Edition with Arabic mathematical notation

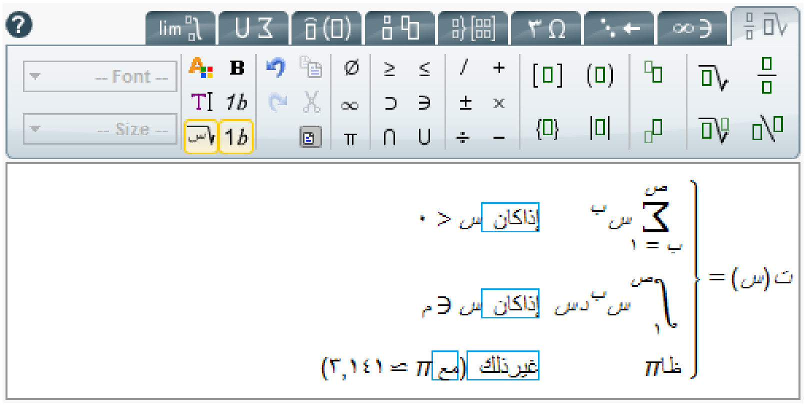
Daniel Marquès Solé
14 May 2013

This paper is related to user interfaces for editing formulas and the challenges that introduce the Arabic mathematical notation. A brief introduction to Arabic mathematical notation is provided with special attention to the MathML format and the difficulties that the formula editor should solve.

Introduction and state of the art

Some preexisting work exists about how display Arabic formulas and how to represent them with MathML [1]. There is also a LaTeX package for generating Arabic mathematical documents [2]. Edition of formulas cannot be understood without first carefully study how to display them.

There are many examples of formula editors, also called equation editors. For example, WIRIS editor is a web based formula editor from Maths for More and is the first JavaScript editor to provide a rich user interface for the edition of the Arabic mathematical notation [4]. The Dadzilla MathML browser for Arabic mathematical presentation also includes a MathML editor [3]. It would not be fair not to mention Math Type editor or the Microsoft Equation Editor among others.



WIRIS editor configured with the right-to-left layout of the toolbar and with some icons redesigned for Arabic. Example extracted from [1].

A formula editor allows building in a visual and easy way all kind of formulas. Adding new constructions no available before is quite easy to develop and usually requires some localized modifications. On the other hand, the Arabic mathematical notation involves more general improvements.

In the era of tablet devices and smartphones, editing text in an alphabet different from a Latin is very common and straightforward. Let’s mention Arabic, Chinese, Cyrillic, Devanagari, Greek and Hebrew. For the case of our formula editor, we have strongly benefited from these technical advances. Because WIRIS editor is JavaScript based, it works in all browsers including tablet devices.

Arabic mathematical notation overview

The mathematical notation depends strongly on the country and other factors like the education level. For example, math formulas used in Morocco are quite different from those used in Saudi Arabia or Egypt. While the former are Latin, the latter have all the traits of the Arabic mathematical notation explained in this section. A classification by countries has been done according to [5]. In addition, French or English is the higher education language in many regions. Hence, the math formulas follow the European mathematical notation.

The most obvious difference between Latin formulas and Arabic formulas is the use of the Arabic alphabet both for letters or numbers. From the technical point of view, the use of the Arabic alphabet is not a real challenge because all modern operating systems and browsers support them.

Arabic text has a feature called ligatures which consists on joining two consecutive letters in the same word. For example, ا + ه + ن yield نها. Thus, each letter has an isolated form which adopts an initial, a medial or a final form when forming part of a word. Again, current operating systems and browsers support ligatures.

The Arabic alphabet is written from right to left (RTL) and in most countries formulas are also from right to left. For example, -1 might be written as $1\,-$. For completeness, Latin alphabet is said to be left-to-right (LTR). A related topic is the so-called mirroring of formulas, which means that a formula layout is adapted to the right-to-left writing in such a way that it resembles Latin formula seen though a mirror. One notorious example is the square root.

$$\overline{\text{٣} + \text{س}}\!\!\diagdown$$

Some characters like, $\in, \leq$ are also mirrored. See [6] for a general discussion about Arabic mathematical symbols in Unicode. In addition, the Unicode group has done an excellent work defining the pairs of mirrored symbols [7].

