Network Working Group
Request for Comments: 1019

D. Arnon
Xerox PARC

September 1987

Report of the Workshop on Environments for Computational Mathematics
July 30, 1987
ACM SIGGRAPH Conference
Anaheim Convention Center, Anaheim, California

Status of This Memo

This memo is a report on the discussion of the representation of equations in a workshop at the ACM SIGGRAPH Conference held in Anaheim, California on 30 July 1987. Distribution of this memo is unlimited.

Introduction

Since the 1950's, many researchers have worked to realize the vision of natural and powerful computer systems for interactive mathematical work. Nowadays this vision can be expressed as the goal of an integrated system for symbolic, numerical, graphical, and documentational mathematical work. Recently the development of personal computers (with high resolution screens, window systems, and mice), high-speed networks, electronic mail, and electronic publishing, have created a technological base that is more than adequate for the realization of such systems. However, the growth of separate Mathematical Typesetting, Multimedia Electronic Mail, Numerical Computation, and Computer Algebra communities, each with its own conventions, threatens to prevent these systems from being built.

To be specific, little thought has been given to unifying the different expression representations currently used in the different communities. This must take place if there is to be interchange of mathematical expressions among Document, Display, and Computation systems. Also, tools that are wanted in several communities (e.g., WYSIWYG mathematical expression editors), are being built independently by each, with little awareness of the duplication of effort that thereby occurs. Worst of all, the ample opportunities for cross-fertilization among the different communities are not being exploited. For example, some Computer Algebra systems explicitly associate a type with a mathematical expression (e.g., 3 x 3 matrix of polynomials with complex number coefficients), which could enable automated math proofreaders, analogous to spelling checkers.

The goal of the Workshop on Environments for Computational Mathematics was to open a dialogue among representatives of the

Computer Algebra, Numerical Computation, Multimedia Electronic Mail, and Mathematical Typesetting communities. In July 1986, during the Computers and Mathematics Conference at Stanford University, a subset of this year's participants met at Xerox PARC to discuss User Interfaces for Computer Algebra Systems. This group agreed to hold future meetings, of which the present Workshop is the first. Alan Katz's recent essay, "Issues in Defining an Equations Representation Standard", RFC-1003, DDN Network Information Center, March 1987 (reprinted in the ACM SIGSAM Bulletin May 1987, pp. 19-24), influenced the discussion at the Workshop, especially since it discusses the interchange of mathematical expressions.

This report does not aim to be a transcript of the Workshop, but rather tries to extract the major points upon which (in the Editor's view) rough consensus was reached. It is the Editor's view that the Workshop discussion can be summarized in the form of a basic architecture for "Standard Mathematical Systems", presented in Section II below. Meeting participants seemed to agree that: (1) existing mathematical systems should be augmented or modified to conform to this architecture, and (2) future systems should be built in accordance with it.

The Talks and Panel-Audience discussions at the Workshop were videotaped. Currently, these tapes are being edited for submission to the SIGGRAPH Video Review, to form a "Video Proceedings". If accepted by SIGGRAPH, the Video Proceedings will be publicly available for a nominal distribution charge.

One aspect of the mathematical systems vision that we explicitly left out of this Workshop is the question of "intelligence" in mathematical systems. This has been a powerful motivation to systems builders since the early days. Despite its importance, we do not expect intelligent behavior in mathematical systems to be realized in the short term, and so we leave it aside. Computer Assisted Instruction for mathematics also lies beyond the scope of the Workshop. And although it might have been appropriate to invite representatives of the Spreadsheets and Graphics communities, we did not. Many of those who were at the Workshop have given considerable thought to Spreadsheets and Graphics in mathematical systems.

Financial support from the Xerox Corporation for AudioVisual equipment rental at SIGGRAPH is gratefully acknowledged. Thanks are due to Kevin McIsaac for serving as chief cameraman, providing critical comments on this report, and contributing in diverse other ways to the Workshop. Thanks also to Richard Fateman, Michael Spivak, and Neil Soiffer for critical comments on this report. Subhana Menis and Erin Foley have helped with logistics and documentation at several points along the way.

Information on the Video Proceedings, and any other aspect of the Workshop can be obtained from the author of this report.

I. Particulars of the meeting

The Workshop had four parts: (1) Talks, (2) Panel Discussion, (3) Panel and Audience discussion, (4) and Live demos. Only a few of the systems presented in the talks were demonstrated live. However, many of the talks contained videotapes of the systems being discussed.

The talks, each 15 minutes in length, were:

1. "The MathCad System: a Graphical Interface for Computer Mathematics", Richard Smaby, MathSOFT Inc.

2. "MATLAB - an Interactive Matrix Laboratory", Cleve Moler, MathWorks Inc.

3. "Milo: A Macintosh System for Students", Ron Avitzur, Free Lance Developer, Palo Alto, CA.

4. "MathScribe: A User Interface for Computer Algebra systems", Neil Soiffer, Tektronix Labs.

5. "INFOR: an Interactive WYSIWYG System for Technical Text", William Schelter, University of Texas.

6. "Iris User Interface for Computer Algebra Systems", Benton Leong, University of Waterloo.

