[The Reform of Mathematical Notation]

Computation, Mathematical Notation, and Linguistics

Stephen Wolfram
Wolfram Research

Much like ordinary natural languages, most of the mathematical notation we have today has grown up over a long period of time by a kind of natural selection. Occasionally explicit efforts to systematize the notation have been made—though they have been remarkably few and far between.

In the late 1600s, Leibniz, for example, was quite concerned with mathematical notation—seeing it as an opportunity to move toward a more universal language, free of the controversies of particular ordinary languages. He invented the integral sign, the d/dx notation for derivatives (where he worried people would try to "cancel the d's"), and attacked the use of * for multiplication ("will be confused with the letter x").

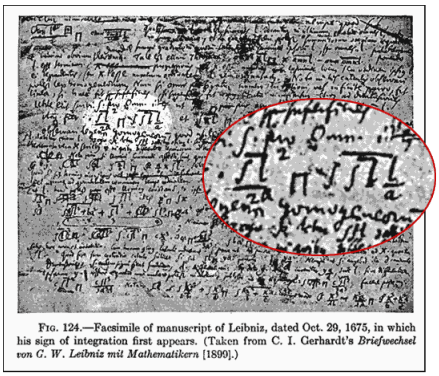


Fig. 124.—Facsimile of manuscript of Leibniz, dated Oct. 29, 1675, in which his sign of integration first appears. (Taken from C. I. Gerhardt's *Briefwechsel von G. W. Leibniz mit Mathematikern* [1899].)

Figure 1: Leibniz was serious about developing notation for math. His most famous piece of notation was invented in 1675. For integrals, he had been using "omn.", presumably standing for omnium. But on Friday October 29, 1675 he used, for the first time, the symbol that is used today.

In the 1800s Babbage wrote polemics about mathematical notation, and by the 1880s Frege, Peano and others were trying hard to create more systematic ways to represent mathematical processes. And no doubt that systematization

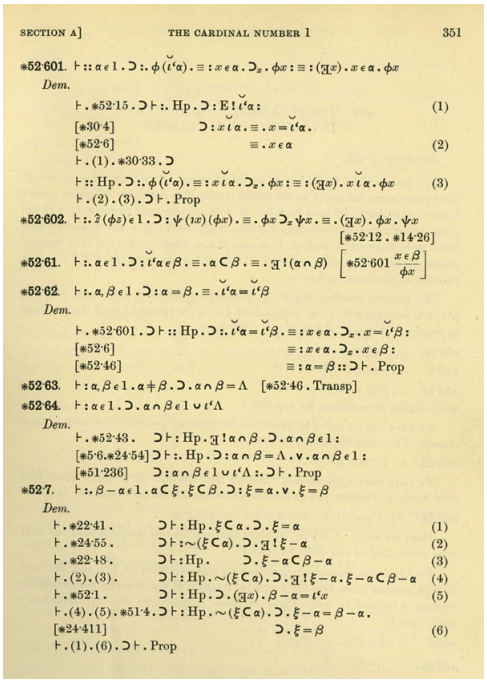
was a necessary precursor to Hilbert's program, Gödel's theorem, and ultimately Turing's own work on defining what amounts to a universal mechanism for mathematical processes.

In a sense, though, a Turing machine is a very low-level representation of mathematical processes. And I suspect Turing was curious about what would be involved in creating a higher-level representation: a full systematic language for mathematics at the level people actually do it.

As it happens, I have spent a significant part of my life developing *Mathematica*—which among other things aims to provide just such a language.

And in fact the core concept of *Mathematica* as such a language owes an important debt to the paradigm initiated by Turing's work. In the early 1900s, when people thought about systematizing mathematics, they had a definite idea about what had to be done: one had to find a way to represent mathematical proofs—as a sort of modern version of something like logical syllogisms.

And for example Whitehead and Russell in their Principia *Mathematica* developed an elaborate and arcane scheme for doing this.



SECTION A] THE CARDINAL NUMBER 1 351

*52·601. $\vdash :: \alpha \in 1 . \supset :. \phi(\breve{\iota}`\alpha) . \equiv : x \in \alpha . \supset_x . \phi x : \equiv : (\exists x) . x \in \alpha . \phi x$

Dem.