There are also styling and calligraphic issues. In some regions, certain symbols like “Σ” (for iterated sums) and “lim” (for limits) became stretchy letters of the Arabic alphabet.

$$^{2}\text{س}\ \underset{\infty + \leftarrow \text{س}}{\text{نهـــا}} \qquad \text{ل}\!\!\boxed{12}$$

Limit and factorial

From the point of view of a software implementation, the difficulty here is that the “stretchy part” is not a straight line but a curve.

MathML is the most used XML standard for expressing mathematics. At present the Arabic mathematical notation is supported by MathML 3 together with the Unicode standard. It is worth to mention the “dir” attribute which states whether a given scope is left-to-right or right-to-left. At the moment of writing this article, Firefox and Dadzilla were the only browsers that supported MathML natively which also implemented RTL features.

Challenges of editing overview

In the previous section we introduced the Arabic mathematical notation. Now we will explain the issues specific to the editing process.

Editors use a visual mark called *caret* to express where the next typed character will be placed inside an editing formula. When considering Arabic text, placing such caret is not obvious. A quick survey with different editors and operating systems shows different behaviors. Related to the caret, it is the selection of different parts of a formula with the intention of copy, cut and paste. In the following sections we will explain how we solved it for WIRIS editor.

The input of Arabic letters is possible provided that a proper keyboard is configured. This is actually a software configuration, not a hardware requirement, which means that any keyboard can be configured to input Arabic letters. Indic-Arabic numbers are not easy to input in a Microsoft Windows OS. For this reason, we have enabled a mechanism to input them with any keyboard configuration.

Variables in the Arabic alphabet are expressed using the isolated form of the letters. Since some formulas also contain words with ligatures, any formula editor should allow toggling between joining letters to get ligatures and displaying them isolated.

MathML

For its input and output formats WIRIS editor has chosen MathML. The MathML 3 specification already has features that cope with the Arabic notation and RTL. Firefox browser currently supports MathML and the “dir” attribute. Unfortunately, the Chrome browser just turned off the MathML support and it was not possible to check the Arabic features provided by the “dir” attribute.

We assume a basic knowledge about MathML markup language. MathML is built upon XML and is used to express formulas for mathematics and science in general. It has two variants presentation MathML and content MathML. For the purpose of this document, we will consider only presentation MathML because it captures all visual properties of a formula, which are culture-dependent, while content MathML intends to represent the mathematical meaning.

For example, $\frac{1}{2} - 10xy$ is expressed in MathML as:

```
<math>
    <mfrac>
        <mn>1</mn><mn>2</mn>
    </mfrac>
    <mo>-</mo><mn>10</mn><mi>x</mi><mi>y</mi>
</math>
```

The representation is not unique and many alternatives are possible. Some observations are worth to be mentioned

1) Symbols are classified using the <mn>, <mi> or <mo> tags for numbers, variables and operators, respectively. While 10 is inside a <mn> tag, each letter x and y are isolated in different <mi>.
2) There is no markup to hint that 10, *x* and *y* are multiplied or about the precedence of the operations.

An Arabic formula like , $\sqrt{س+٣}$ would be expressed as:

```
<math dir="rtl">
    <msqrt>
        <mi>س</mi><mo>+</mo><mn>٣</mn>
    </msqrt>
</math>
```

The “dir” attribute determines that the formula is presented from right to left. No other indication that a formula is Arabic would be found than the “dir” attribute and the Arabic characters by themselves.