7. "CaminoReal: A Direct Manipulation Style User Interface for Mathematical Software", Dennis Arnon, Xerox PARC.

8. "Domain-Driven Expression Display in Scratchpad II", Stephen Watt, IBM Yorktown Heights.

9. "Internal and External Representations of Valid Mathematical Reasoning", Tryg Ager, Stanford University.

10. "Presentation and Interchange of Mathematical Expressions in the Andrew System", Maria Wadlow, Carnegie-Mellon University.

The Panel discussion lasted 45 minutes. The panelists were:

Richard Fateman, University of California at Berkeley
Richard Jenks, IBM Yorktown Heights
Michael Spivak, Personal TeX
Ronald Whitney, American Mathematical Society

The panelists were asked to consider the following issues in planning their presentations:

1. Should we try to build integrated documentation/computation systems?

2. WYSIWYG editing of mathematical expressions.

3. Interchange representation of mathematics.

4. User interface design for integrated documentation/computation systems.

5. Coping with large mathematical expressions.

A Panel-Audience discussion lasted another 45 minutes, and the Demos lasted about one hour.

Other Workshop participants, besides those named above, included:

S. Kamal Abdali, Tektronix Labs
George Allen, Design Science
Alan Katz, Information Sciences Institute
J. Robert Cooke, Cornell University and Cooke Publications
Larry Lesser, Inference Corporation
Tom Libert, University of Michigan
Kevin McIsaac, Xerox PARC and University of Western Australia
Elizabeth Ralston, Inference Corporation

II. Standard Mathematical Systems - a Proposed Architecture

We postulate that there is an "Abstract Syntax" for any mathematical expression. A piece of Abstract Syntax consists of an Operator and an (ordered) list of Arguments, where each Argument is (recursively) a piece of Abstract Syntax. Functional Notation, Lisp SExpressions, Directed Acyclic Graphs, and N-ary Trees are equivalent representations of Abstract Syntax, in the sense of being equally expressive, although one or another might be considered preferable from the standpoint of computation and algorithms. For example, the functional expression "Plus[Times[a,b],c]" represents the Abstract Syntax of an expression that would commonly be written "a*b+c".

A "Standard Mathematical Component" (abbreviated SMC) is a collection of software and hardware modules, with a single function, which if it reads mathematical expressions, reads them as Abstract Syntax, and if it writes mathematical expressions, writes them as Abstract Syntax. A "Standard Mathematical System" (abbreviated SMS) is a collection of SMC’s which are used together, and which communicate with each other in Abstract Syntax.

We identify at least four possible types of components in an SMS.

Any particular SMS may have zero, one, or several instances of each component type. The connection between two particular components of an SMS, of whatever type, is via Abstract Syntax passed over a "wire" joining them.

1) EDs - Math Editors

These edit Abstract Syntax to Abstract Syntax. A particular system may have editors that work on some other representations of mathematics (e.g., bitmaps, or particular formatting languages), however they do not qualify as an ED components of a SMS. An ED may be WYSIWYG or language-oriented.

2) DISPs - Math Displayers

These are suites of software packages, device drivers, and hardware devices that take in an expr in Abstract Syntax and render it. For example, (1) the combination of an Abstract Syntax->TeX translator, TeX itself, and a printer, or (2) a plotting package plus a plotting device. A DISP component may or may not support "pointing" (i.e., selection), within an expression it has displayed, fix a printer probably doesn't, but terminal screen may. If pointing is supported, then a DISP component must be able to pass back the selected subexpression(s) in Abstract Syntax. We are not attempting here to foresee, or limit, the selection mechanisms that different DISPs may offer, but only to require that a DISP be able to communicate its selections in Abstract Syntax.

3) COMPs - Computation systems

Examples are Numerical Libraries and Computer Algebra systems. There are questions as to the state of a COMP component at the time it receives an expression. For example, what global flags are set, or what previous expressions have been computed that the current expression may refer to. However, we don't delve into these hard issues at this time.

4) DOCs - Document systems

These are what would typically called "text editors", "document editors", or "electronic mail systems". We are interested in their handling of math expressions. In reality, they manage other document constituents as well (e.g., text and graphics). The design of the user interface for the interaction of math, text, and graphics is a nontrivial problem, and will doubtless be the subject of further research.

A typical SMS will have an ED and a DISP that are much more closely coupled than is suggested here. For example, the ED's internal representation of Abstract Syntax, and the DISP's internal representation (e.g., a tree of boxes), may have pointers back and

forth, or perhaps may even share a common data structure. This is acceptable, but it should always be possible to access the two components in the canonical, decoupled way. For example, the ED should be able to receive a standard Abstract Syntax representation for an expression, plus an editing command in Abstract Syntax (e.g., Edit[expr, cmd]), and return an Abstract Syntax representation for the result. Similarly, the DISP should be able to receive Abstract Syntax over the wire and display it, and if it supports pointing, be able to return selected subexpressions in Abstract Syntax.

The boundaries between the component types are not hard and fast. For example, an ED might support simple computations (e.g., simplification, rearrangement of subexpressions, arithmetic), or a DOC might contain a facility for displaying mathematical expressions. The key thing for a given module to qualify as an SMC is its ability to read and write Abstract Syntax.

