# Towards Universal Rendering in MathJax

Davide Cervone
MathJax Consortium
Union College, NY
dpvc@union.edu

Peter Krautzberger
MathJax Consortium
krautzource UG
peter.krautzberger@mathjax.org

Volker Sorge
MathJax Consortium
University of Birmingham, UK
V.Sorge@cs.bham.ac.uk

www.mathjax.org*

## ABSTRACT

Mathematics support on the web is currently very limited mainly due to the lack of browser support for the dedicated markup language MathML. While for visual display JavaScript libraries like MathJax can ensure flawless rendering of formulas across all platforms and browsers, there currently exists no system independent assistive technology solutions for Mathematics. We report on the attempt to transform MathJax into a universal rendering solution, providing accessibility also for readers with visual, print, and motor impairments. As our goal is to integrate seamlessly with other assistive technology solutions, we are faced with a number of technical challenges to provide a consistent user experience, regardless of the particular combination of platform, browser, and screen reader employed by readers. This paper presents MathJax's accessibility features and some of the technical challenges to motivate a wish-list for future standards to enhance web accessibility for STEM content.

## Keywords

STEM Accessibility, Mathematics, MathJax

## 1. INTRODUCTION

The current state of mathematics support on the web is in a sad state of affairs. Although Mathematical formulas on the web can be represented in their own specialized markup language, MathML [3] as part of the HTML5 standard [2], only very few major browsers implement MathML rendering natively, leading to sketchy support for displaying formulas included in pure MathML on web pages. Those browsers that do provide MathML support generally only offer it partially, or lack its full integration into other technologies of the Open Web Platform (OWP). For example, Firefox does not implement the `GlobalEventHandler` API and has inconsistent support for Cascading Style Sheets (CSS) on MathML elements. While Safari has a more modern integration into the standard interfaces for the DOM (the tree structure representing the HTML document), it does not implement all MathML elements, such as `maction`, and it collapses line-broken content onto the same line. While neither Mozilla nor Apple are actively developing the MathML support in their browsers, other browser vendors like Google (Chrome) and Microsoft (IE/Edge), or 55%-75% of the browser market, list MathML support as simply not planned.

It would be unfair, however, to blame the current lack of MathML support on the browser vendors alone. Equal blame must fall on the failure of standardization efforts at the W3C to pro-actively advance MathML in the face of developments in other OWP standards, and to adapt it to the modern realities of browser rendering and web programming. MathML is underspecified in terms of describing layout rules, but more importantly, it lacks corresponding modules in CSS and ARIA (assistive technology standards) to clarify implementation details for math layout and semantics in browser engines. In addition, some layout features are incompatible with their corresponding CSS. For example `mtable` layout has simultaneously more (e.g., labeled rows), less (e.g., border options), and incompatible (relative spacing) features compared to regular HTML `table`s. Similarly, `mpadded` is incompatible with CSS `padding`. Yet we also see features of math layout being subsumed by modern CSS technology (e.g., flexbox) and re-invented in upcoming standards (e.g., in-table alignments). Thus we can expect more incompatibilities with existing specifications in the future.

While MathML is a markup language for both the syntactic composition of expressions via Presentation MathML and highly specialized semantic representation via Content MathML, that will certainly continue to be used in publishing workflows, its future as a web standard is doubtful. As a consequence, authors of mathematical web content and platforms supporting mathematical authoring rely on JavaScript libraries such as MathJax [4] to ensure flawless rendering of formulas across all platforms and browsers.

Given the sad state of visual rendering of math on the web, it is of little surprise that the web accessibility of math is even worse. Before 2012, math accessibility on the web was synonymous with MathPlayer [8] on Internet Explorer. With IE11 deprecating plugins like MathPlayer, and with ChromeVox [9] and VoiceOver [1] adding some MathML support, the landscape has grown more complex. Unfortunately, the core problem for web-based math accessibility has not changed: browsers offer very limited MathML support. Consequently, the majority of assistive technology solutions on the market today rely to some degree on the

*This work was partially supported by the Alfred P. Sloan Foundation.



