to determine the space around the object.

In math mode TEX knows different styles. We all know that there is a difference between inline math in a paragraph — started and ended with a single $ — and the display math mode that's started and ended with two dollar signs. TEX knows four styles when it typesets mathematics ([1], p. 141): (a) *display style* for material in display math mode, (b) *text style* for inline math, (c) *script style* for super- and subscripts in display or text style, and (d) *scriptscript style* for super- and subscripts of super- or subscripts. These four styles have also a *cramped* version, so there are eight styles in all.

TEX decides when to use the cramped versions so we don't need to be worried about them. But the original styles must be addressed if we want to create a symbol that TEX can use in all situations. TEX provides us with primitives that select one of the four styles: \displaystyle, \textstyle, \scriptstyle, and \scriptscriptstyle. The last two were already used in the code for \goodpaper and \frowny.

We don't have to use these primitives directly to code four different symbols. The plain format

helps us as it provides the macro \mathpalette ([1], p. 151). It offers TEX the symbol in the four styles.

The macro receives two parameters. The first is the name of the symbol that should be processed. \mathpalette requires that the macro for the symbol accepts at least one argument. This argument is supplied by \mathpalette and is the primitive that switches to one of the four styles available in math modes. For example, the argument might be \displaystyle.

The second argument of \mathpalette adds some flexibility to the macro of the new symbol: The argument is passed on to the macro of the symbol, but has no special meaning for \mathpalette.

Combine \mathpalette and \ooalign. To apply \mathpalette in \NOTin we must change the macro \crossedoutin as it now gets an argument. In our example there is no need for an additional argument to \crossedoutin, so we will use {} in the call of \NOTin. But, please, don't be confused: This empty group isn't the first argument to \crossedoutin, it's an argument of \mathpalette. In a second step, \mathpalette takes this empty group and offers it to the macro found as its first argument if this macro asks for two arguments.

```
\def\crossedoutin#1{% #1: a math style
 {\mathsurround=0pt %  a group, no \mathsurround
  \ooalign{% build two rows and use the style #1
     $#1\hfil\mkern 1mu/\hfil$\crcr    % the '/'
     $#1\in$\crcr}}%  close group and end macro
\def\NOTin{% applicable in all math styles
 \mathrel{\mathpalette\crossedoutin{}}}
```

This code fixes problem (b). Here is our test input: "$8 \in \{0,2,4,6,8\}$ and $7 \notin \{0,2,4,6,8\}$ and (a little bit silly) as a subscript

$$X_{8\in\{0,2,4,6,8\}} \quad \text{and} \quad X_{7\notin\{0,2,4,6,8\}}"$$

We no longer need a group for \ooalign (and \mathsurround) as the construction is later placed in a group for \mathrel. But it doesn't hurt to have this group.

It's time now to look at plain.tex again and see how Knuth implemented the macro \notin. The macro \m@th sets \mathsurround to 0 pt.

```
\def\m@th{\mathsurround\z@}
\def\c@ncel#1#2{\m@th
 \ooalign{$\hfil#1\mkern1mu/\hfil$\crcr$#1#2$}}
\def\notin{\mathrel{\mathpalette\c@ncel\in}}
```

We find a more general macro, \c@ncel, that prints the slash over its second argument. Thus it can be used in more constructions than just for the symbol \in. Which symbol gets canceled is the second argument of \mathpalette. Otherwise the construction is very similar to our solution.

Create two new binary operators. As the last example in this section, let's consider creating two new binary operators: ⊠ and ⊡, named with control words \bast and \bstar, respectively. As we need more than one symbol, we apply the technique that was used in the macro \notin with a second argument for \mathpalette.

The symbols \ast (∗) and \star (⋆) can be used for the inner symbol. For the square, let's check the font tables in Appendix F of [1]. But none is there. Either we now check font tables of other fonts, for example, the AMS symbol font, or we can construct a square: Use a combination of \sqcap and \sqcup.

```
\def\bbox#1#2{% #1: a math style;
% #2: a symbol to be placed inside a square
 \mathsurround=0pt % initialize for inline math
 \ooalign{% with three rows; two for the square
     $#1\sqcap$\cr  % three sides of the square
     $#1\sqcup$\cr        % and its fourth side
   \noalign{\vskip-0.109ex}
     % center the symbol and cancel its width
     $#1\hidewidth#2\hidewidth$\cr}}
\def\bast{\mathbin{\mathpalette\bbox\ast}}
\def\bstar{\mathbin{\mathpalette\bbox\star}}
```

Now we can type $x \boxtimes y$ and $X \boxtimes Y$ as well as $Z_{x\boxtimes y}$. More symbols of this kind are possible; for example, \bdot and \bcirc: $x \boxdot y \neq x \circledcirc y$.

```
\def\bdot{\mathbin{\mathpalette\bbox\cdot}}
\def\bcirc{\mathbin{\mathpalette\bbox\circ}}
```

Note, in the definition of \bbox the extra group for \mathsurround is omitted as we have a group for \mathbin. Moreover, \cr is used instead of \crcr; it doesn't make a difference.