Right to left

One of the main differences with the European languages is that the Arabic language is written from right to left (RTL). With the exception of numbers that are written from left to right (LTR). While Modern Standard Arabic is official or national language of most Arabic countries and yields uniformity in the writing system, how to write formulas is not so well standardized. Thus, in Morocco, formulas are written LTR but in the other Arabic countries are RTL. For example,

Morocco: $-3x+3$
Saudi Arabia: $-٣س+٣$

Formulas usually contain general text, for example, used to comment the entire formula or part of it. This implies that no matter which format is used in the formula, even if it is Latin, it should still be necessary to input Arabic text. For these cases, an input text box is provided by WIRIS editor. Inside the input text box, no formulas are allowed, Arabic text is always with ligatures and the presence of a single RTL symbol converts the whole content of the box RTL. The MathML standard represents general text using the <mtext> tag.

In practice, many formulas contain a mixture of LTR and RTL content. For the particular case of text without formulas, it exists the Unicode bidirectional algorithm with states how to be positioned a sequence of characters with a mixture of LTR and RTL text.

We introduce here the internal representation of a text (or formula) in contraposition to the display representation. The internal representation describes the logical order of the symbols: first, second, …, last, independently to how they are drawn. More precisely, the internal representation is how the given text will be digitally recorded. Internal representation is expressed left to right only because this document is written in English. It is not obvious how a given internal representation will be displayed, which is what precisely the Unicode bidirectional algorithm solves.

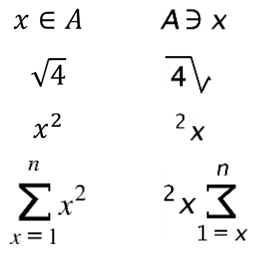
In the following, the lower case letters (x, y and z) represent LTR letters and upper case letters (A, B and C) represent RTL. In the internal representation we will place the letters inside framed boxes.

Display representation: `xyzCBA`
Internal representation: $\boxed{A}\boxed{B}\boxed{C}\boxed{x}\boxed{y}\boxed{z}$

According to the internal representation, we will read in order: A, B, C, x, y and z.

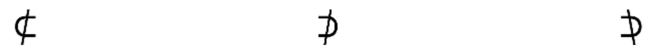
Mirroring

Right to left mathematical directionality is related to mirroring, with respect to the Latin notation. By mirroring we mean that some mathematical symbols and notations, as consequence of the right-to-left writing, are accordingly adapted. For example,



From the point of view of a tool that displays formulas, mirroring the square root or the superscripts implies adapting an existing algorithm. On the other hand, other character-like-symbols are easily mirrored because the mirrored symbol drawing (glyph) already exists. For example, to display the mirroring of "element of", ∈, you can use "contains as member", ∋. Since this is only a visual effect, the Unicode point is still "element of". Serifs and the negation line should be properly adapted when mirroring a symbol:

LTR symbol Incorrect mirroring Mirrored RTL symbol



The question now is to know which symbols are mirrored and how to draw them. A partial answer is given by the Unicode group [7]. The existence of a corresponding mirrored symbol does not guarantee that it is drawn optimally because the serif traits and any negation line should be properly adapted as seen before.

Another place where information about the mirrored version of a symbol can be found is the font typeface files. Mirrored version of the STIX fonts can be found [8]. In general, a web application does not have access to this information.

Maybe it is not clear at this point, but while some symbols can be mirrored keeping the same Unicode point other symbols needs a different code.

LTR symbol	**Unicode point (hex)**	**Mirrored RTL symbol**	**Unicode point (hex)**
∈	8712	∋	8712
→	8594	←	8592

The mirror of "belongs" keeps the Unicode point but the "right arrow" changes the code.

Thus, deciding what symbols should be mirrored cannot be left to the font-rendering engine if we want to obtain the same result under different operating systems or browsers.

Some approaches about RTL and mirroring implementation

The starting point is that we have a LTR implementation of a formula editor and that the rendering algorithm, which is responsible for displaying a formula, will have to be adapted to support RTL. The rendering algorithm decides where each part of a formula should be displayed in a given system of coordinates.

Formulas are expressed in MathML as a tree of XML tags. A simplified rendering algorithm works in two phases: a bottom-up phase is used to compute the width, the height of the nodes and its relative position respect to the parent. The second phase is a top-down phase that commands to draw the nodes.

