COMMUNICATING MATHEMATICS: USEFUL IDEAS FROM COMPUTER SCIENCE

By Charles Wells

Published by the *American Mathematical Monthly*, Vol. 102, May, 1995

UPDATE 21 NOVEMBER 2013

I have written extensively on this subject in the last 18 years. These are available on line:

Abstractmath, a website

Gyre&Gimble, a blog

"Communicating Logical Reasoning" , by Atish Bagchi and Charles Wells

"Varieties of Mathematical Prose" by Atish Bagchi and Charles Wells

--Charles Wells

Current contact information:

charles@abstractmath.org

Reprinted from the AMERICAN MATHEMATICAL MONTHLY
Vol. 102, No. 5, May 1995

Communicating Mathematics: Useful Ideas from Computer Science

Charles Wells

1. INTRODUCTION

1.1. Purpose. This article describes certain ideas originating in the theory and practice of computer science, and shows how the teaching and exposition of mathematics could benefit if these ideas were widely understood by mathematicians and used in their teaching and writing.

These ideas are discussed here because I believe they are important for mathematicians to understand. Some of them are based on theoretical work by computer scientists and others are based on the practice of computer professionals inside and outside academia. Computer scientists would not regard the various concepts as of equal importance, and the whole collection of ideas is nothing like a fair presentation of the current state of computing.

2. SPECIFICATION

2.1. External behavior. A programmer writing a large program may have a tentative conception of how to write the program in terms of subprograms that perform specific tasks. For example, a program for factoring large integers might use a function `PrimeQ:Z → {True, False}` with the property that `PrimeQ` $[n]$ returns `True` if the integer n is prime and `False` otherwise. This description gives the function's *external behavior*[1] but says nothing about how that behavior is implemented. Perhaps the first implementation of `PrimeQ` $[n]$ will test whether any integer k for which $1 < k < \sqrt{|n|}$ divides n. This might be enough for debugging the program that uses `PrimeQ`, but not fast enough for practical use. Later, an implementation using modern fast techniques could be substituted. Since the *external behavior* of the function `PrimeQ` is the same in either implementation, substituting the new implementation for the old should not introduce new bugs in the program.

In this section, I discuss some issues involved in presenting mathematics to students that can be clarified by this idea of specifying external behavior.

2.2. Is everything a set? One concern of those who study the foundations of mathematics is to show how to develop the main body of modern mathematics from a small number of principles or concepts that are as clear and primitive as possible. The bulk of the work in this direction has been to reduce all mathematical constructions to the primitive notion of "set" and "element of a set" and to

[1] Many practicing programmers call this its "functional behavior" but I would avoid that phrase in teaching mathematics students because of confusion with the function concept. My thanks to the referee for suggesting the name "external behavior".

impose a small number of clear axioms on these primitives.[2] In carrying this program out, an ordered pair $\langle a, b \rangle$ is typically interpreted as the set $\{\{a, b\}, \{b\}\}$, and a function as a set of ordered pairs with the functional property.

Many mathematicians have taken this approach to mean that every mathematical object is *really* a complicated set. At least, they say this when the topic of foundations comes up. They often don't behave as if they actually believe every mathematical object is a set. Wouldn't you expect a mathematician to be confused, at least momentarily, if you asked which points in the plane had nonempty intersection with the point $\langle 3, 2 \rangle$?[3] I maintain that *in practice* many mathematicians regard points as one type of mathematical object, sets as another, and perhaps functions as a third. Since intersection is an operation defined on sets and points are not sets, the question about which points have nonempty intersection with $\langle 3, 2 \rangle$ is to be rejected as meaningless.

The best way to think of the reduction to sets that has been carried out by those who study foundations is that it is a *representation* of mathematics which is desirable for various reasons, for example showing consistency. Although an ordered pair is not a set, it can be represented as a set and that representation may be useful for certain purposes.