$\vdash . *52·15 . \supset \vdash :. \mathrm{Hp} . \supset : \mathrm{E} ! \breve{\iota}`\alpha :$ (1)

[*30·4] $\supset : x \breve{\iota} \alpha . \equiv . x = \breve{\iota}`\alpha .$

[*52·6] $\equiv . x \in \alpha$ (2)

$\vdash . (1) . *30·33 . \supset$

$\vdash :: \mathrm{Hp} . \supset :. \phi(\breve{\iota}`\alpha) . \equiv : x \breve{\iota} \alpha . \supset_x . \phi x : \equiv : (\exists x) . x \breve{\iota} \alpha . \phi x$ (3)

$\vdash . (2) . (3) . \supset \vdash . \mathrm{Prop}$

*52·602. $\vdash :. \hat{z}(\phi x) \in 1 . \supset : \psi(\iota x)(\phi x) . \equiv . \phi x \supset_x \psi x . \equiv . (\exists x) . \phi x . \psi x$ [*52·12 . *14·26]

*52·61. $\vdash :. \alpha \in 1 . \supset : \breve{\iota}`\alpha \in \beta . \equiv . \alpha \subset \beta . \equiv . \exists ! (\alpha \cap \beta)$ $\left[*52·601 \frac{x \in \beta}{\phi x}\right]$

*52·62. $\vdash :. \alpha, \beta \in 1 . \supset : \alpha = \beta . \equiv . \breve{\iota}`\alpha = \breve{\iota}`\beta$

Dem.

$\vdash . *52·601 . \supset \vdash :: \mathrm{Hp} . \supset :. \breve{\iota}`\alpha = \breve{\iota}`\beta . \equiv : x \in \alpha . \supset_x . x = \breve{\iota}`\beta :$

[*52·6] $\equiv : x \in \alpha . \supset_x . x \in \beta :$

[*52·46] $\equiv : \alpha = \beta :: \supset \vdash . \mathrm{Prop}$

*52·63. $\vdash : \alpha, \beta \in 1 . \alpha \neq \beta . \supset . \alpha \cap \beta = \Lambda$ [*52·46 . Transp]

*52·64. $\vdash : \alpha \in 1 . \supset . \alpha \cap \beta \in 1 \cup \iota`\Lambda$

Dem.

$\vdash . *52·43 . \quad \supset \vdash : \mathrm{Hp} . \exists ! \alpha \cap \beta . \supset . \alpha \cap \beta \in 1 :$

[*5·6.*24·54] $\supset \vdash :. \mathrm{Hp} . \supset : \alpha \cap \beta = \Lambda . \vee . \alpha \cap \beta \in 1 :$

[*51·236] $\supset : \alpha \cap \beta \in 1 \cup \iota`\Lambda :. \supset \vdash . \mathrm{Prop}$

*52·7. $\vdash :. \beta - \alpha \in 1 . \alpha \subset \xi . \xi \subset \beta . \supset : \xi = \alpha . \vee . \xi = \beta$

Dem.

$\vdash . *22·41 . \quad \supset \vdash : \mathrm{Hp} . \xi \subset \alpha . \supset . \xi = \alpha$ (1)

$\vdash . *24·55 . \quad \supset \vdash : \sim(\xi \subset \alpha) . \supset . \exists ! \xi - \alpha$ (2)

$\vdash . *22·48 . \quad \supset \vdash : \mathrm{Hp} . \quad \supset . \xi - \alpha \subset \beta - \alpha$ (3)

$\vdash . (2) . (3) . \quad \supset \vdash : \mathrm{Hp} . \sim(\xi \subset \alpha) . \supset . \exists ! \xi - \alpha . \xi - \alpha \subset \beta - \alpha$ (4)

$\vdash . *52·1 . \quad \supset \vdash : \mathrm{Hp} . \supset . (\exists x) . \beta - \alpha = \iota`x$ (5)

$\vdash . (4) . (5) . *51·4 . \supset \vdash : \mathrm{Hp} . \sim(\xi \subset \alpha) . \supset . \xi - \alpha = \beta - \alpha .$

[*24·411] $\supset . \xi = \beta$ (6)

$\vdash . (1) . (6) . \supset \vdash . \mathrm{Prop}$

Figure 2: A page from Whitehead and Russell's monumental work Principia *Mathematica* devoted to showing how the truths of mathematics could be derived from logic.