*W4A'16, April 11-13, 2016, Montreal, Canada*

 ISBN 978-1-4503-4138-7/16/04...$15.00
DOI: http://dx.doi.org/10.1145/2899475.2899494

```
<math>
  <mi>a</mi>
  <msup>
    <mi>x</mi>
    <mn>2</mn>
  </msup>
  <mo>+</mo>
  <mi>b</mi>
  <mi>x</mi>
  <mo>+</mo>
  <mi>c</mi>
  <mo>=</mo>
  <mn>0</mn>
</math>
```

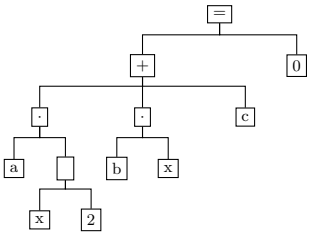



**Figure 1: Quadratic equation $ax^2 + bx + c = 0$ in presentation MathML and as semantic term tree.**

ability of MathJax to turn Mathematics on the web into a uniform MathML representation regardless of whether its source is in LaTeX, ASCIIMath, or MathML notation.

It is therefore a natural step to turn MathJax into a *Universal Rendering Solution*. That is, making it an inclusive software solution in the *Universal Design* sense, rendering mathematics in a way that makes it accessible to a wide spectrum of readers, regardless of their disabilities, while retaining the same quality and user experience, regardless of the platform, browser, or assistive technology solution employed by a user. The basis forms a uniform semantic enrichment of MathML expression (presented in §2) that allows us to implement a number of accessibility features (§3). However, our requirement to be a platform independent solution, leads to a number of technical challenges and suggestions for future developments and standards to enhance STEM accessibility on the web, which we discuss in §4.

## 2. SEMANTIC ENRICHMENT

One of the main challenges for turning MathJax into a universal renderer is that it was originally designed exclusively for displaying formulas, offering a number of different rendering solutions, such as HTML with CSS or SVG. Thus, although MathJax uses MathML as its internal format, expressions in the DOM are collections of `div` and `span` elements or SVG graphics elements, which often retain very little of the structure of the MathML tree they represent.

A second problem stems from the semantic weakness of Presentation MathML, which contains relatively little mathematical information. In particular, MathML expressions are often geared towards display, and so omit or obscure much of the actual mathematical structure.

We have overcome these problems by imposing a "light" semantic interpretation on math expression and generating a tree representation that can be embedded into rendered MathJax expressions to ensure a similar user experience across browsers. The idea of the semantic interpretation is an extension of the heuristics implemented in the screen reader ChromeVox [9], which effectively rewrites a flat MathML expression into a term tree structure by first interpreting the basic nature of symbols and propagating this through the expression to determines the scope of operators, relations, etc. Figure 1 presents this transformation for the example of the quadratic equation $ax^2 + bx + c = 0$, which is rewritten from its Presentation MathML form in the middle into its semantic interpretation on the right.

The resulting semantic tree can be understood as an orthogonal view of the mathematical expression, and we make concrete use of it by embedding it as `data` attributes within MathJax's internal MathML representation. This then allows MathJax's various renderers to push these data attributes into the actual rendering elements used in the DOM, thus retaining the semantic tree structure and making it available within the page. Data attributes provide a fast and standardized means of retrieving information from the DOM that is fully consistent with HTML5 practices.

## 3. ASSISTIVE TECHNOLOGY SUPPORT

The assistive technology extension that we have built for MathJax is mainly aimed at supporting users with reading disorders, such as dyslexia, and visual impairments. However, some aspects if it could also be used as a general aid for readers unfamiliar with the content or for learners at different levels. We summarize the main features in this section.

### 3.1 Highlighting

Mathematical expressions generally are large collections of mostly unconnected symbols in a two-dimensional layout that can be particularly daunting for readers with dyslexia. Reading comprehension can be supported, however, by a choice of high-contrast colors [7] and selective highlighting [5]. We realize this in MathJax by offering changes of fore- and background colors, and selective highlighting of sub-expressions of a complex formula. We again exploit the semantic structure of the formula to provide a more meaningful highlighting, as the following example demonstrates (it is the first line of the example given in Fig. 2) by contrasting the syntactic highlighting above against the semantic highlighting below; every other element is highlighted.