2.3. Specification in exposition. My thesis in this section is that we should borrow the idea of specifying external behavior and use *specifications* rather than *definitions* of many common mathematical objects in a way that will exhibit how the objects relate to other types of objects. The formal definition of a concept may require the mention of details of representations used for other purposes (such as consistency proofs) that obscure the way the concept is used in practice. In courses for undergraduates other than in foundations, we should not even attempt to say what sets, pairs, functions and other basic mathematical objects "really are". What matters is how they relate to other objects.

Recommendation. *In elementary exposition, explain a basic concept by giving a* specification *of the concept—a carefully written description of the interaction of the object with other mathematical objects.*[4]

Here are two examples based on my text [Wells, 1993], which is aimed at students who have had calculus but no course in abstract mathematics. In these specifications, I use the word "object" to denote any sort of mathematical entity.

> **Ordered pairs:** An *ordered pair* is a mathematical object which is distinct from but completely determined by objects called its *first coordinate* and its *second coordinate*. The ordered pair with first coordinate x and second coordinate y is denoted by $\langle x, y \rangle$.

[2] Foundations can also be done using category theory [McLarty, 1993].

[3] This is discussed from a different point of view in [Barr, 1993].

[4] The word "specification" in computer science generally means a description in a formal language of the external behavior of a program, suitable for being transformed by strict rules into an actual program. In this document, the analogy is more with the informal descriptions practicing programmers give of the external behavior of a program, as described in Section 2.1.

It follows that ordered pairs are the same if and only if their coordinates are the same, that is,

$$(\langle x, y\rangle = \langle x', y'\rangle) \quad \Leftrightarrow \quad (x = x' \text{ and } y = y').$$

Thus we have a *method of proof*: To prove two ordered pairs $\langle x, y\rangle$ and $\langle x', y'\rangle$ are the same, prove that $x = x'$ and $y = y'$.

Functions: A function F is a mathematical object which determines and is completely determined by the following data:

F.1 F has a *domain*, which is a set and is denoted by dom F.
F.2 F has a *codomain*, which is also a set and is denoted by cod F.
F.3 For each element $x \in \text{dom } F$, F has a *value* at x. This value is completely determined by x and F and must be an element of cod F. The value of F at x is denoted by $F(x)$.

I am sure these specifications could be improved in various ways and would welcome suggestions concerning them. There may be a better name than "specification" for the practice, too, but it seems clear that the practice should have an explicit name to signal its logical status.

3. SYNTAX AND SEMANTICS. There is a sense in which $2/(4 + 3)$ is $2/7$ and another sense in which $2/(4 + 3)$ is not $2/7$. The *number* $2/(4 + 3)$ is indeed the same number as $2/7$. The *expression* $2/(4 + 3)$ is not the same as the expression $2/7$. For one thing, the expression $2/(4 + 3)$ has seven symbols and the expression $2/7$ has only three.

Syntax is the study of expressions in linguistics or computer science, and *semantics* is the study of how meaning is assigned to expressions. There are two different points to make concerning syntax and semantics: (a) Expressions represent mathematical objects but are not the objects themselves, and (b) expressions have structure.

3.1. Expressions and their denotations. A mathematical expression denotes a mathematical object. The object it denotes is not the expression—the expression is only a representation of the object. In particular, different expressions may denote the same object.

This point of view, that there is an object independent of the expressions that denote it, is often called "Platonist"[5]. By contrast, some assert that the expressions are merely themselves ("everything is syntax") and that mathematics consist of manipulating these expressions according to precise rules. Presumably, those who hold that attitude will admit that there is an equivalence relation on expressions which identifies different expressions that name the same object from a Platonist's point of view. In any case, people who hold these differing points of view generally agree on which statements are theorems.

It is my observation that students and teachers at the college level in the USA don't communicate well with each other because many teachers talk like Platonists,

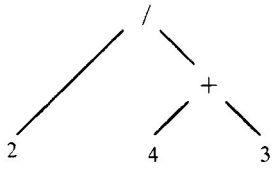
[5]This usage of the word "Platonist" does not imply an endorsement of *all* of Plato's attitudes toward reality and truth.

but many students have the attitude[6] that what they need are rules to manipulate the expressions (more about this in 3.3 below). Students who go on to higher mathematics learn to talk as if the mathematical objects were "out there", but it is noticeable that many college freshmen in calculus courses do not talk that way.

