1 Characters in math

This chapter goes into some more details about math characters but after some remarks goes on about about discretionaries. Traditionally TEX users enter ascii characters mixed with commands and expect to get the right visual representation. Because there are plenty math characters outside the ascii range this means that most are accessed by a command. However, Unicode changes that: when an editor can show the character there is no reason not to use that feature. In that case we end up with utf characters in the input. In this perspective is it important to realize that there is a distinction between such a direct utf character and a command, especially when it is defined as follows:

```
\Umathchardef\mathcharacterf 0 \mathordinarycode `f

\startformula
    f = \mathcharacterf = \Uchar"1D453
\stopformula
```

This gives the expected:

$$f = f = f$$

The ascii `f` will eventually become character `U+1D453` but let's not worry here about how that is done; what is more important is that this character has some extra properties. Just like the definition of the command we use a primitive `\Umathcode` that registers that we have an ordinary and that the origin is family zero, we also need to make sure that the `U+1D453` has those properties. The way it works is that the engine injects a math noad with a math character nucleus and when it does that it needs to resolve the family and the class. Depending on the style the family will resolve in a text, script or scriptscript font. The class determines the spacing and some specific engine behavior.

In ConTEXt and therefore in LuaMetaTEX we go as step further. There we also have dictionary fields, which makes it possible to adapt properties like the class as we like after the user has entered them. This (experimental) features relates to the fact that often Unicode math, TEX character names, and usage doesn't really reveal what the character is about and if it is needs to have class binary, relation or something else. If the command does carry some meaning it gets lost when we end up with these injected math characters. In LuaMetaTEX we do carry more around. Because this is experimental and evolving we stick to mentioning that there is for instance a primitive `\Umathdictdef` that does what `\Umathchardef` does but expects three additional numbers: properties, group and index.

In LuaMetaTEX we try to be as detailed as possible when we resolve and store references to characters in the math nodes (noads), even if the engine itself doesn't always need that information, for instance: for handling a single character superscript we don't need to know its class.

This detour was needed in order to understand the following: discretionaries in math mode. In LuaMetaTEX we are already more tolerant with respect to what can end up in a discretionary and math discretionaries have been supported for a while now. In the next examples, class 2 is used: binary.

```
test $ \dorecurse{50}{a \discretionary class 2 {$+$}{$+$}{$+$} } b$ test
```

test $a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+$
$+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+a+b$
test

But this is not nice: we need to enter math mode in the three snippets and likely also need to make sure that we do that in the right style. So, that was why we can now also do this:

```
test $ \dorecurse{50}{a \Umathdiscretionary class 2 {<}{>}{=} } b$ test
```

test $a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a<$
$>a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=a=b$
test

We can wrap this in a command:

```
\def\weirdrelation{\Umathdiscretionary class 2 {<}{>}{=}}
```

but this is not what we want when we are talking + and - which are candidates for repetition. And these are entered as utf character so there is indication of them being treated special. This is why LuaMetaTEX has a new vector \hmcode where one can trigger specific characters to become discretionaries.

```
\hmcode"002B=1 % +
\hmcode"2212=1 % -
```

```
test $ \dorecurse{50}{a + b - } c$ test
```

test $a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+$
$+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-$
$-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+$
$+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-a+b-c$
test

Setting bit one of the code will enable this feature. But as usual with TEX and math there is a pitfall. Take this (unusual) example:

```
\hmcode"1D453=1 % we trigger promotion to discretionary
```

```
test $\dorecurse{50}{a \Umathchar 2 0 "1D453 b} b$ test
```

test $a f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f$
$f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f$
$f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f bb$ test

We see the f being repeated but also notice that the italic correction disappears because that is what happens in the line break. But in math this correction is actually part of the width (we've written plenty about that over the years). However, when we set bit two of the code, the correction is moved into the discretionary:

```
\hmcode"1D453=3 % we carry the italic correction along
```

```
test $\dorecurse{50}{a \Umathchar 2 0 "1D453 b} b$ test
```

test $a f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f$
$f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f$
$f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f ba f bb$ test

So, where characters need to retain their family and class, we also need to make sure that we retain the fact that a character is to be automatically repeated at a line break. The reason why this ended up in the engine while it could be delegated to a callback is that we do need to process discretionaries in math anyway and also want to avoid it when we're not at the outer level. And because we already carry around all kind of options with noads and glyphs it was not that hard to support this.

It is a bit of a side track but discretionaries in LuaMetaTEX are a bit more permissive anyway. Take this:

```
\dorecurse{20}{%
    xxxxxx
    \discretionary {>>} {<<} {==}
    xxxxxxxxxx
}
```

xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx >> << xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx >> << xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx

Here we depend on the tolerance and stretch settings in order to not overflow the text boundaries. But how about the next:

```
\dorecurse{20}{%
    xxxxxx
    \discretionary
      {>\hskip0pt plus 5pt>}
      {<\hskip0pt plus 5pt<}
      {=\hskip0pt plus 5pt=}
    xxxxxxxxxx
}
```

xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx > > < < xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx > > < < xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx = = xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx

This time we have some glue in the snippets. But we can even do the next trickery, where we can stretch the boxed content after the line break routine has done it work. It is this mechanism that we use deep down in the math engine too.

```
\dorecurse{20}{%
    xxxxxx
    \discretionary
        {\uleaders \hbox to 2em{>\hss>}\hskip0pt plus 10pt minus 5pt}
        {\uleaders \hbox to 2em{<\hss<}\hskip0pt plus 10pt minus 5pt}
     %  {\uleaders \hbox to 2em{=\hss=}\hskip0pt plus 10pt minus 5pt}
        {==}
  % xxxxxxx
```

```
    xxxxxxxxxx
}
```

xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx > > < < xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx > > < < xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx xxxxxx == xxxxxxxxxx

So, in some way, extending the math engine lets features trickle back into the text engine and vise versa. It is all about seeing (weird) opportunities because it is often after playing with this that one sees more potential.