III. Recommendations and Qualifications

1. It is our hypothesis that it will be feasible to encode a rich variety of other languages in Abstract Syntax, for example, programming constructs. Thus we intend it to be possible to pass such things as Lisp formatting programs, plot programs, TeX macros, etc. over the wire in Abstract Syntax. We also hypothesize that it will be possible to encode all present and future mathematical notations in Abstract Syntax (e.g., commutative diagrams in two or three dimensions). For example, the 3 x 3 identify matrix might be encoded as:

   Matrix[[1,0,0], [0,1,0], [0,0,1]]

   while the Abstract Syntax expression:

   Matrix[5, 5, DiagonalRow[1, ThreeDots[], 1],
   BelowDiagonalTriangle[FlexZero[]],
   AboveDiagonalTriangle[FlexZero[]]]

   might encode a 5 x 5 matrix which is to be displayed with a "1" in the (1,1) position, a "1" in the (5,5) position, three dots between them on the diagonal, a big fat zero in the lower triangle indicating the presence of zeros there, and a big fat zero in the upper triangle indicating zeros.

2. We assume the use of the ASCII character set for Abstract Syntax expressions. Greek letters, for example, would need to be encoded with expressions like Greek[alpha], or Alpha[]. Similarly, font encoding is achieved by the use of Abstract Syntax such as the following for 12pt bold Times Roman: Font[timesRoman, 12, bold, <expression>] Two SMCs are free to communicate in a larger character set, or pass font specifications in other ways, but they should always be able to

express themselves in standard Abstract Syntax.

3. COMPs (e.g., Computer Algebra systems), should be able to communicate in Abstract Syntax. Existing systems should have translators to/from Abstract Syntax added to them. In addition, if we can establish a collection of standard names and argument lists for common functions, and get all COMP's to read and write them, then any Computer Algebra system will be able to talk to any other. Some examples of possible standard names and argument lists for common functions:

```
Plus[a,b,...]
Minus[a]
Minus[a,b]
Times[a,b,...]
Divide[<numerator>, <denominator>]
Power[<base>, <exponent>]
PartialDerivative[<expr>, <var>]
Integral[<expr>, <var>, <lowerLimit>,<upperLimit>] (limits optional)
Summation[<<summand>, <lowerLimit>, <upperLimit>] (limits optional)
```

A particular algebra system may read and write nonstandard Abstract Syntax. For example:

```
Polynomial[Variables[x, y, z], List[Term[coeff, xExp, yExp, zExp],
...
```

but, it should be able to translate this to an equivalent standard representation. For example:

```
Plus[Times[coeff, Power[x, xExp], ...
```

4. A DOC must store the Abstract Syntax representations of the expressions it contains. Thus it's easy for it to pass its expressions to EDs, COMPs, or DISPs. A DOC is free to store additional expression representations. For example, a tree of Boxes, a bitmap, or a TeX description.

5. DISPs will typically have local databases of formatting information. To actually render the Abstract Syntax, the DISP checks for display rules in its database. If none are found, it paints the Abstract Syntax in some standard way. Local formatting databases can be overridden by formatting rules passed over the wire, expressed in Abstract Syntax. It is formatting databases that store knowledge of particular display environments (for e.g., "typesetting for Journal X").

   The paradigm we wish to follow is that of the genetic code: A mathematical expression is like a particular instance of DNA, and upon receiving it a DISP consults the appropriate formatting database to see if it understands it. If not, the DISP just

"passed it through unchanged". The expression sent over the wire may be accompanied by directives or explanatory information, which again may or may not be meaningful to a particular DISP. In reality, formatting databases may need to contain Expert System-level sophistication to be able to produce professional quality typesetting results, but we believe that useful results can be achieved even without such sophistication.

6. With the use of the SMC's specified above, it becomes easy to use any DOC as a logging facility for a session with a COMP. Therefore, improvements in DOCs (e.g., browsers, level structuring, active documents, audit trails), will automatically give us better logging mechanisms for sessions with algebra systems.

7. Note that Abstract Syntax is human-readable. Thus any text editor can be used as an ED. Of course, in a typical SMS, users should have no need to look at the Abstract Syntax flowing through the internal "wires" if they don't care to. Many will want to interact only with mathematics that has a textbook-like appearance, and they should be able to do so.

8. Alan Katz's RFC (cited above) distinguishes the form (i.e., appearance) of a mathematical expression from its content (i.e., meaning, value). We do not agree that such a distinction can be made. We claim that Abstract Syntax can convey form, meaning, or both, and that its interpretation is strictly in the eye of the beholder(s). Meaning is just a handshake between sender and recipient.

9. Help and status queries, the replies to help and status queries, and error messages should be read and written by SMC's in Abstract Syntax.

10. In general, it is permissible for two SMC's to use private protocols for communication. Our example of a tightly coupled ED and DISP above is one example. Two instances of a Macsyma COMP would be another; they might agree to pass Macsyma internal representations back and forth. To qualify as SMC's, however, they should be able to translate all such exchanges into equivalent exchanges in Abstract Syntax.