Recommendation. *Teachers and authors of textbooks should make the distinction between syntax and semantics explicit.*

If the student has words for this distinction, he or she may avoid certain types of confusion that result from being unaware of the distinction.

3.2. Parsing. Computer science is intimately concerned with the relationship between syntax and semantics, particularly with *parsing*, which is the explication of the abstract structure of an expression such as $2/(4 + 3)$. This structure is often given as a tree



(1)

To *parse* an expression such as $2/(4 + 3)$ is to exhibit its structure. The first task a compiler for a computer language has is to parse the commands of the language, for only then can the commands be executed.

Laborde [1990] notes that many students (these were mostly below the USA freshman level) see expressions such as $2/(4 + 3)$ merely as strings and are not really aware of their abstract structure.

Recommendation. *Introduce informal parsing of mathematical expressions as a learning tool.*

3.3. Mathematics as syntax. Another point of view is that we should *stop* talking like Platonists and go along with the students' desire for rules for manipulation. Some hold that mathematics is a game of syntax, and that to succeed in mathematics you must master the rules of the game (more about this in Section 4.3). You need not hold that view, however, to realize that a lot of mathematics is accomplished precisely by syntactic manipulation—what else is high school algebra? And even Platonists agree that you have to master the rules of the game.

Recommendation. *Make explicit the allowable syntax for statements about a type of object.*

For example, one can helpfully explain application of functions by adding the following sentence to the specification of function in Section 2.3:

> The expression "$F(x)$" is meaningful if and only if "$x \in \text{dom } F$" is true, and in that case "$F(x) \in \text{cod } F$" is true.

[6]Usually, this attitude is unexpressed. I am saying this out of my observations of students rather than what they actually say.

4. FORMAL TRANSFORMATIONS. Formal transformations, called *rewrite rules* in many contexts, are used in many different ways in computing. In general, they work this way: In an expression, you recognize a subexpression as matching the pattern of the left side of a transformation rule, and you rewrite the expression, replacing the subexpression by the right side of the rule. For example, in algebra you may rewrite $a + bx + by$ as $a + b(x + y)$. Computing an integral in freshman calculus can be thought of as recognizing patterns and applying formal transformations, although there may be several possible transformations to apply and the process need not terminate.

Sometimes rewrite rules are applicable to any occurrence of the suitable pattern (they are "context free") and at other times they depend on specific conditions ("context sensitive"). A notorious example of the latter is L'Hôpital's Rule. Students resist constraints of this sort [Maurer, 1987].

4.1. Definitions as macros. Most implementations of the C language allow the user to define *macros* that a preprocessor converts into standard C commands. For example, you might want to limit the number of times a program will repeat some action. By writing `#define maxit 20` you defined the macro `maxit` to have the value 20; the preprocessor will replace the word `maxit` with `20` everywhere it appears in the source program. Later, after you have debugged the program, you could change `maxit` to some much larger number and recompile. In general, in contrast to this example, macros can have parameters.

Mathematical definitions play the role of macros in the context of proofs. A colleague of mine in computer science who majored in mathematics as an undergraduate has described how as a student he suddenly caught on that he could do at least B work in most math courses by merely rewriting the definitions of the terms involved in the questions and making a few obvious deductions.

Recommendation. *Encourage students to begin proving a theorem by replacing (some or all of) the words that have definitions with the text of their definitions.*

4.2. Proof by rewriting. Gries [1991], Dijkstra and Scholten [1990] and others have urged that *proofs* be done by applying formal transformations. Many examples may be found in [van Gasteren, 1990] and [Gries and Schneider, 1993]. Proofs are not often done that way by mathematicians except in teaching formal logic. The preferred method is more often the "semantic" approach: the proof proceeds via an understanding of the objects involved rather than by the formal application of rules. Gries, and Dijkstra and Scholten, urge a syntactic approach: express the desired result in a formal language and apply meaning-preserving transformations to it until it becomes a consequence of a known fact. This turns a *proof* into a type of *computation*.