$$I_\nu(\nu^{-1},1) = \frac{\pi^2}{4}\ln\frac{(1+\nu)^{1+\nu}}{\nu^\nu} - \frac{7\zeta(3)}{8}\nu + 2\int_1^{\frac{1-\nu}{1+\nu}}\frac{\chi_3(v)}{(1+v)^2}\mathrm{d}v$$

$$I_\nu(\nu^{-1},1) = \frac{\pi^2}{4}\ln\frac{(1+\nu)^{1+\nu}}{\nu^\nu} - \frac{7\zeta(3)}{8}\nu + 2\int_1^{\frac{1-\nu}{1+\nu}}\frac{\chi_3(v)}{(1+v)^2}\mathrm{d}v$$

In practice, sub-expressions are highlighted when hovering over them with the mouse pointer, or by switching highlighting on permanently. This can be recursively refined for sub-expressions, e.g., hovering on the denominator or numerator of a fraction only. For permanent highlighting, the opaqueness of the background gradually increases in nested sub-expressions. Nevertheless, in large expressions, highlighting is of limited utility in getting an overview of the structure of a formula; to aid this further, we introduce a technique for structural abstraction in the next section.

### 3.2 Structural Abstraction

The idea of structural abstraction is to assist readers by simplifying the structure of the formula initially and letting them individually explore the equation by manually expanding selected sub-expressions. We have implemented this via a user interface based on MathML's `maction` element (cf. [3, 3.7.1]). It allows users to explore the content using click, keyboard, and touch events. The `maction` elements are nested so that only the next level of the collapse is revealed. The element indicating collapsed content is a simple Unicode construction, ◀$X$▶, with X indicating the top-level structure that was collapsed. For example we use ◀+▶ to indicate a sum, ◀$\int$▶ an integral, etc.

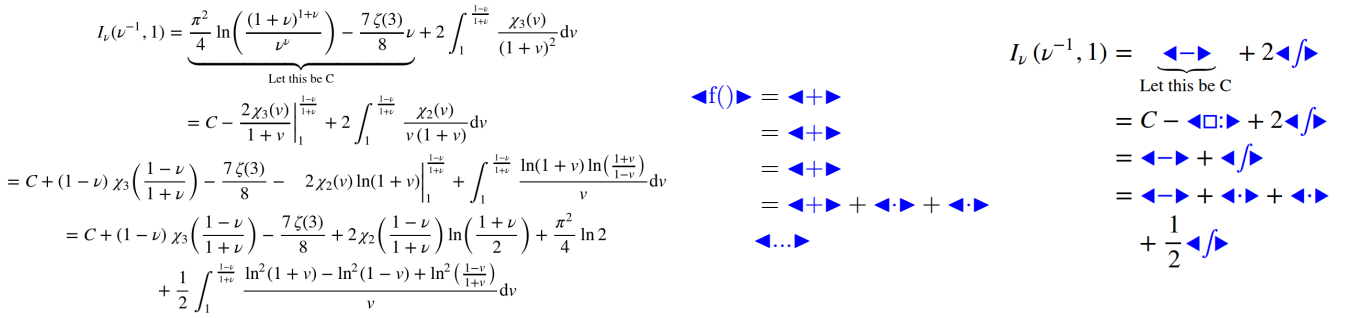
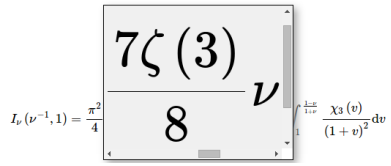



**Figure 2: Two abstractions of the complex equation system on the left.**

Fig. 2 demonstrates three different states of collapse of our example equation. To determine which parts are collapsed, our algorithm somewhat surprisingly does not calculate sub-expression width, since these are not available before rendering. Instead, it estimates the complexity of an expression by recursively evaluating the enriched MathML tree. For example, token elements are assigned a complexity value according to their string content while more complex elements such as roots or fractions are assigned the sum of their children's complexity measures modified by a value corresponding to their own visual complexity. Cut-off values for each semantic type are then used to decide which expressions are collapsed. The parameters (character weight, operator modifier, cut-off) are configurable by the author.

## 3.3 Magnification

While changing contrasts can already help low-vision users, many rely on screen magnification to read content. MathJax supports magnification in multiple ways: standard browser zoom is supported by re-rendering mathematics at the new size. MathJax also provides global scaling settings to enlarge all math elements at once. Individual math expressions can be further magnified using a "lens" that ordinarily offers a zoom window on top of an expression, with scrolling to pan across the entire zoomed expression. Exploiting the semantic markup, this lens can be restricted to particular sub-expressions only as for instance in the example below.