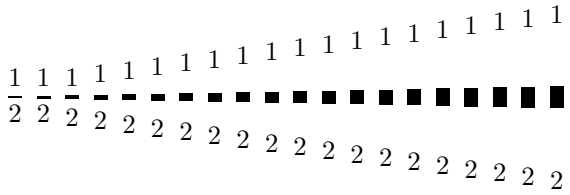
5 An experiment

I presume that every reader will agree that TEX is a complex system. And in Knuth's words ([3], p. 661): "Any complex system can be improved; therefore the goal of absolute perfection and optimality is unattainable." Along with the errors that he has made and fixed during the lifetime of TEX, he also lists some aspects of TEX that can be improved; some of them he calls "design flaws" in [3], p. 660.

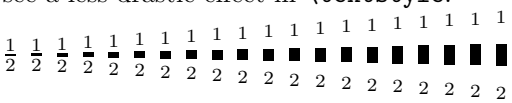
Here is the text of the first design flaw from [3]: "Additional parameters in symbol fonts could govern the minimum distance between ruled lines in fractions, \sqrt, \overline, and \underline; at present this minimum distance depends only on the thickness of the line." (The first statement under the heading "Design errors that are too late to fix" in [5] is very similar.)

What does that mean? Let's look at the quotient $1\above x pt 2$ with x going from 0.4 to 8

in steps of 0.4. In `\displaystyle` we get:



We see a less drastic effect in `\textstyle`:



The analog sequences created using `\scriptstyle` and `\scriptscriptstyle` are quite similar to the sequence for `\textstyle`.

Quotients up to $x = 1.6$ seem to look acceptable, so a writer of a textbook on elementary mathematics can choose from clearly distinct line thicknesses ([1], p. 143). Nevertheless, the situation is disappointing. Can the macro `\ooalign` help?

This is the idea: We can set the fraction with a small size for the line thickness and then we overprint this thin line with a line of the desired thickness. We assume that (a) the fraction is enclosed by `\lA` and `\rA` instead of braces and (b) `\mA` is put after the line thickness. Macro `\lA` splits the input and uses the macro `\mathpalette`, whose second argument collects the input. But this input must be split again to be usable for the main macro `\Aalign`.

```
\dimendef\Adim=0 % the new line thickness, local
\def\lA#1\above#2\mA#3\rA{% read fraction
   \mathord{\mathpalette\Aargs{{#1}{#2}{#3}}}}
\def\Aargs#1#2{\Aalign#1#2\end}% #1: math style
\def\Aalign#1#2#3#4\end{% #1: math style;
% #2: numerator; #3: dimen<=8pt; #4: denominator
 % guess an acceptable value for \above's dimen
 % based on the integer part and the 1st decimal
 % of the given dimen (compared to unit ``pt'')
{\Adim=\ifx#1\displaystyle
   \ifdim #3<1.5pt #3\relax
   \else\ifdim #3<2.5pt 1.5pt
   \else\ifdim #3<5pt 1.75pt
   \else 2pt \fi\fi\fi
 \else % the other styles need more cases
   \ifdim #3<1.5pt #3\relax
   \else\ifdim #3<2pt 1.5pt
   \else\ifdim #3<3pt 1.75pt
   \else\ifdim #3<4pt 2pt
   \else\ifdim #3<5pt 2.25pt
   \else\ifdim #3<6pt 2.5pt
   \else\ifdim #3<7pt 3pt
   \else 3.5pt \fi\fi\fi\fi\fi\fi\fi   \fi%\ifx
 \ooalign{% build fraction thrice: 1) output
   % fraction, 2) overprint line, 3) fix depth
   $#1{#2\above\Adim #4}$\cr  % use a thin line
   $#1{\phantom{#2}\above#3\phantom{#4}}$\cr
   $#1{\phantom{#2}\above\Adim\phantom{#4}}$}}}
```

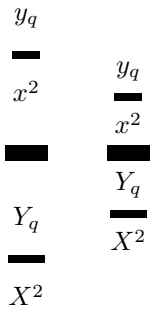
Next, let's look at the two figures from above using `\Aalign`. This time I entered the quotients as `\lA 1 \above 0.4pt\mA 2\rA`, etc.

Here is the sequence in `\textstyle`:

And here is another comparison between TEX's default output and the one from `\Aalign`. The code

```
$${\displaystyle{y_q \above1mm x^2}\above2mm
   \displaystyle{Y_q \above1mm X^2}}    \qquad
\lA{\displaystyle\lA y_q \above1mm\mA x^2\rA}
   \above2mm\mA
   {\displaystyle\lA Y_q \above1mm\mA X^2\rA}
\rA$$
```

outputs



Clearly, there is an improvement. Nevertheless, the values for `\Adim` should be studied in more depth if you want to use this trick. The excessive use of `\mathpalette` — the `\phantom` macro ([1], p. 360) applies it too — makes it an expensive procedure.

References

[1] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[2] Donald E. Knuth, *Computer Modern Typefaces*, Volume E of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[3] Donald E. Knuth, "The Final Errors of TEX", Chapter 34 of [4], 655–662, written August 1998.

[4] Donald E. Knuth, *Digital Typography*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 78, 1999.

[5] Donald E. Knuth, "`tex82.bug`", file in the distributed errata for *Computers & Typesetting*. `ctan.org/tex-archive/systems/knuth/dist/errata/tex82.bug`

[6] Pierre A. MacKay, "Recycled METAFONT", *TUGboat* **15**:4 (1994), 444–446. `tug.org/TUGboat/tb15-4/tb45mack.pdf`

◇ Udo Wermuth
Dietzenbach, Germany
`u dot wermuth (at) icloud dot com`