附录 A　数学排版工具之要求 [1]

Jonathan Sterling

许多非 LaTeX 的超文本工具声称在数学排版方面具备一定兼容性: 举例而言, 在任何基于 HTML 的工具中都可以使用 MathML, 或者通过导入 KaTeX 或 MathJax 来获得更好的跨浏览器支持和便捷的书写体验. 诸如 Logseq、Obsidian 和 Notion 都支持使用 KaTeX 或 MathJax 渲染 LaTeX 数学代码. 不幸的是, 此类“支持”极为有限, 远不足以满足职业数学工作者的实际需求 — 因此, 此类支持的存在本身便令人费解. 本文将探讨任何旨在支持数学笔记的工具所必须满足的基本要求, 若缺乏这些要求, 则该工具便无法适用于专业用途.

A.1　数学撰写中的记号宏

数学写作通常涉及大量记号 [*notations*], 这些记号一方面难以手工排版, 另一方面往往随时间推移而发生变化. 相较手工排版的困难, 记号随时间演变的特性更为关键: 当某项数学工作中需要变更记号时, 必须同步更新所有相关出现位置; 然而, 若记号的表示形式缺乏结构化, 则工具 (如“查找 - 替换”功能) 无法可靠识别所有需更新的实例. 因此, 数学记号的表示形式必须具备结构性.

LaTeX 通过宏 [*macro*] 机制为作者提供了极为简便的结构化记号方式, 这些宏可以使用像是 `\newcommand` 或 `\NewDocumentCommand` 的指令来引入. 只需修改宏的定义, 即可自动更新所有使用该记号的位置.

遗憾的是, 大多数声称支持数学表达式嵌入的工具并未提供充分的宏支持. 尽管 KaTeX 与 MathJax 本身对宏配置具有出色支持, 但基于它们构建的多数工具 (如 Logseq、Obsidian 和 Notion) 并未开放相关配置选项. 事实上, Obsidian 社区虽存在一个插件可添加宏功能, 但仅支持为整个库 [*vault*] 设置全局宏库, 这显然不足.

A.2　记号宏应为局部而非全局之物

在 LaTeX 中, 宏通常组织为宏包 [*package*], 并在单一文档中全局导入. 由于 LaTeX 文档通常对应单一项目, 且所有嵌入 [*transclusions*] (通过 `\input` 或 `\include`) 均限于该项目内部, 该模型尚可接受 — 尽管经验丰富的 LaTeX 用户深知, 在多个宏包或文档类之间进行宏命令命名空间管理仍颇具挑战.

然而, 对于旨在长期整合多个项目的工具而言, 需求则有所不同: 用户的整个数学工作生涯中会使用大量互异的记号宏包, 不可能固定使用单一的全局记号集. 事实上, 要求用户所有笔记共享同一套记号既不合理, 亦不可行. 在许多情况下, 不同项目中的宏名甚至必然发生冲突——这可能源于项目彼此正交, 也可能源于作者个人偏好的演变. 若工具强制用户在记号变更时对成千上万条笔记进行大规模重构, 则此类工具极易因负担过重而导致大型数学项目崩溃.

[1] 译注: 原文标题为 “Requirements for typesetting mathematics”, 虽未明言, 但此处 “typesetting mathematics” 所指实为排版工具而非格式要求. 可见于 https://www.forester-notes.org/tfmt-000E/index.xml.

因此, 任何将数学记号宏限定为全局定义的思维工具 [*tool for thought*] 均无法满足数学用途. 另一方面, 亦不应要求作者在每条笔记中重复定义全部宏: 笔记通常体量较小, 而大量笔记往往构成共享相同记号的簇 [*cluster*] 在此类簇中, 记号变更仅需小范围重构, 这反而有助于增强簇内的一致性与内聚性.

综上, 对宏库支持的精确要求如下:

1. 作者必须能够 (在自有文件中) 定义多个记号宏库;
2. 每条笔记必须能够指定其所采用的宏库 (如有);

上述要求必须与数学笔记的嵌入 [*transclusions*] 机制妥善协同: 被引用的笔记应以其自身的宏库进行渲染, 而非继承父笔记的宏库.

A.3 数学图表与宏支持

科学思维工具的一项基本要求是支持数学图表的渲染. 所需图表类型自然取决于具体问题领域: 就笔者工作而言, 主要需要交换图 [*commutative diagrams*] 与弦图 [*string diagrams*].

A.3.1 数学表达式与图表紧密耦合

对非数学工作者而言, 图表绘制似乎与记号宏支持无关; 但实际上, 任何图表解决方案必须与数学表达式的渲染紧密且原生地集成——因为绝大多数图表均包含数学表达式, 而这些表达式几乎必然涉及记号宏. PGF/TikZ 之所以广受欢迎, 正是因为它尊重了这一紧密耦合.

