Providing color in math*

Frank Mittelbach

June 8, 2022

1 Introduction

It is possible to use the `\color` command in math formulas but doing that has a number of restrictions and often leads to rather unreadable input. For example, to color the summation sign in red in the following formula

$$X = \sum_{i=1}^{n} x_i$$

A total of four color commands and a number of seemingly unnecessary extra braces in the sub and superscript are needed:

```
\[ X = \color{red} \sum
       _{{\color{black} i=1}}  % without {{ the superscript is misplaced
       ^{{\color{black} n}}    % without {{ the \sum is black
      \color{black}            % without the x_i is red
      x_i                   \]
```

While this is ugly but at least works, other things are simply impossible, e.g.,

$$\left\{ \frac{1}{2} \right\}$$

are not achievable at all because of the fact that `\left` and `\right` form a group. Thus, a `\color` command immediately before the `\right` has no effect and `\}` remains black. Putting the color change before the `\left` and then resetting the color inside would work if you want to color both braces, but the above combination or the attempt to color only the left brace fails always.

Using `\textcolor` in formulas appears to work on first glance, but it produces spacing problems that are not easy to overcome either, because it generates a single `\mathord` and no longer reflects its input, thus `\[ a = b \textcolor{red}{\neq} c \]` produces

$$a = b \textcolor{red}{\neq} c$$

This is a case one could mend by wrapping the `\textcolor` and its arguments in a `\mathrel`, but even when that is possible, the resulting formula source is difficult to maintain and in other situations the solutions are even more complicated or there is no solution at all.

*This file has version v1.0b dated 2022/01/28, © LaTeX Project.

We therefore offer now a dedicated math coloring command named `\mathcolor`.

`\mathcolor` `\mathcolor [⟨model⟩] {⟨color-spec⟩} {⟨math material⟩}`

It has the same arguments as `\textcolor` but is intended for use in formulas. The command does not generate a group and the ⟨*math material*⟩ retains its math atom states and it correctly handles sub and superscripts that follow.

The command can also be used to color a single opening or closing symbol, e.g., the correct input to our earlier example is

```
\[  \left\{ \frac{1}{2} \mathcolor{red}{\right\}} \]
```

which is how it was produced.

If you attempt to color large operators that use explicit `\limits`, `\nolimits`, or `\displaylimits`, then these limit controls need to immediately follow the operator. However, `\mathcolor` is written in a way that it is possible write

```
\[  \mathcolor{red}{\int}\limits_0^1 \textrm{ or }
    \mathcolor{red}{\int\limits}_0^1                \]
```

and achieve the same effect:

$$\mathcolor{red}{\int}\limits_0^1 \text{ or } \mathcolor{red}{\int\limits}_0^1$$

2 The Implementation

The code is called inside of the color or xcolor packages, so `@` is already a letter, but the coding here is done in the L3 programming layer, so we need to activate that. But first we check if this file was loaded before, e.g., when both color and xcolor are in the preamble and in that case we stop immediately.

```
⟨*code⟩
\ifcsname mathcolor\endcsname \endinput \fi
\ExplSyntaxOn
⟨@@=mathcolor⟩
```

`\g__mathcolor_seq` We need to keep our own stack of color commands, so we allocate a sequence for that.

```
\seq_new:N \g__mathcolor_seq
```

(*End definition for* `\g__mathcolor_seq`.)