There are three approaches:

1) Change the coordinate system globally inside a formula. By changing the coordinate system, for example x→w-x, where w is the total width of a formula it is possible to mirror it. This works, of course, assuming you place all elements of a formula with absolute coordinates but relative to the whole formula. The main drawback of this method is that is difficult to achieve the mixture of LTR and RTL content.
2) Each node of the MathML tree is aware of the directionality and knows how to compute it. The drawback is that each node would have two algorithms: one for each directionality, RTL and LTR. In long term, the duplicity of the code would negatively affect its sustainability.
3) WIRIS editor solution combines the previous two approaches. The directionality is controlled node by node. The first phase is not aware of the directionality and the second phase inverts the coordinate system locally inside a node.

Caret position and its interpretation

Text and formula editors use a vertical bar, "|", called caret as a way to hint where the next typed character will be inserted. For example, in the following visual representation

xyz|CBA

the caret is displayed between LTR and RTL symbols. When a new key is pressed, where is it going to appear? The answer is that it actually depends on many factors like the typed character directionality, the operating system, the browser or even the application.

WIRIS editor stores the caret using the internal representation:

A	B	C|	x	y	z

WIRIS editor sets the caret after C because, when mixing RTL and LTR content, there is a preference with RTL. The other option would be to place the caret after z, at the end. Other systems interpret the caret differently.

Consequently, when a new key is pressed, WIRIS editor choice is the following:

1. if the character is RTL, **D**, we will get: `xyz`**|D**`CBA`
2. otherwise if it is LTR, **w**, we obtain: **w|**`xyzCBA`

The choice of the internal representation for placing the caret is apparently counterintuitive; but solves the scenario when the user continuously alternate between RTL and LTR characters.

Ligatures and Arabic as text

The Arabic language has a feature called ligatures. This means that two consecutive letters of the same work are combined together. Thus any letter can appear as final, medial or initial form. For example,

Isolated letter (Unicode point)	isolated ا (627)	isolated ه (647)	isolated ن (646)
Contextual form (Unicode point)	final ـا (FE8E)	medial ـهـ (FEEC)	initial نـ (FEE7)
Ligature	نها		

Browsers handle ligatures automatically and content does not usually contain any contextual form Unicode point. A formula editor, however, needs the contextual form Unicode points in order to compute the position of the caret or when low level drawing is required.

Variables are expressed only using the isolated form. Since a formula usually contains full words for well-known functions or some sentences to explain the formulas, there should be a way to write both letters joined with ligatures and isolated letters for variables.

Fortunately, MathML has a feature, which although not exactly the same, can be used to solve this issue. With Latin or Greek letters, the italic font style means a variable and each letter are placed inside a <mi>. More than one letter inside a <mi> are not displayed in italic and means a function name. For example, $\sin x$ is <mi>sin</mi><mi>x</mi>. Following this idea, in Arabic, italic will place letters in its isolated form to express variables and non-italic will do all the ligatures to express function names. For general text, it is possible to remove the italic from a sentence but a better approach will be to use the text icon that will place all text inside an <mtext> and will perform all ligatures.

Numerals

Three sets of numbers are considered:

European	0	1	2	3	4	5	6	7	8	9
Arabic-Indic	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
Eastern Arabic-Indic	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹

The use of numerals depends on the country configuration. For example, in Morocco, the European numbers are used but in Saudi Arabia the Arabic-Indic are preferred. Arabic non-European numerals yield two challenges. The first one is related to the directionality. Numerals are written LTR which results in a mixture of LTR and RTL for Arabic formulas. This is easily solved because WIRIS editor allows mixing both directions.

The second challenge is that we cannot rely on the operating system for typing them. For example, the Arabic keyboard configuration for Windows still types the European-Arabic numbers when the numeral keypad is pressed. For this reason we decided to override this behavior and two buttons are used to toggle what numeration is used when using the numbers of the keyboard or the numeric keypad.