相较 LaTeX 与 PGF/TikZ 的成熟生态, 超文本数学工具在图表支持方面仍显不足, 但仍有若干选项可供讨论.

A.3.2 KaTeX 中的交换图

KaTeX 内置了对交换图的初步支持, 通过模拟 `amscd` 宏包实现. 然而, 该支持对当代数学工作者而言完全不足:

1. 仅支持方形图结构: 实际交换图常包含对角线或曲线, 而这些均不被支持;
2. 即便是对这有限类型的支持, 其渲染效果在大多数浏览器 (至少是 Safari 和 Firefox) 中也存在严重问题: 线条由竖线与箭头拼接而成, 因细微错位而呈现锯齿状.

A.3.3 MathJax 中的交换图

与 KaTeX 类似, MathJax 亦支持 `amscd` 命令以绘制基础方形交换图. 不同于 KaTeX 的实现, MathJax 所渲染的图形线条平滑、无锯齿; 因此, 对于极少数仅需方形图的数学工作者而言, 其内置支持尚可接受.

此外, MathJax 用户还可选用更高级的方案: XyJax-v3 插件, 该插件为 MathJax 添加了完整的 `xypic` 图表支持. 值得注意的是, Stacks Project[2] 即采用此插件. 其主要缺点在于与无障碍功能

[2]译注: 按 Stack 的数学内涵, Stack Project 一般可译作 “叠计划”.

[*accessibility features*] 兼容性较差 (但与其他非平凡交换图渲染方案相比并无更劣), 且 `xypic` 所生成图表在专业观感上逊于[3] `tikz` 或 quiver.

KaTeX 与 MathJax 均具备一项关键优势: 其所渲染的图表能正确应用工具所配置的宏库. 因此, 尽管其交换图支持较为基础, 但核心需求仍得以满足. 另一值得讨论的工具是 quiver.

A.3.4 quiver 交互式图表工具

quiver 是一款优秀的交互式交换图构建工具, 提供高质量渲染. 其优点之一是允许用户加载自定义宏库, 从而确保包含自定义记号的图表在图形界面中正确显示. 但该宏库必须托管于公开可访问的 URL, 并手动粘贴至 quiver 界面.

quiver 还提供出色的 LaTeX 文档嵌入支持: 创建图表后, 可生成包含编辑链接的 LaTeX 代码片段. 例如, 以下代码对应 URL `https://q.uiver.app/?q=WzAsMixbMCwwLCJBIl0sWzEsMCwiQiJdLFswLDFdXQ==`:

```
% https://q.uiver.app/#q=WzAsMixbMCwwLCJBIl0sWzEsMCwiQiJdLFswLDFdXQ==
\begin{tikzcd}
A & B
\arrow[from=1-1, to=1-2]
\end{tikzcd}
```

然而, quiver 在 HTML 文档中的嵌入支持目前尚不足以满足专业需求. 其提供的 HTML 嵌入代码仅生成一个 `<iframe>`, 且无法对嵌入框架内部进行样式定制 (如修改背景色或减少边距[4]).

因此, 尽管 quiver 是传统 LaTeX 文档作者的优秀工具, 但目前尚不适合作为超文本数学撰写工具的组成部分.

```
<!-- https://q.uiver.app/?q=WzAsMixbMCwwLCJBIl0sWzEsMCwiQiJdLFswLDFdXQ== -->
<iframe class="quiver-embed" src="https://q.uiver.app/?q=WzAsMixbMCwwLCJBIl0sWzEsMCwiQiJdLFswLDFdXQ==&embed" width="304" height="176" style="border-radius: 8px; border: none;"></iframe>
```

鉴于 quiver 在超文本嵌入方面的不足, 我们不再将其纳入考量. 实践中最常用的替代方案是: 通过本地 LaTeX 工具链静态生成 SVG 图像.

A.3.5 使用 LaTeX 静态生成图像

鉴于其他工具普遍不足, 多数需图表支持的超文本数学作者倾向于依赖本地 LaTeX 工具链静态生成图像. 例如, 可通过 Lua 过滤器 对 pandoc 进行扩展, 将 TikZ 代码渲染为 SVG 图像.

另有若干类似工具常用于静态站点生成:

- Paolo Brasolin 开发的 antex, 被 Krater 与 jekyll-sheafy (通过 jekyll-antex) 所采用;
- 笔者开发的 Forester, 用于本网站.

[3]译注: 这有一定历史原因. 在 LaTeX 的角度看, `xypic` 要比 `tikzcd` 早出现 18 年.

[4]译注: 注意这并非技术困难, 单纯是 Jon Sterling 写此文章时, quiver 的作者 varkor 尚未加入此类功能.