`\mathcolor` The document-level command which expects a color (if unnamed then the optional argument is the model) and colors the content of the second argument (like `\textcolor`, but without spacing problems in math).

```
\DeclareDocumentCommand \mathcolor { o m m } {
```

The `\mathcolor` is only supported in math mode because in text mode it has problems scanning away a space after it, for example. We therefore raise an error if it executes anywhere else. The LaTeX 2_ε error command is a bit strangely named, because in the kernel it is only used for math alphabets, but the message it gives is fine.

```
  \mode_if_math:F { \non@alpherr {\mathcolor\space} }
```

First real action is to save the current color value on a stack (needed if the command is nested or contains some further color changes with `\color` inside).

```
\seq_gpush:No \g__mathcolor_seq \current@color
```

Then we switch to the new color, but we do not want to reset the color after the group (which is done by `\color` using `\aftergroup\reset@color`). The best solution here would be if the color packages would provide a command doing just the color switching, but for now we simply undo that part by pushing `\use_none:n` which gobbles the `\reset@color` added by `\color` with `\aftergroup`.

```
\group_insert_after:N \use_none:n
```

Switching the color is also slightly suboptimal, because depending on whether or not we have a ⟨*model*⟩ argument, we have to call `\color` with or without the optional argument. But going low-level here is not an option as we need to support different color packages and their internals are not identical.

```
\IfValueTF{#1} { \color[#1]{#2} }{ \color{#2} }
```

Then comes the math material we want to see colored:

```
#3
```

After that we need to reset the color ourselves (without a group that does it for us), i.e., popping the saved color from our stack, but there are some twists to that so we do this in a separate command (which in fact needs to be called several times, so inlining the code wouldn't be possible.

```
  \__mathcolor_scan_for_scripts:w
}
```

(*End definition for* `\mathcolor`. *This function is documented on page* 2.)

`\__mathcolor_scan_for_scripts:w`

The complication when changing the color back is due to the fact that the `\mathcolor` may be followed by `^` or `_` or the hidden superscript `'` and its argument may end in a `\mathop` in which case the sub and superscripts may be attached as `\limits` instead of after the material. All cases need separate treatment.

```
\cs_new_protected:Npn \__mathcolor_scan_for_scripts:w {
```

We therefore first parse for a `\c_math_subscript_token` ignoring (dropping) any spaces and `\relax` as TEX would do. We do this with expansion so that hidden sub or superscripts in macros are still found as long as the macros are expandable.

```
  \__mathcolor_peek_catcode_ignore_filler_expand:NTF \c_math_subscript_token
```

If we found that character we call a function that handles sub or superscripts.

```
    { \__mathcolor_handle_scripts:Nw }
```

Otherwise check if this token we peeked at (which is now stored in `\l_peek_token` is a superscript token.

```
    { \token_if_math_superscript:NTF \l_peek_token
```

If found we call the sub/superscript handler.

```
      { \__mathcolor_handle_scripts:Nw }
```

Otherwise we check if it was any of the limit operation primitives. If that is the case, e.g., if we have a situation such as

```
\mathcolor{red}{\int}\limits_1
```

we have to move it directly after the `\int` to ensure there is no color reset between the operator and the `\limits` command.

```
      { \token_case_meaning:NnTF \l_peek_token
          {
            \limits   { \limits }
            \nolimits { \nolimits }
            \displaylimits { \displaylimits }
          }
```

Once that is done, we have to get rid of the token we peeked at and then restart scanning for sub or superscripts. Given that `\__mathcolor_scan_for_scripts:w` expands while scanning the simplest solution is to add `\use_none:n` in front of the peeked at token.

```
          { \__mathcolor_scan_for_scripts:w \use_none:n }
```

If it was one of these we look for a `'` and if found remove it and replace it by its expansion. The reason we have to do this (and not rely on the earlier peeking to expand for us is the fact that `'` is only "math active" and that doesn't expand under `\expanded` or `\expandafter`.

```
          { \peek_meaning_remove:NTF '
            { \__mathcolor_handle_scripts:Nw ^\c_group_begin_token \prim@s }
```

If it is anything else we finish off which means we reset the color (because we prevented that before to happen automatically after the next group) and pop the color stack setting `\current@color`.

```
            { \reset@color
              \seq_gpop:NN \g__mathcolor_seq \current@color
            }
          }
      }
    }
}
```

(*End definition for* `\__mathcolor_scan_for_scripts:w`.)