I will now give two proofs of a simple statement about the ordering of the real numbers drawn from Gries [1991]. The statement to prove is

$$(x > z) \Rightarrow ((x > y) \vee (y > z)). \tag{2}$$

4.2.1. *Semantic proof.* The way I proved (2) when I first saw it was to envision x and z as points on the line placed this way:

There are three different regions into which we can place y. In the right two, $y > z$ and in the left two, $x > y$. End of proof.

This proof is written in English, not in symbolic notation, and it refers to a particular mental representation of the structure in question (the usual ordering of the real numbers).

4.2.2. *Syntactic Proof*. This proof is due to David Gries (private communication). It is based on these principles:

P.1 (Contrapositive) The equivalence of $P \Rightarrow Q$ and $\neg Q \Rightarrow \neg P$.
P.2 (DeMorgan) The equivalence of $\neg(P \vee Q)$ and $\neg P \wedge \neg Q$.
P.3 The equivalence in any totally ordered set of $\neg(x > y)$ and $x \leq y$.

Proof:

$$(x > z) \Rightarrow ((x > y) \vee (y > z))$$
$$\Leftrightarrow \quad \text{by P.1}$$
$$\neg((x > y) \vee (y > z)) \Rightarrow \neg(x > z)$$
$$\Leftrightarrow \quad \text{by P.2}$$
$$(\neg(x > y) \wedge \neg(y > z)) \Rightarrow \neg(x > z)$$
$$\Leftrightarrow \quad \text{by P.3 three times}$$
$$((x \leq y) \wedge (y \leq z)) \Rightarrow (x \leq z)$$

which is true by the transitive law.

4.2.3. *About the syntactic proof*. There are many advantages to the technique illustrated by the second proof. It holds in any totally ordered set, not just in the real numbers. Each instance of the application of a transformation can be mechanically checked to see that it is correctly applied. (Ingenuity, of course, is still required to *create* the proof.) This mechanical verifiability is certainly not true in the case of the usual mathematical proof; it is notorious that if you read a proof written by someone whose mental representation of the concepts is very different from yours, the proof is next to impossible to follow.

4.2.4. *About the semantic proof*. There are several arguments for the semantic proof. For one thing, many mathematicians prefer the proofs to be written out in English sentences rather than in the symbolic notation of 4.2.2. This view is advocated in the works on mathematical writing by Halmos [1975] (page 42), Steenrod [1975] (page 57), Gillman [1987] (page 15) and Boas [1981]. Another point is that it is easy to make mistakes in checking the application of transformations, particularly when the patterns that must match are complex.

However, the major argument for semantic proofs concerns mental representations.

4.3. Mental representations. Several mathematicians who read the syntactic proof in 4.2.2 in an earlier version of this paper expressed dissatisfaction with it as compared to the pictorial proof preceding it. One objection they gave is that the pictorial proof helps them to *understand* the theorem. I believe that when they say that, they mean they want a *mental representation* of the object involved in the theorem that makes the truth of the theorem obvious or easy to understand.[7] A

[7] I do not claim that making the truth of the theorem obvious constitutes proof of the theorem. Argumentative philosophers of science who suspect mysticism in this paragraph should observe that I am making a checkable claim about the behavior of mathematicians, not a philosophical claim about Truth.

mental representation of a particular concept is an elaborate *metaphor*. If you can find the right metaphor for a mathematical object, you can follow proofs concerning the object much more easily and you can frequently avoid falling into conceptual traps. (Not only that, it is the mental representation that suggests how the mathematical structure can be used in applications.) Following the proof line by line may convince one that the theorem is correct, but it gives no understanding unless the proof aligns in some sense with one's internal representation of the concept.[8] Indeed, the hope is that it will *refine* or *reform* one's inner representation of the concept.[9]

The statements in the preceding paragraph about mental representation are controversial: they reflect my own position but not the position of all mathematicians. Those statements caused far more correspondence than any others in this article. Some said that they do not use mental representations. For them, mathematics is all syntax. Others were dismayed by the syntactic proof and felt that the primary purpose in teaching was to transmit to the students useful mental representations of the concepts. Thus there are two kinds of mathematicians: Those for whom the mental representation (they often say "intuition" or "understanding") is paramount, and those who insist that syntax is primary. The gulf between these two kinds of mathematicians is vast. It is as if we were two different kinds of intelligent beings who are deluded into thinking we are communicating with each other. (But we *do* communicate.)

