CurExt
Typesetting variable-sized curved symbols

Azzeddine Lazrek
Department of Computer Science, Faculty of Science,
University Cadi Ayyad, P.O. Box 2390, Marrakech, Morocco
Phone: +212 44 43 46 49 Fax: +212 44 43 74 07
`lazrek@ucam.ac.ma`
`http://www.ucam.ac.ma/fssm/rydarab`

Abstract

The main goal of this contribution is to present a program that allows the composition of variable-sized curved symbols such as those occurring in mathematics. This application, called **CurExt**, extends the capabilities of the well-known system TEX designed by D. E. Knuth for typesetting. Big delimiters, such as brackets, or special curved symbols, such as the Arabic mathematical symbol of sum, can be built automatically according to the size and the shape of the concerned mathematical expression. **CurExt** will make it possible to stretch Arabic letters according to calligraphic rules in order to draw the kashida. It follows a useful tool for justifying texts written with Arabic alphabet. Unlike in Latin alphabet based writing, where the justification is done through inserting small blanks among characters, cursive writing fills in the space between characters with the kashida.

The problem

Variable-sized symbols Besides *fixed size* symbols, such as characters of Latin alphabet in a given font or basic mathematical symbols (e.g., +, −), there are symbols with a context dependant size. Some mathematical symbols, such as delimiters or Arabic characters, are examples of these *variable-sized* symbols. The variation of size can concern:

- the width of the symbol, such as in:

  – the various parts of some symbols (e.g., $\underline{\sum}$ or $\underline{\overline{\langle}}$, $\Rightarrow$). A horizontal stretching, according to the expression covered by the operator, is sometimes necessary;

  – the kashida for some characters of the Arabic alphabet (e.g., بـ , جـ) or certain Arabic mathematical symbols such as those found in the abbreviations of usual functions (e.g., جـــــ , جـــتا , نهـــا) in mathematical expressions. The kashida ـــ, a small curve, is used to stretch some characters in order to cover the concerned mathematical expression or to break the line when the left margin is reached;

  – some diacritics or accents (e.g., $\underline{abc}$, $\overline{abc}$, $\widehat{abc}$, $\widetilde{abc}$, $\overleftarrow{abc}$, $\overrightarrow{abc}$, $\overbrace{abc}$, $\underbrace{abc}$ or $\underbrace{\text{ابح}}$ ، $\overbrace{\text{ابح}}$ ، $\overrightarrow{\text{ابح}}$ ، $\overleftarrow{\text{ابح}}$ ، $\widetilde{\text{ابح}}$ ، $\widehat{\text{ابح}}$ ، $\overline{\text{ابح}}$ ، $\underline{\text{ابح}}$).

- the length of the symbol, such as in:

  – delimiters (e.g., $\langle$, (, $|$, [, $\|$, \{, \}, $\|$,], $|$,), $\rangle$);

  – symbols of operators (e.g., $\int$ or ⎫, ⎫, $\Uparrow$).

- the width and the length of the symbol, such as in some mathematical symbols: (e.g., $\sqrt{}$ or $\overline{}\!\!\diagdown\!\!\surd$).

Production of the variable-sized symbols Various approaches can be adopted to produce variable-sized symbols:

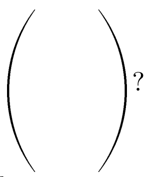
- *Through measuring*: glyphs are then made on the basis of measurements directly taken from the context of the symbol. This manner leads to a very high precision but a *posteriori*. A second processing of the text is absolutely necessary, after a first one where measurements of sizes are taken and recorded. The glyphs can be produced once the measurements made. This way of proceeding makes it possible to have one glyph of sign per size through dynamic fonts.

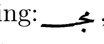
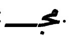
- *Ready to wear*: standardized sizes are determined. Glyphs are then drawn according to this set of sizes. No second processing will be necessary of course. The precision cannot reach the level attained through the previous manner. A glyph of sign per interval of sizes, through static fonts, is then produced *a priori*.
- *Semi-finished*: a combination of the two previous ways.