`\__mathcolor_handle_scripts:Nw` The tricky part of handling sub and superscripts is that we have to reset color to the one that is on the stack but reset it back to what it was before to allow for cases like

```
\[  \mathcolor{red}{a+\sum}_{i=1}^n  \]
```

Here, TEX constructs a `\vbox` stacking subscript, summation sign, and superscript. So technically the superscript comes first and the `\sum` that should get colored red is the middle.

```
\cs_new_protected:Npn \__mathcolor_handle_scripts:Nw #1 {
```

The argument is either `^` or `_`, so we execute it and explicitly open two `{` groups. We need two because color resets are always done after a group, so the first group is for script material (in case it was just something like `_i`) and the second is needed for the color reset to keep it within the super or subscript. If that reset would happen after it then the color `\special` would interfere with TEX math spacing.

```
    #1 \c_group_begin_token \c_group_begin_token
```

Now it is time to change the color back to the one on the stack.

```
    \seq_get:NN \g__mathcolor_seq \current@color
    \set@color
```

\set@color adds \aftergroup\reset@color. We now a add bit more so that the code executed after the current (inner) group looks like this:

```
\reset@color } \@@_scan_for_scripts:w
```

The __mathcolor_scan_for_scripts:w then retakes control and initiates parsing for another sub or superscript.

```
    \group_insert_after:N \c_group_end_token
    \group_insert_after:N \__mathcolor_scan_for_scripts:w
```

Before we give control to TEX to process the sub or superscript some final adjustment is necessary: if the input was ^{...} then we have one { too many, because we already supplied the outer one already. In that case we drop it. Otherwise we have an unbraced single token sub or superscript which means we are missing a closing } at the end and need to account for that: this is done in __mathcolor_handle_unbraced_script:N.

```
    \__mathcolor_peek_catcode_ignore_filler_expand:NTF \c_group_begin_token
      { \peek_catcode_remove:NT \c_group_begin_token { } }
      { \__mathcolor_handle_unbraced_script:N }
  }
```

(*End definition for* __mathcolor_handle_scripts:Nw.)

__mathcolor_handle_unbraced_script:N

All we have to do here is to add the final closing brace.

```
\cs_new_protected:Npn \__mathcolor_handle_unbraced_script:N #1 {
  #1 \c_group_end_token }
```

(*End definition for* __mathcolor_handle_unbraced_script:N.)

2.1 Utility function for the peek module

This peek is expanding and ignores spaces and \relax (even though the name doesn't say so) but \relax is usually allowed in such places where we so it may not be bad to have it in this way in general — maybe this should move to expl3 peek functions (or otherwise the naming should perhaps be adjusted).

```
\tl_new:N \l__mathcolor_peek_tmp_tl
\cs_new_protected:Npn \__mathcolor_peek_catcode_ignore_filler_expand:NTF #1#2#3
  {
    \tl_set:Nn \l__mathcolor_peek_tmp_tl
      { \token_if_eq_catcode:NNTF \l_peek_token #1 {#2} {#3} }
    \__mathcolor_peek_expand:w
  }
\cs_new_protected:Npn \__mathcolor_peek_expand:w
  {
    \exp_after:wN \peek_catcode_remove:NTF
    \exp_after:wN \c_space_token
    \exp_after:wN \__mathcolor_peek_test_expand:w
    \exp_after:wN \__mathcolor_peek_test_relax:w
    \exp:w \exp_end_continue_f:w
  }
```

```
\cs_new_protected:Npn \__mathcolor_peek_test_expand:w
  {
    \token_if_expandable:NTF \l_peek_token
      { \__mathcolor_peek_expand:w }
      { \l__mathcolor_peek_tmp_tl      }
  }
\cs_new_protected:Npn \__mathcolor_peek_test_relax:w
   {
    \peek_meaning_remove:NTF \scan_stop:
      { \__mathcolor_peek_expand:w }
      { \__mathcolor_peek_test_expand:w   }
   }
⟨@@=⟩
\ExplSyntaxOff
⟨/code⟩
```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.