It is not unreasonable to assume that some of our students tend one way and some the other. I am in the mental representation camp, but in recent years, I have used syntax and transformations of statements in class more than I used to, and I believe it makes a difference for the better to the students.

4.4. Explicit use of logic. Even mathematicians in the mental representation camp use computation in proofs. Finding a suitable representation of an object that allows one to compute is as old as mathematics. However, most mathematicians rarely compute explicitly with the rules of logic as exemplified in 4.2.2. I believe that mathematicians should be aware of the possibilities of this approach, in teaching if not in their own research.

Recommendation. *Transmit your mental representation of concepts whenever you can, but* also *give proofs as explicit logical calculations when appropriate, because that provides the student with a second way to deal with the problem and provides him or her with the tools to carry out similar proofs.*

Related to explicit mention of logical concepts is the idea of giving a rule of inference for particular types of objects. For example, setbuilder notation has the following rules of inference: From the statement $a \in \{x|P(x)\}$ (where $P(x)$ is a predicate) one can deduce $P(a)$, and from the truth of $P(a)$ one can deduce

[8]Some readers commented that after reading the syntactic proof in 4.2.2 they suddenly understood that in a totally ordered set the condition to be proved is merely the contrapositive of transitivity. So a syntactic type of proof can be illuminating, too.

[9]There is a sizeable research literature on the subject of mathematicians' and students' mental representations of mathematics. See the articles in [Schoenfeld, 1987a], particularly [Schoenfeld, 1987b] and [Maurer, 1987], as well as [Harel and Dubinsky, 1992], [Miller, 1987] and the discussion starting in the second column of page 1187 of [Devlin, 1992].

$a \in \{x|P(x)\}$. My students have found this explicit mention of allowable inference to be helpful.

Recommendation. *Give explicit rules of inference for concepts when they are introduced.*

Note that this was done in the specification for ordered pairs in Section 2.3.

Lamport [1993] provides a detailed model for presenting proofs in a structured way that have the potential for clarifying proofs in either style, symbolic or based on mental representations.

5. TYPES AND POLYMORPHISM. In Pascal and in other typed programming languages, if you declare a variable to be of type Boolean and then try to set it equal to 3 the compiler gives you an error message. This is an example of mismatched *types*.

5.1. The multiplicity of types. The further students go in mathematics, the more different types of data they have to deal with. The typical second or third semester calculus course introduces two and three dimensional vectors, matrices, functions F: $\mathsf{R} \times \mathsf{R} \times \mathsf{R} \to \mathsf{R}$ ("scalar fields"), functions F: $\mathsf{R} \to \mathsf{R} \times \mathsf{R} \times \mathsf{R}$ ("paths in space") and functions F: $\mathsf{R} \times \mathsf{R} \times \mathsf{R} \to \mathsf{R} \times \mathsf{R} \times \mathsf{R}$ "vector fields") to know about, as well as all the objects of single-variable calculus. At some point we cease to be able to distinguish all these different things by different letters and typefaces, and the students have to learn to understand the types of the expressions they see by reading the surrounding text.

More than that, the meanings of the operator symbols in the formulas at that level may depend on the types of the operands. Consider the "$\times$" symbol. In the expression 3×5 it denotes numerical multiplication. If A and B are three dimensional vectors, $A \times B$ denotes the vector product, but if they are sets, it denotes the cartesian product. In computer science terms, the "$\times$" symbol is "polymorphic", in the sense that its meaning is dependent on the types of its arguments.