There is a difference between producing these glyphs and using them. The production can be done through programs, with parameters to be determined, with `METAFONT` or `PostScript`. A particular system of edition will be necessary to make use of the glyphs. This system will be used to send the required parameters to the previous program for generating the fonts.

Curvilinearity of the variable-sized symbols Besides symbols drawn with *segments* (e.g., [,], $\Rightarrow$), there are variable-sized symbols composed with small *curves* (e.g., (,), $\int$). Thus, the extensibility of the symbol should be done in the respect of this curvilinearity. The task becomes difficult for the shapes of the curves that vary according to the size. Then, the problem is not limited to stretching or lengthening these curves. It consists of producing curves according to different sizes. As far as we know, up till now, there is no system that allows the production of variable-sized curved symbols through parameterized dynamic fonts. Say, for example, curvilinear variable-sized parenthesis. Neither the system TEX [4] nor `MathType`[1] offers such possibility. An attempt had been made with `Math-Fly`[3] font [1] over the system `Grif/Thot`[4] [9]. It didn't go far from the step of experimentation nor has it been added to the system. The system Ω[6] doesn't offer the possibility of producing such symbols. An extension to `ditroff/ffortid`[7] allows the abilities to stretch letters themselves with dynamic `PostScript` fonts [2].

A very detailed survey on the ways for producing variable-sized symbols and the general problem of the optical scaling can be found in [1].

The parentheses in big sizes, say, for example, those of matrices offered by the system TEX, are such that: $\left(\rule{0pt}{5em}\qquad\right)$. Can we get curved parentheses such as: ?

In the same way, one can wonder how to produce kashida so as to get the correct writing: , instead of the simple straight-line lengthening: .

Variation of symbol's size in TEX

The system TEX handles the problem of producing variable-sized symbols through two different ways at the same time, with the `Computer Modern` font developed by D. E. Knuth in `METAFONT` [5]:

- through the production, a priori, of glyphs up to some sizes[8] (e.g., $\Bigg(\bigg(\Big(\big((\quad)\big)\Big)\bigg)\Bigg)$), using the primitive **charlist** from `METAFONT`;
- through composition, starting from small parts, whenever the size goes beyond some level (e.g., $\left(\rule{0pt}{5em}\qquad\right)$), using the primitive **extensible** from `METAFONT`. Horizontal or vertical segments are then added to get the desired size (e.g., $\begin{matrix}\underbrace{\qquad}\\\underbrace{\qquad}\\\underbrace{\qquad}\\\underbrace{\qquad\;\;}\\\underbrace{\qquad\quad}\end{matrix}$ or $\Bigg\{\bigg\{\Big\{\big\{\{()$.

When a certain limit of size is reached, some symbols can't be extended any more[8] (e.g., $\widehat{a}, \widehat{ab}, \widehat{abc}, \widehat{abcd}, \widehat{abcde}$).

[1] `MathType` is an equations processing system, including `Equation Editor`, from Design Science, Inc.[2]

[2] `http://www.mathtype.com` or `http://www.dessci.com`

[3] `Math-Fly` is a parameterized `PostScript type 3` font.

[4] `Thot` is an interactive system for the production of structured document. `Thot` is an evolution of the `Grif` system developed by the Opera team with the INRIA and the IMAG. `Amaya`, the navigator of W3C[5], is based on this system.

[5] `http://www.w3.org`

[6] `http://omega.cse.unsw.edu.au`

[7] `ditroff/ffortid` is a system for formatting bidirectional text in Arabic, Hebrew and Persian developed by J. Srouji and D. M. Berry [10].

[8] Some other sizes can be obtained by `yhmath` package, that extended math's fonts for LATEX, developed by Y. Haralambous `CTAN:/macros/latex/contrib/supported/yhmath`

The primitives `\Bigg`, `\bigg`, `\Big` and `\big` (e.g., `\Bigg( ... \Bigg)`) denote various possibilities of extension that the user can handle to get the desired size. The primitives `\left` and `\right` (e.g., `\left( ... \right)`) allow an automatic determination of the size according to the context.