On small devices, zooming usually is not beneficial, as it leads to overflow and two-dimensional scrolling. For magnification, it is possible to avoid excessive scrolling or panning by re-flowing with line-breaks; but applying line-breaking programmatically to long or complex equations often destroys their readability due to cluttered results. We exploit the semantically enriched markup to perform line-breaks at mathematically appropriate positions only. While this can occasionally lead to less optimal usage of screen real-estate, it does significantly help readability of formulas and also yields good results on small form factors.

## 3.4 Aural Rendering

To support screen-reader users, but make them independent of their particular screen-reader's Math capabilities, MathJax now exploits direct access to the speech-rule engine [9] that generates the semantic-tree transformation in the first place and provides aural rendering of mathematical expressions that can be directly exposed to a screen reader. Speech string computation is based on the embedded semantic tree and uses the MathSpeak rule set [6]. Speech strings for a formula are either pre-computed and embedded as another data attribute, or generated on the fly.

To expose the speech strings to a screen reader, MathJax introduces a dedicated assertive ARIA live region into the DOM and updates it with the desired speech output. Speech strings can be computed not only for an entire formula, but also for all its sub-expressions, which is helpful for interactive exploration. We describe this next.

## 3.5 Interactive Exploration

Since mathematical formulas are generally of a complex nature, and already a small formula can make for a complex utterance, just listening to an expression once generally is not enough for comprehension. It is particularly important, therefore, to provide the reader with a means of engaging with formulas interactively. MathJax offers an interface for interactive exploration of mathematical expressions that allows a reader to step through sub-expressions either along the syntactic structure or the semantic tree.

Technically, this is realized by giving each math expression an ARIA role of `application`, and upon entering an expression, the user can walk it using the cursor keys. Exploration is supported by both highlighting and voicing sub-expressions under consideration, where the former is achieved by updating CSS properties of elements and the latter by updating the ARIA live region introduced in the DOM. Again, since the walking is effectively based on the embedded semantic structure, the user experience of exploring a structure is the same regardless of the specific renderer used.

# 4. TECHNICAL CHALLENGES

The challenges we encountered fall broadly into four categories: the assistive technology (AT) landscape, testing, documentation, and standards.

While universal support is always MathJax's goal, the accessibility landscape adds additional complexity beyond browser support: the quality of the browser accessibility tree, its exposure to accessibility APIs on the OS level, and work-arounds of AT for limitations in both. Our initial support targets the most common screenreaders: NVDA, JAWS, WindowEyes, VoiceOver, ChromeVox, and Orca. We tested these screenreaders on Windows XP, 7, 8.1, and 10, OSX, and Ubuntu (where applicable) using Internet Ex-

plorer 9-11, Microsoft Edge, Chrome, Firefox, Safari, and Gnome Web (where applicable). Our tool does not yet support mobile devices or AT that does not interact with live regions (e.g. Dolphin, Claro, Texthelp, ReadSpeaker). Overall, we believe we support all feasible combinations of the above. A notable edge cases is JAWS, which built a custom solution targeted at MathJax which directly interferes with alternative solutions like ours. For more details, see our support matrix at `https://github.com/mathjax/MathJax-RespEq/wiki/Support-Matrix-a11y-tool`.

A core problem for extensive accessibility testing are the tools. While new projects help with layout issues, and while browsers are slowly adding debugging tools for accessibility, debugging advanced applications that essentially polyfill future accessibility features has very little support. Automated tools seem infeasible for now, but it seems plausible to have browser testing services (cf. SauceLabs, BrowserStack) specialized on accessibility (e.g. with pre-installed AT software). Similarly, a (community) effort for specialized virtual development environments (e.g. Vagrant) with suitable licenses for AT would be important to make testing more reliable and increase reproducibility in teams.

A common problem are, of course, inconsistent implementations. Since we have to work pragmatically with what users have today, we run into issues on all three layers mentioned earlier. From individual browsers having incomplete ARIA implementations to incomplete integration into OS APIs, to issues with live regions, to AT mishandling custom navigation, to ATs using undocumented workarounds for specific browser and OS versions. This makes it complex to debug issues, as one is easily misled as to where the actual problem lies. It was a steep learning curve for us and we believe a concerted documentation effort (cf. caniuse.com) would help enormously to reduce friction.