Recommendation. *Use the concepts of type and polymorphism explicitly to help students to understand and avoid the traps of type confusion.*

Most students now have had some experience with programming languages that use typing. I have found that to refer to types explicitly is helpful. I write "`TYPE ERROR`" on their homework when such a mistake is made, and sometimes when I forget to put the little arrow over a symbol for a vector on the blackboard, I get a chorus of "`TYPE ERROR`" from the class, which I think is great.

5.2. Teaching conceptual distinctions. A particularly bad typing error concerns functions. A function F: $A \to B$ has a value $F(a)$ at each element of A and, particularly in undergraduate mathematics, it may be given by an expression that is used to compute its values. The function, its value at an input, and an algorithm for computing it are three different mathematical objects that must be kept distinct.[10] Students often learn to cope with this in calculus courses by gaining an

[10]Our mathematical ancestors confused these, too [Selden and Selden, 1992], [Sfard, 1992].

implicit understanding of the differences. Because the distinctions are not *explicit*, the students' understanding is not on firm intellectual ground.

For the most part, we do not try very hard to convince our students that a function is not the same thing as its defining expression or its values. I have pushed that point in some courses I teach and it helps. If you really want them to know it, of course, you have to test them on it. Far too many mathematicians are unwilling to try testing first and second year students on conceptual content of this sort because they feel that most students will fail the questions. In fact, students can be taught conceptual distinctions if the teacher starts slowly and asks very simple questions at first. All you have to be willing to do is give up about 20% of the "content" of the course. I believe the gain far outweighs the loss, in courses for non-math-majors and math majors alike.

Recommendation. *Expect conceptual understanding at the appropriate level from all students in any course, and test them on it.*

6. SELF-MONITORING

6.1. Name your behavior. The New Hacker's Dictionary [Raymond, 1991] is a compilation of computer jargon which has to be one of the most enjoyable dictionaries ever composed. One thing that becomes noticeable if you read it straight through is the number of words and phrases hackers[11] have invented to describe their own mental states or behavior while working. This is discussed in the introduction to the Dictionary. "Juggling eggs", for example, refers to the necessity of keeping a lot of details in your head while modifying a program—with the consequence that an interruption can cause you to scramble the program (this is a paraphrase of the book's definition).

Of course, that is a phenomenon familiar to research mathematicians. You can't spend short, separated pieces of time trying to understand a complicated mathematical phenomenon; you need the time to concentrate and to get it all in your head at once—to be in "hack mode" (p. 190 of [Raymond, 1991]). The point is, mathematicians don't have a name for this, as far as I know. Computer hackers do. We should emulate them.

Computer people give names to their own counterproductive behavior quite freely—look up "creationism", "kluge", "mung" and "thrash" in [Raymond, 1991]. (I have personally munged several chapters of my class notes and had to tear them up and start over.) It would be particularly helpful to give names to common mistakes made by students in math courses. Pólya [1948] emphasized the importance of introducing notation for the quantities in a problem. It is equally important to name *behaviors*, both useful and harmful, that occur in problem solving.

Recommendation. *Describe and* name *the common kinds of mistakes students make.*

If there is a memorable name for such mistakes the student will be more likely (and will find it easier) to monitor his or her behavior. Self-monitoring is widely

[11]A hacker is someone who programs for the sake of programming—although useful tools may result, they are not the primary motivation—and who enjoys learning and using the obscure features and behavior of various operating systems and programming languages. The latter-day meaning of someone who breaks into private systems is not intended here.

cited in the educational literature as one of the properties that distinguish good students from poor ones. See [Resnick, 1987], pages 25–27, and [Schoenfeld, 1987c].

One typical mistake occurs when using concepts that by definition require the existence of something. For example, for integers m and n, m divides n if there is an integer q such that $n = qm$. When faced with proving that if m divides both n and p, then m divides $n + p$, a common mistake is to write down the assumptions as $n = qm$ and $p = qm$, using the same q. Recently in class I exclaimed, "If Bob and Ray are both married, that doesn't mean they have the same wife!" If I had said this on the network, such behavior might have become known as "existential bigamy" (or some such phrase).