The compiler TeX keeps room for each character. The character is considered as a rectangular box, with a width and length, on a baseline, passing between the height and the depth. These values are taken from the corresponding `TFM`'s files of the fonts in use.

The font should be determined a priori in the preamble of the document. No specification of size will be allowed while the document is processed. A possibility of *magnification*, for all text, is allowed a priori, in the preamble of the document. This magnification consists of a reduction or an enlargement of the font.

Curvilinear extensibility

Parameters of dynamic font Hereafter, a solution for obtaining variable-sized curved symbols is proposed. The particular case of parentheses, brackets of a mathematical expression, will be presented in detail, as an example of *vertical* extensibility. The kashida, as an example of *horizontal* extension, will be presented also. It goes without saying that all other variable-sized curved symbols can be handled in the same way.

The size of such symbols is determined a posteriori according to the context. Instead of taking the size of the symbol from the `TFM`'s files of the specified font, this size is computed starting from the size of the mathematical expression covered by the symbol. The required room is then reserved (using the TeX primitives `\hbox` and `\vbox` with `\wd`, `\ht` and `\dp`). Then, the program `METAFONT` is called for generating the fonts according to the taken sizes. Thus, dynamic fonts are built.

The necessary repetition of processing is not a problem, for the compiler TeX who should be called already, two times at least, for the table of the contents, the bibliography, the index, etc.

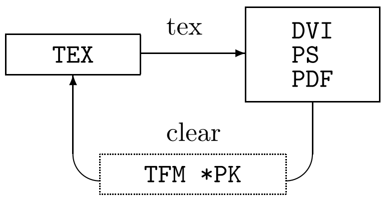
This way of processing will lead to the following constraints:

- there is a need for one font per character. The system TeX can use simultaneously up to 16 fonts. This can be a restriction of the number of variable-sized symbols that can be processed;
- for every variable-sized symbol, a file for storing the sizes computed after the first compilation is required. The number of files allowed by TeX for the input/output (by using the primitives `\read` and `\write`) is limited to 16. That also limits the number of variable-sized symbols to deal with;
- the program `METAFONT` allows up to 256 symbols per font processed by TeX. This seems to limit the number of different sizes of glyphs. This is not a true limitation because the width of a symbol is already limited by the width of the sheet and it is so for the length.

Actually, the number of required files can be reduced through the use of the same file for each pair of delimiters: the closing bracket is generally required wherever an opening one appears. Thus, the same file will be used for the two brackets. Another reduction of the number of required files may be carried out through recording a size only once for all the occurrences of a given symbol at a given size. TeX doesn't allow the use of a file in input and output. A file opened in output will be overwritten whenever it is called once again for input and conversely. This leads to the use of an intermediate temporary file to look for any possible existence of a given size. This file will be used for all variable-sized symbols handled.

As it was said before, the number of sizes of a variable-sized symbol is limited up to 256. Beyond this limit, the system will use the smallest size higher than the required one. The system holds the biggest size as default size. The intermediate file used previously will be used to determine this size of substitution. This will be done for all variable-sized symbols.

The size of an extensible symbol of an expression can be given if the expression itself does not comprise another extensible symbol. A problem arises when several extensible symbols are overlapping in the same expression. TeX and `METAFONT` should be called as many as variable-sized symbols overlap. The `TFM`'s and `*PK`'s files must be cleared in other to compute them every time with the news sizes as font's parameters, as might look like at following:



Parameters of glyph During the development of such as dynamic font of variable-sized symbols, some difficulties arise to determinate:

- the shape of the glyph according to the dimensions of the character (e.g., the curvilinearity, the level of concavity or convexity, of a bracket);
- the shape of the glyph of small sizes characters and that of big sizes;
- the position of the glyph, points of control, according to the dimensions of the character;
- the dimensions, the width and the length, of the box of the character;
- the position, the height and depth, of the character compared to the baseline;
- the position of the character with respect to the other characters of the same line, the blank between characters;
- the position of the character compared to the expression covered by this character;
- etc.