But is systematizing proofs really the only meaningful way to systematize mathematical processes? The Turing machine in a sense makes it clear that it is not. For a Turing machine provides a representation not of a proof, but of a

computation.

Of course, practical mathematics had involved computation ever since Babylonian times. But pure mathematics—following the ideas of Euclid, and later of logic—had concentrated instead on proof. The concept of a Turing machine connected pure mathematics to computation in a systematic and universal way.

And when I came to develop *Mathematica*, I did so within the paradigm of computation rather than proof.

Mathematica represents mathematics in an actionable way: its purpose is not to show, or find, the steps in proofs, but rather to find results, and find what is true, by explicitly computing output from input.

As a direct consequence of universal computation, *Mathematica* can internally represent any possible computation. But then the challenge—as Turing in effect recognized—is to connect those possible computations to ones that humans can describe.

I have spent more than three decades designing languages—most importantly *Mathematica*—that allow computations to be specified conveniently. And in a sense the way I have worked is to try to imagine all the computations that people might want to do, and then to identify repeated chunks of computational work that occur in those—and then to give names to those chunks.

The result—if one succeeds—is an artificial language in which typical computations and programs can be expressed in the shortest and clearest possible way. And indeed, after countless millions of lines of *Mathematica* language have been written, I believe I can claim a certain degree of success.

But what about traditional mathematics? How can we represent it, as Turing wondered, in a systematic way?

If one is going to be able to automate mathematical computations, then ultimately one has to have a precise and systematic representation of the mathematics.

And with all the precision traditional in pure mathematics, one might imagine that its notation would somehow have evolved to a high degree of precision. But it has not. Traditional mathematical notation is full of implicit conventions, strange elisions and historical accidents.

In designing the mathematical components of the *Mathematica* language [3], however, I had to create a systematic form of the notation. But to make *Mathematica* easy for humans to learn and understand, I wanted to stay as close as possible to traditional notation.

The result is that I undertook an extensive study of the way that mathematical notation is used in practice. In a sense, this study was similar in character to the way a linguist might try to infer the grammar and syntax of some ordinary spoken human language. But the literature of mathematics provides a somewhat more systematic corpus than is usually available.

And somewhat to my surprise, despite the diversity of the mathematical literature, there was a remarkable degree of consistency in the way notation tended to be used—down even to consistent unwritten conventions about the precedence of all sorts of mathematical operators.

And it took only a modest set of innovations to go from this notation to something completely precise and computable. (It helped that *Mathematica* can support not just linear textual input, but also full two-dimensional input, like traditional mathematical notation.)

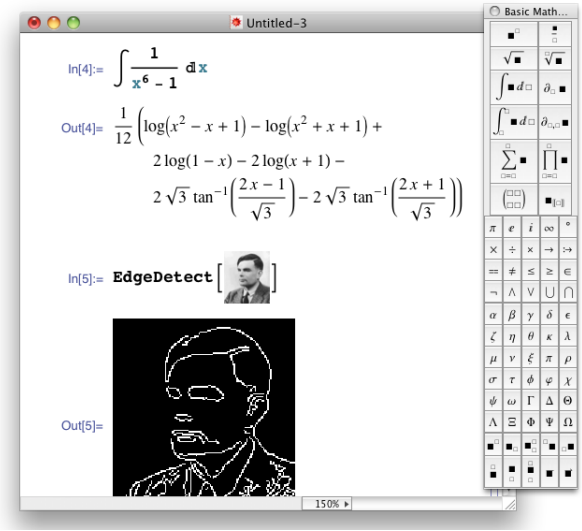


Figure 3: Mathematical and other notation in *Mathematica*. Note the two-dimensional character of the input.

A great deal of mathematics has now been described in the precise notation of *Mathematica* [1].

But a few years ago I became curious about the extent to which it would be possible to handle by computer completely free-form mathematical notation and input.

For in developing Wolfram|Alpha [4], my goal was to allow people to specify their queries—whether about mathematics or anything else—just in the way that they think of them, without having to convert them to any kind of precise formal language.

At first, it seemed as if this kind of free-form linguistic input might simply be impossible, or impractical. But thanks to a series of breakthroughs, we have been able to make this work in a highly successful way.