Locales

Each Arabic region has its own peculiarities which might also depend on the education level. For example, formulas in primary or secondary can be different from the University. Although a web application might notify the current country of the user using the editor, this does not provide enough information to define the desired behavior. Thus, all Arabic features should be available for all users independently of the country. As consequence, anyone should be able to type RTL formulas even with an English configuration of the editor.

A *locale* is a specific language and parameter configuration for a given user related to its region and culture. From the point of view of WIRIS editor, there are three parameters of the locale to be configured

1. whether the directionality is RTL or LTR,
2. the default numeration system and
3. the user interface language.

Let’s remember that the keyboard layout and, thus, the alphabet are configured directly at the operating system level.

A well configured Web application is able to notify the current language and country locale of the user. Any editor might use this information to select the default behavior. However, in practice an author would like to use a feature different from the default one. For this reason, each of the above features has its own representation in WIRIS editor:

س	Selects the right-to-left edition
٤٦	Selects the Arabic-Indic numerals
۴۶	Selects the Eastern Arabic-Indic numerals
TI	Selects the text mode

Conclusions and future

Adapting WIRIS editor for the Arabic mathematical notation has been a progressive work. We initially allowed writing Arabic text inside the <mtext> box with proper ligatures. The following step was more challenging and we introduced the RTL, mirroring and default configuration according to the locale.

Despite the effort done with this release of the editor, there are some issues that are not addressed. For example, some Arabic stretchy symbols with Arabic calligraphy were not implemented. Although is not the optimal solution, these same symbols can be replaced by a mirrored version of the Latin version.

A feature that might be requested is an automatic tool for converting formulas from LTR to RTL. We understand that the name of variables cannot be automatically changed. But numbers can be adapted and the directionality of some symbols could be mirrored. At present, it is possible to toggle the directionality of any formula but many character-like symbols would need a manual adjustment.

In this paper we have introduced the Arabic mathematical notation and its implication for a formula editor. One task that is pending to do is to perform an exhaustive survey in order to identify how the different Arabic mathematical notation traits are used according to regions or socials patterns.

To finish we would like to thank the members of the math distribution list of the World Wide Web Consortium (W3C). Many doubts we had during the implementation of WIRIS editor were successfully clarified by its members. In special most information available online comes from Azzeddine Lazrek, the Unicode Consortium and the W3C.

Bibliography

[1] M. Eddahibi, A. Lazrek and K. Sami, "Arabic mathematical notation," 2006. [Online]. Available: http://www.w3.org/TR/arabic-math/.

[2] Cadi Ayyad University. Department of Computer Science, "RyDArab," 2001. [Online]. Available: http://www.ucam.ac.ma/fssm/rydarab/.

[3] Maths for More, "WIRIS editor demo," 2013. [Online]. Available: http://www.wiris.net/demo/editor/demo/ar_sa/.

[4] M. Eddahib and A. Lazrek, "Dadzilla. A MathML browser for Arabic mathematical presentation," 2004-2010. [Online]. Available: http://www.ucam.ac.ma/fssm/rydarab/dadzilla.htm.

[5] Maths for More, "Arabic numbers and math notation by countries," 2013. [Online]. Available: http://www.wiris.com/en/editor/docs/resources/arabic-numbers-countries.

[6] M. J. E. Benatia, A. Lazrek and K. Sami, "Arabic Mathematical symbols in Unicode," 2005.

[Online]. Available: http://www.ucam.ac.ma/fssm/rydarab/doc/communic/unicodem.pdf.

[7] The Unicode Consortium, "Bidi_Mirroring_Glyph property," 2012. [Online]. Available: ttp://www.unicode.org/Public/UNIDATA/BidiMirroring.txt.

[8] K. Hosny, "The XITS font project," 2011. [Online]. Available: https://github.com/khaledhosny/xits-math.

[9] The Unicode Consortium, "Unicode Bidirectional Algorithm," 2012. [Online]. Available: http://www.unicode.org/reports/tr9/.