Choices should be made with respect of:

- the nature of the symbol that determines the space before and after the composed symbol (e.g., `\mathinner`);
- the lengthening to be added to the initial length (respectively the width) of the expression covered by the brackets (respectively the kashida) under the terms of the rules of the typography;
- the form of boundaries of kashida in other to join it with the proceeding part and/or the following one (e.g., for the symbol of sum, product and limit);
- etc.

For example, the parameters that determine the brackets are:

- the level of curvilinearity of the bracket;
- the level of fat of the middle of the bracket;
- the level of fat of the ends of the bracket;
- the shape of the ends of the bracket, as they are to be identical in the top and the bottom;
- the shape of the bracket depends on the size of the covered expression.

Examples

$$\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}\begin{pmatrix}0&1&2&3\\4&5&6&7\\1&2&3&4\\5&6&7&8\end{pmatrix}\begin{pmatrix}0&1&2&3&4\\5&6&7&8&9\\0&1&2&3&4\\5&6&7&8&9\\0&1&2&3&4\end{pmatrix}$$

The kashida always holds the same thickness but the concavity of the kashida varies within limits fixed by the rules of the calligraphy of the style *Naskh* [3].

The symbol ـــجـ obtained by the command `\lsum`, from the system **RyDArab**[9] [7] [8], is a composition of the fixed part جـ, obtained with the `xnsh` font from ArabTEX[10] [6], and of the extensible rectilinear part ▬ whose effective size depends on an automatic way of the context.

The symbol ـــجـ obtained by the command `\csum`, from the system **RyDArab** with the present package **CurExt**, is a composition of the fixed part جـ , of the font **NasX**[11], and the variable-sized curvilinear part kashida ـــ whose effective size depends on an automatic way of the context. The problem of drawing these parts of a component symbol arises then.

Examples

$$\overset{ص}{\underset{1-ح-س-ط=ب}{\text{ـــــــــــجـ}}}\qquad\overset{ص}{\underset{1-س-ط=ب}{\text{ـــــــــجـ}}}\qquad\overset{ص}{\underset{1-ط=ب}{\text{ـــــــجـ}}}$$

Syntax of commands Hereafter, some commands offer by **CurExt** package.

The syntax for parentheses command is:

```
$\parentheses
 {\matrix{1 & 2 & 3\cr
          4 & 5 & 6\cr
          7 & 8 & 9\cr
 }}$
```

$$\boxed{\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}}$$

The syntax for open (or left) parenthesis command is:

```
$\openparentheses
 {\matrix{1 & 2 & 3\cr
          4 & 5 & 6\cr
          7 & 8 & 9\cr
 }}$
```

$$\boxed{\left(\begin{matrix}1&2&3\\4&5&6\\7&8&9\end{matrix}\right.}$$

The syntax for close (or right) parenthesis command is:

[9] The extension **RyDArab** is a system for processing mathematical documents in an Arabic presentation. It allows the composition of mathematical expressions with specific symbols spreading out from right to left according to the Arabic writing. It has been developed by the author.

[10] The extension ArabTEX is a system for processing Arabic textual document. It has been developed by K. Lagally.

[11] The font **NasX** is a kernel of an Arabic mathematical font. It offers some Arabic literals symbols. It has been developed by the author.

```
$\closeparentheses
 {\matrix{1 & 2 & 3\cr
          4 & 5 & 6\cr
          7 & 8 & 9\cr
 }}$
```

$$\left.\begin{matrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{matrix}\right)$$