It would be useful to come up with punchy names for good behaviors such as the following:

- Working examples before attacking the general case of a problem.
- Naming all the variables in a problem.
- Checking special cases of a statement to see if it is consistent with the rest of mathematics. (This is sometimes called a "sanity check".)

We should also name destructive behaviors such as these:

- Forgetting to check trivial cases. Hackers have an analogous error they call a "fencepost error"—getting the bounds wrong in a loop is an example.
- Proving an implication backward—in other words, being asked to prove $P \Rightarrow Q$ and coming up with a proof of $Q \Rightarrow P$. This is distressingly common among students whom I teach mathematical reasoning.
- Reading variable names as labels ([Nesher and Kilpatrick, 1990], pages 101–102) so that a statement such as "There are six times as many students as professors" gets translated as $6s = p$ instead of $6p = s$ (where p and s have the obvious meanings).

6.2. Context. People who have grown up together or who have worked in the same place for a long time have what have been called "high-context" conversations with many elliptical references to shared ideas, opinions and experiences. A group of people from different cultures or who live in a large city and don't know each other well will have "low-context" conversations with more made explicit and more attempt to avoid the assumption that the others share one's point of view.

Mathematicians seem to avoid connotative, high-context conversation about doing mathematics even though the potential is there for communicating quite complex ideas about the subject and about one's behavior while doing it. It is clear from the New Hacker's Dictionary that hackers do have high-context communication about these things. Mathematicians have been trained explicitly and through bitter experience that connotations can mislead when doing a proof. Perhaps many of us have mistakenly applied this lesson to other areas of our life, insisting on low-context conversation even when high-context conversation is possible. Or perhaps bright people who are not particularly talented at picking up social context are attracted to mathematics.

I don't know how we can change the mathematical culture to encourage high-context interaction. Perhaps the spread of email will help; linguists have discovered that linguistic innovation spreads much more rapidly in a language spoken by a large number of people in contact with each other than it does in small groups.

ACKNOWLEDGMENTS. This work has benefited by discussion with and suggestions and corrections from Atish Bagchi, Michael Barr. Stephen J. Bevan, J. E. Fritz, Leonard Gillman, David Gries, C. A. R. Hoare, Colin McLarty, Eric S. Raymond, Daniel M. Rosenblum, Guy Steele, and Leon Sterling. I am particularly grateful to Eric Raymond for many detailed suggestions and insights, particularly in Sections 2 and 6, and for organizing an electronic mailing list for discussions of earlier drafts of this article and of [Bagchi and Wells, 1993].

REFERENCES

[Bagchi and Wells, 1993] Atish Bagchi and Charles Wells. *The varieties of mathematical prose*. Available by anonymous FTP from `ftp.cwru.edu`. It is the file `mathrite.dvi` in the directory `math / wells`. 1993.

[Barr, 1993] Michael Barr. *Functional set theory*. Preprint. Department of Mathematics, Burnside Hall, McGill University, 805 Sherbrooke St. West, Montréal, P.Q., Canada H3A 2K6. It is available by anonymous FTP from `triples.math.mcgill.ca.` in the files `variable.sets.tex.Z` and `variable.sets.dvi.Z` in the directory `pub / barr`, 1993.

[Boas, 1981] Boas. *Can we make mathematics intelligible*? American Mathematical Monthly, 88(10):727–731, December 1981.

[Devlin, 1992] Keith Devlin. *Computers and mathematics*. Notices of the American Mathematical Society, 39(10):1186–1188, 1992.

[Dijkstra and Scholten, 1990] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, 1990.

[Gillman, 1987] Leonard Gillman. *Writing Mathematics Well*. Mathematical Association of America, 1987.

[Gries and Schneider, 1993] David Gries and F. B. Schneider. *A Logical Approach to Discrete Mathematics*. Springer-Verlag, 1993.

[Gries, 1991] David Gries. *Teaching calculation and discrimination: A more effective curriculum*. Communications of the ACM, 34:44–55, 1991.