Indeed, when it comes to textually typed mathematical input we can now recognize what a person meant in very close to 100% of all cases—at least those that a trained human would find recognizable. Of course, it helps that we have been able to study many, many millions of inputs that have been fed to Wolfram|Alpha.

And among the results of this is that we can say with some precision the extent to which people do or do not use the various notational conventions that Turing describes in his notes.

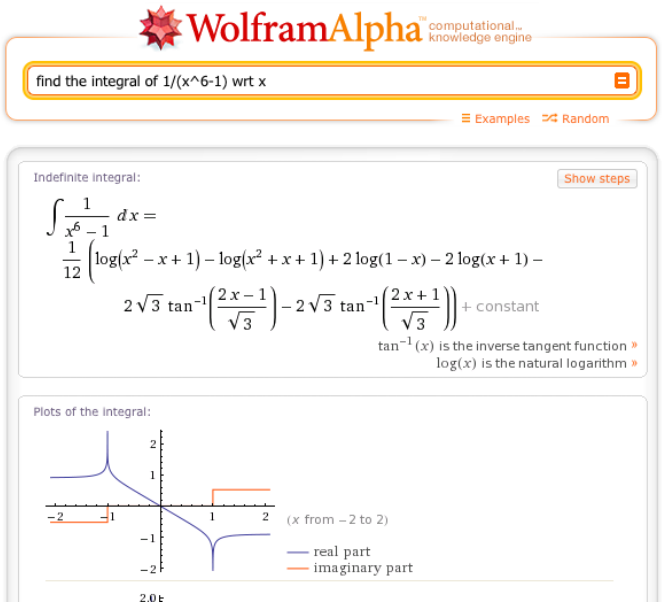


Figure 4: Wolfram|Alpha understands free-form natural language specifications of mathematical operations.

In *Mathematica* we try to do what Turing advocates: to create a completely systematic and precise notation for mathematics. And indeed this is a very powerful thing. But in Wolfram|Alpha we have now succeeded in doing something that is in a sense maximally convenient for humans: just taking mathematical notation in the form that humans think of it, and interpreting it into a precise computable form.

I rather suspect—and hope—that Turing would appreciate the notation of *Mathematica*—set up as it is to provide a precise and unambiguous representation that can immediately be computed with.

And perhaps he would be surprised—as I was—that it is possible in Wolfram|Alpha to go from the strange and inconsistent notation that has grown up in mathematics, and in a sense use the sparsity of typical mathematical questions to be able to deduce what the corresponding precise notation should be.

In his work at the dawn of systematic computation, Turing could only begin to imagine what it would be like to make mathematics computational. Today—especially with *Mathematica* and Wolfram|Alpha—we have succeeded in making large swaths of mathematics computational.

One issue that has remained is the style of mathematics traditional in the 20th century, which centers around the creation of mathematical structures ("Let F be a field ..."). A recent realization is that the basic paradigm of Wolfram|Alpha is exactly what is needed to make such mathematics computational.

In Wolfram|Alpha, it is common to enter some entity (say a city or a chemical), and then have Wolfram|Alpha automatically generate a report on what might be considered "interesting" about that entity. The same can be done for mathematical structures.

In effect, Wolfram|Alpha must take the structure, and then deduce what facts or theorems are "interesting" about it. In part, this can be done from a curation of known mathematical theorems. In part, it must be done by a collection of mathematical and meta-mathematical algorithms and heuristics. But the result, I believe, will be that the vast majority of the parts the human activity that we call "mathematics" will successfully be completely automated. The concept of systematization—and computation—that Turing had will have been realized.

And, as it happens, thanks to the likes of Wolfram|Alpha technology, there won't even be a need to "reform" mathematical notation in order for humans to successfully describe what they want to do.

References

[1] `http://www.wolfram.com/mathematica/`

[2] Wolfram, S. "Mathematical Notation: Past and Future", Transcript of a keynote address presented at *MathML and Math on the Web: MathML International Conference*, October 20, 2000. `http://www.stephenwolfram.com/publications/recent/mathml/mathml2.html`.

[3] Wolfram, S. "The Poetry of Function Naming", blog post, October 18, 2010. `http://blog.stephenwolfram.com/2010/10/the-poetry-of-function-naming/`

[4] `http://www.wolframalpha.com/`