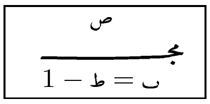
The syntax for open (or right) parenthesis, in an Arabic mathematical presentation, command is:

```
\arabmath
$ {\openparentheses
 {{\matrix{1 & 2 & 3\cr
           4 & 5 & 6\cr
           7 & 8 & 9\cr
 }}}}$
```

$$\left.\begin{matrix}3 & 2 & 1\\ 6 & 5 & 4\\ 9 & 8 & 7\end{matrix}\right)$$

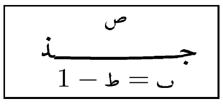
The syntax for Arabic sum command is:

```
\arabmath
$ {\csum_{b=T-1}^{s}}$
```



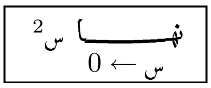
The syntax for Arabic product command is:

```
\arabmath
$ {\cprod_{b=T-1}^{s}}$
```



The syntax for Arabic limit command is:

```
\arabmath
${\clim_{c \to 0}
 c{{}^2}}$
```



The syntax for Arabic kashida command is:

```
\arabmath
$ {\kashida{9mm}}$
```

Conclusions

The application **CurExt** allows the composition of variable-sized curvilinear symbols. This system will make it possible to compose automatically delimiters of mathematical expressions that can vary in a bi-dimensional way. It also allows the composition of kashida of the Arabic mathematical symbols such as the symbols of *sum*, *product* and *limit*.

The number of various sizes for the brackets or the kashida is limited to 256. Beyond this limit, the system will use the smallest size higher than the required one or the biggest size already computed. The number of occurrences of the same size is unlimited.

A choice of the parameters to compose brackets is made with respect of typographic and calligraphic rules in use. A compromise between certain parameters is necessary. In some cases, the choice is subjective.

The system **CurExt** will allow looking after the typography of the variable-sized curvilinear symbols such as brackets or the symbol of integral. It also makes it possible to take into account the compliance with the rules of the calligraphy of a cursive writing such as Arabic. The kashida will be carried out in the strictest respect of Arab calligraphy.

A significant application of such a system will be the justification of a text in a cursive writing through complying with the calligraphic rules.

In addition to the previous limits, `METAFONT` fonts under an `ASCII` encoding generate many problems on the new formats of multilingual electronic documents. A study is started for `PostScript Type 1/3` or `OpenType` fonts under an `Unicode` encoding.

References

[1] Jacques André and Irène Vatton, *Dynamic optical scaling and variable-sized characters*, EPODD **7** (1994), no. 4, 231–250.

[2] Daniel M. Berry, *Stretching Letter and Slanted-baseline Formatting for Arabic, Hebrew and Persian with `ditroff/ffortid` and Dynamic `POSTSCRIPT` Fonts*, Software–Practice & Experience (1999), no. 29:15, 1417–1457.

[3] Mohamed haCm al XTaT, *Les règles de la calligraphie arabe, Ensemble calligraphique des styles d'écritures arabes*, Univers des livres, Beyrouth, Liban (1986).

[4] Donald Ervin Knuth, *The TEXbook*, Addison-Wesley, 1984.

[5] ———, *The METAFONTbook*, Addison-Wesley, 1986.

[6] Klaus Lagally, *ArabTEX - Typesetting Arabic with Vowels and Ligatures*, EuroTEX'92, Prague (1992).

[7] Azzeddine Lazrek, *A package for typesetting Arabic mathematical formulas*, Die TEXnische Komödie, DANTE e.V. **13** (2001), no. 2/2001, 54–66.

[8] ———, *Aspects de la problématique de la confection d'une fonte pour les mathématiques arabes*, Cahiers GUTenberg **39–40, Le document au XXI$^{\mathrm{e}}$ siècle** (2001), 51–62.

[9] Vincent Quint and Irène Vatton, *Grif: an interactive system for structured document manipulation*, Text Processing and Document Manipulation, ed. J. C. van Vliet **Cambridge University Press, Cambridge, UK** (1986), no. 4, 200–213.

[10] Johny Srouji and Daniel M. Berry, *Arabic formatting with `ditroff/ffortid`*, EPODD **5** (1992), no. 4, 163–208.