[Harel and Dubinsky, 1992] Guershon Harel and Ed Dubinsky, editors. The Concept of Function, volume 25 of *MAA Notes*. Mathematical Association of America, 1992.

[Kieran, 1990] Carolyn Kieran. *Cognitive processes involved in learning school algebra*. In *Mathematics and Cognition*, Pearla Nesher and Jeremy Kilpatrick, editors, ICMI Study Series, pages 96–112. Cambridge University Press, 1990.

[Knuth *et al.*, 1989] Donald E. Knuth, Tracy Larrabee, and Paul M. Roberts. *Mathematical Writing*, volume 14 of *MAA Notes*. Mathematical Association of America, 1989.

[Laborde, 1990] Colette Laborde. *Language and mathematics*. In *Mathematics and Cognition*, Pearla Nesher and Jeremy Kilpatrick editors. ICMI Study Series, pages 53–69. Cambridge University Press, 1990.

[Lamport, 1993] Leslie Lamport. *How to write a proof*. Technical Report SRC-094, Digital Systems Research Center, February 1993. Available by FTP from `gatekeeper.dec.com` in the directory `/archive / pub / DEC / SRC / research-reports`. It is file `SRC-094.ps.Z`.

[Maurer, 1987] Stephen B. Maurer. *New knowledge about errors and new views about learners: What they mean to educators and more educators would like to know*. In *Cognitive Science and Mathematics Education*, Alan Schoenfeld, editor, pages 165–188. Lawrence Erlbaum Associates, 1987.

[McLarty, 1993] Colin McLarty. *Numbers can be just what they have to*. Nous, 27:487–498, 1993.

[Miller, 1987] Arthur I. Miller. *Imagery in Scientific Thought*. MIT Press, 1987.

[Nesher and Kilpatrick, 1990] Pearla Nesher and Jeremy Kilpatrick. *Mathematics and Cognition*. ICMI Study Series, Cambridge University Press, 1990.

[Pólya, 1948] G. Pólya. *How to Solve It*. Princeton University Press, 1948.

[Raymond, 1991] Eric Raymond. *The New Hacker's Dictionary*. The MIT Press, 1991.

[Resnick, 1987] Lauren B. Resnick. *Education and Learning to Think*. National Academy Press, 1987.

[Schoenfeld, 1985] Alan Schoenfeld. *Mathematical Problem Solving*. Academic Press, 1985.

[Schoenfeld, 1987a] Alan Schoenfeld, editor. *Cognitive Science and Mathematics Education*. Lawrence Erlbaum Associates, 1987.

[Schoenfeld, 1987b] Alan Schoenfeld. *Cognitive science and mathematics education: An overview*. In *Cognitive Science and Mathematics Education*, Alan Schoenfeld, editor, pages 1–32. Lawrence Erlbaum Associates, 1987.

[Schoenfeld, 1987c] Alan Schoenfeld. *What's all the fuss about metacognition?* In *Cognitive Science and Mathematics Education*, Alan Schoenfeld, editor. Lawrence Erlbaum Associates, 1987.

[Selden and Selden, 1992] Annie Selden and John Selden. *Research perspectives on conceptions of functions*. In *The Concept of Function*, Guershon Harel and Ed Dubinsky, editors, volume 25 of *MAA Notes*, pages 1–16. Mathematical Association of America, 1992.

[Sfard, 1992] Anna Sfard. *Operational origins of mathematical objects and the quandary of reification—the case of function*. In *The Concept of Function*, Guershon Harel and Ed Dubinsky, editors, volume 25 of *MAA Notes*, pages 59–84. Mathematical Association of America, 1992.

[Steenrod *et al.*, 1975] Norman E. Steenrod, Paul R. Halmos, Menahem M. Schiffer, and Jean A. Dieudonné. *How to Write Mathematics*. American Mathematical Society, 1975.

[van Gasteren, 1990] A. J. M. van Gasteren. *On the Shape of Mathematical Arguments*, volume 445 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[Wells, 1993] Charles Wells. *Discrete mathematics*. Class notes, Department of Mathematics, Case Western Reserve University, 1993.