此类工具的基本架构为: 扫描 LaTeX 代码块, 并以其内容的哈希值作为 `.tex` 文件名; 随后通过 `dvisvgm` 工具将 `.tex` 编译为 `.dvi`, 再转换为 `.svg`; 最终通过 `<img>` 标签嵌入 HTML. 亦可选择直接内联 `<svg>` 元素, 但需注意对其中所有标识符进行唯一重命名, 以防同一页中多个 `<svg>` 元素相互干扰.

antex 与 Forester 均支持在渲染时传入宏库. jekyll-sheafy 与 Forester 均支持按页面局部设置宏库. 该方案的主要缺点是对无障碍工具造成负面影响. 即便采用内联 `<svg>` 而非 `<img>`, 此问题亦仅略有缓解. 总体而言, 数学图表的无障碍支持仍是未解难题, 当前关于超文本数学无障碍性的讨论对此几乎未有涉及.

最后, 我们简要评述未来可能基于 Web 标准 (如 SVG 与 MathML) 的更原则性方案.

A.3.6 SVG 并非撰写语言

SVG 是一种功能强大的矢量图形底层语言, 应用广泛. 然而, 直接以 SVG 作为作者撰写语言并不现实: 与 PGF/TikZ 等程序化工具不同, SVG 中所有位置均为固定值, 无法实现关键抽象 (如 "粘附" 于两个节点之间的连线). 另一方面, SVG 亦具优势, 例如可与 MathML 等格式混合使用.

由于抽象层级过低, 当前实践中出现的 SVG 图像几乎均由高层工具或编译器自动生成.

A.3.7 MathML 并非撰写语言

尽管 Content MathML 初步支持对高层数学惯用法的结构化表示, 但 MathML 本身并非为撰写而设计, 而是作为其他工具的输出目标. 此外, Content MathML 的内容字典 [*content dictionaries*] 主要面向中小学数学需求, 完全无法满足专业数学工作者的要求:

> "基础内容元素的选择旨在覆盖美国从幼儿园至高中毕业 (乃至大学前两年, 即欧洲 A-Level 或 Baccalaureate 水平[5]) 所使用的绝大多数公式."

尽管如此, 其设计初衷是允许各 "实践社群" [*communities of practice*] 扩展内容字典以满足特定需求:

> "因此, 用户代理通常无法在缺乏上下文及实践社群信息的情况下, 自动确定 `definitionURL` 值的正确解释."

> "然而, 在需要高度精确语义的场景 (如计算机代数系统间通信、形式化系统如定理证明器内部等), 相关实践社群有责任验证、扩展或替换 OpenMath 内容字典所提供的定义."

理论上, 可 借助 XSLT 定义自定义语义记号宏, 此方向值得深入探索. 然而, 由于历史性的浏览器厂商支持薄弱, 加之与实际数学工作者社群几乎完全脱节, MathML 的高级直接使用从未普及. 尽管如此, 当前 Web 上存在大量 MathML 内容——主要作为 MathJax 与 KaTeX 的输出产物. 这些工具不

[5] 译注: 即完成学士学位水平, 注意不要与法国高中毕业会考 Baccalauréat 混淆.

仅对跨浏览器一致性与专业级渲染质量至关重要，亦因其简洁标记与便捷宏支持而成为实际撰写所必需.

MathML 的前景或较以往更为光明: Igalia 正领导一项旨在改善浏览器厂商支持的重要项目. 目前，即便支持 MathML 标准的浏览器，其渲染质量亦极不专业，这意味着即便未来厂商全面支持 MathML, MathJax 与 KaTeX 仍可能长期必要. 我们期待，随着厂商支持的改善，能催生利用 XSLT 等语义工具处理宏的新实验. 然而，鉴于数学表达式与图表撰写的紧密耦合 (见 附录 A.3.1)，除非同时开发出兼容超文本的高层图表绘制工具，否则此类转型难以实现.

A.3.8 超文本数学中 SVG 与 MathML 的融合前景

W3C MathML Core 工作草案 指出，MathML 可通过 `<foreignObject>` 元素嵌入 `<svg>` 中. 这一模块化设计极具优势，笔者相信未来可借此在超文本中实现无障碍的数学图表渲染.

当前阻碍该方法广泛应用的核心问题在于: SVG 与 MathML 均非撰写语言——其抽象层级过低，难以被作者直接使用.

只要图表仍需依赖基于 LaTeX 的工具绘制，记号宏的支持就必须基于 LaTeX 语法 (如 KaTeX 与 MathJax 所做). 然而，我们不妨设想一个未来: 数学图表通过面向 SVG 的高层接口绘制，此时纯 MathML 的记号宏方案将变得切实可行. 尽管这并非我们当前所处的世界，但仍值得期待.

時 2023 年 1 月 7 日 Jon Sterling 做此文

译者: Kokic Liu