A specific problem for math and science is the inability to easily expose multiple alternative renderings to users. Typical examples include TEX and AsciiMath notation for mathematics, which users prefer, but it also extends to scientific subjects like Chemistry or Biology, where alternative notation, e.g., for compounds or DNA fragments, can be preferred by users. Annotations will likely provide a more general use case. From a pragmatic stand-point, we see a long-term benefit to enable more flexible rendering options, given the rich history of textual representations.

Finally, we believe our results can feed back into the development of web standards and tools based on them, thus creating paths towards richer representations of math and science that could enable any screenreader to provide exploration and voicing in a straight-forward fashion, independently from the underlying markup. In the short term, our experience could help provide feedback for existing standards and their implementations since our use case seems to expose edge cases that might have been considered negligible in the past. In the medium term, internationalization of Unicode names would greatly help with localization of tools like ours. Since AT does not voice most Unicode characters, we need to generate speech strings using the Unicode names; but there is no official translation of these, (let alone one aligning with local practices and standards such as DIN in Germany).

For the long term, this work, together with related work on chemical diagrams [10], have identified a clear need for a proper ARIA module covering the basics of STEM semantics. This closely relates to (and expands) the work of the W3C SVG Accessibility Task Force, as mathematical and scientific notation often lives on the border between textual and graphical content. While the base suite should only provide a relatively small number of attributes that allow to define features generically — for example a general enclosed element could refer to structures like radices, fences, but also boxes in graphs — it should contain room for flexible customisation to allow each subject area to adapt them to their particular semantic need. In addition it would need to allow for the specification of multiple alternative descriptions of web elements in parallel. This would allow tools like MathJax to focus on generating such information and thus eliminate the complexities of generating ARIA live regions, custom walkers, etc.

# 5. CONCLUSIONS

We have presented the new accessibility features of MathJax that aim to transform MathJax into a universal rendering solution supporting all readers, regardless of their disabilities. The features are based on a new semantic enrichment procedure for Presentation MathML, which not only ensures a uniform user experience across platforms and render solutions, but also shows promise for more advanced AT techniques in the future. The presented AT extensions are not yet available in core MathJax, but will be released as a MathJax extension usable with current MathJax versions and possibly become part of MathJax's core in version 3.0.

Although we have not yet had the opportunity to do extensive user testing, since our code is developed publicly on our GitHub repository, we have already had some user feedback, in particular from specialists, where initial reactions have been positive. Additional end user testing was enligthening. Users struggled with lack of live-regions support in their AT and the complexity of the application. Feedback indicates that options for additional speech styles and semantics will be beneficial. We spent a significant effort on technical testing in order to make the solution compatible with the most common assistive technology solutions. We strongly believe that these experiences and the observed shortcomings, should serve as a use case for the future development of standards, tools and general accessibility solutions for STEM content the web.

# 6. REFERENCES


[1] Voiceover. `apple.com/accessibility/osx/voiceover`.

[2] HTML v5.0. W3C recom., 2013. `www.w3.org/TR/html5`.

[3] D. Carlisle, P. Ion, R. Miner. MathML v3.0. W3C recom., 2010. `www.w3.org/TR/MathML3`.

[4] MathJax v2.5, 2014. `www.mathjax.org`.

[5] R. Jones. Strategies for reading comprehension: Selective underlining, 2008.

[6] A. Nemeth. Mathspeak. `www.gh-mathspeak.com`, 2005.

[7] L Rello and R Baeza-Yates. Optimal colors to improve readability for people with dyslexia. In *Text Customization for Readability Symposium*, 2012.

[8] N. Soiffer. Mathplayer: web-based math accessibility. In *7th Computers and Accessibility Conf.*, 2005. ACM.

[9] V. Sorge, C. Chen, T.V. Raman, D. Tseng. Towards making mathematics a first class citizen in general screen readers. In *11th Web for All Conf.*, 2014. ACM.

[10] V. Sorge, M. Lee, S. Wilkinson. End-to-end solution for accessible chemical diagrams. In *12th Web for All Conf.*, 2015. ACM.