

0 Summary

The design of mathematical arguments is traditionally an intuitive activity. When faced with a problem, most mathematicians will ponder about concepts related to the problem, let their minds draw out the relevant properties, and finally combine these towards a solution using common proof strategies and techniques. Though this approach has carried the field of mathematics a long way, it is a completely implicit methodology: If the right concepts do not come to mind, or an attempted proof strategy fails, the only thing to do is think harder and try again; the invention of new concepts and techniques must come from mathematical insight. Mathematics pedagogy consists largely of exposing students to existing theories, concepts, and methods, hoping they will learn through analogy and practice.

The calculational method of proof design extends the traditional repertoire of implicit, intuitive techniques with explicit, constructive ones. In a calculational approach, one first translates the givens and the demonstrandum into a carefully chosen formal notation. Then, by syntactically analyzing the resulting formulae, one designs the chain of formal manipulations leading from one into the other; the reasoning is guided by the symbols representing the concepts, rather than our intuitions about them. Through their technical writings, Dutch computing scientists Edsger Dijkstra, Wim Feijen, and Netty van Gasteren have demonstrated the efficacy and broad applicability of this approach.

The impetus for the present project is two-fold: Firstly, because research efforts in calculational mathematics have been so diffuse, there is no coherent exposition of the framework as a whole, though expositions of the more common calculational conventions and tools do exist. Secondly, and more importantly, the question of how to use formal techniques —that is, the methodology of calculational proof design— has scarcely been addressed in the literature.

This project addresses both shortcomings: In the first strand of the project, existing research in calculational mathematics will be synthesized and generalized into a unified exposition of the calculational framework. By culling together the essential conventions and tools of the framework, this exposition will simultaneously fulfill a long-standing need within the calculational community, as well as increasing the exposure of formal techniques to the wider scientific community. In the second strand, the use of formal methods will be illustrated through the design of substantial mathematical theorems and theories. Calculational techniques will be explored in detail and in depth, yielding an important resource for mathematical methodology and pedagogy.

Funding is sought for one PhD student.

1 Track record

1.X Jeremy Weissmann

Jeremy Weissmann received his BA in Mathematics from Northwestern University in Evanston, Illinois. During his three years there, he received a thorough training in all areas of mathematics, studying abstract algebra, real and complex analysis, number theory, topology, graph theory, combinatorics, logic, and set theory. He graduated *cum laude* with honors, having written a thesis in mathematical logic.

He also attended the prestigious Budapest Semesters in Mathematics program, founded by Paul Erdős. There, he studied advanced mathematics under some of Hungary's foremost mathematicians, receiving his marks from the Technical University of Budapest.

Last year, he was awarded a Fulbright grant to study calculational approaches to graph theory and math pedagogy, and joined the Technical University of Eindhoven's Software Construction group as a visiting fellow. His advisor there was Wim Feijen, who founded the calculational style along with the late computing scientist Edsger Dijkstra. Professor Feijen also supervised the master's thesis of group member Arjan Mooij [XX4]. As part of his research, Mr Weissmann helped create and maintain *mathmeth.com*, a website devoted to mathematical methodology and calculational methods. His technical work and the work of his colleagues can be found at that site.

Currently, he continues his research through correspondence with group members Roland Backhouse and João Ferreira, as well as with his colleagues in Eindhoven. He is an advisor for the undergraduate thesis of Apurva Mehta, which is an exposition of the heuristics of program design [XX35].

2 Case for support

2.0 Background

We may judge the quality of a mathematical argument by a number of criteria: correctness, ease of verification, elegance, and the generality and teachability of its methods, to name just a few. However, many traditional proofs fail to meet these criteria satisfactorily. For instance, despite the rigor of the refereeing process, there are errors in published proofs. Proofs are rarely self-contained, meaning that readers may have to consult several references before they can be convinced of an argument's correctness. Proofs often have sizeable gaps left for readers to work out, yet it is not always clear where these gaps are. The clumsiness and imprecision of natural language can make proofs awkward and difficult to follow. Arguments are often sloppy and "hacked". The use of special-purpose inventions and tricks in proofs renders generalization impossible, and leaves readers wondering how they could design these inventions themselves. As several authors have lamented [XX0, XX1], this is unfortunate: A proof, being the fundamental unit of mathematical discourse, should be correct, convincing, and readable, if nothing else.

In the first decades of the 20th century, many mathematicians turned to formal methods to improve the quality of mathematical arguments [XX2, XX3]. Indeed, in a formal proof, the assumptions and the structure of the argument become absolutely explicit. Moreover, a formal proof is amenable to mechanical verification. However, the mathematical community largely considered these early efforts to be a failure, arguing that formal methods are cumbersome, unwieldy, and unnatural, and that the gains they make in precision do not make up for the loss in elegance and readability.

The pioneering efforts of Dutch computing scientists Edsger Dijkstra, Wim Feijen, and Netty van Gasteren helped to overcome these objections, and finally craft formal techniques into effective human tools. The turning point in this development was van Gasteren's PhD thesis [XX5], where she demonstrated that formal methods could not only be used to prove theorems, but could in many cases do so more elegantly than traditional methods. Her streamlined expositions and proofs paved the way for a revolution in mathematical style. The emerging style was dubbed "calculational", because the formal manipulations were similar to the ones used so well in, for example, differentiating x^2 , solving $2x + 1 = 3$, or computing $346 + 123$.

The most important consequence of this new style was in methodology: Because calculational proofs were completely syntactic, their design became highly explicit, constructive, and teachable. However, this notion of syntax-driven design was not explored for mathematics in general at first, but rather only in algorithm design, reflecting the primary interests of the creators of the calculational style. In the meantime, the science of calculating programs from their specifications has been explored extensively [XX6, XX7, XX8, XX9, XX10, XX11, XX35], and has developed into a powerful, general methodology.

By contrast, the development of calculational methodology for mathematics in general has been far less successful. The first real effort at such a development was Dijkstra's [XX18]. However, this monograph was never completed, and consists only of a handful of examples, dictums, and history lessons. The textbook [XX12] gives an exposition of only a subset of calculational tools, and focusses almost exclusively on presenting particular formalisms rather than on the calculational design of theorems and theories. Another recent textbook [XX17] focusses primarily on structuring mathematical arguments rather than designing them, and furthermore limits its scope to high school mathematics.

2.1 Timeliness

The above brief historical sketch argues for the importance of calculational mathematics, and demonstrates the present need for a coherent, unified exposition of the modern calculational framework and method. It has been 20 years since van Gasteren's PhD thesis revealed the possibilities of the calculational style, and it is time to consolidate the diffuse research that work inspired.

There are other reasons why this project is particularly timely. With the death of Dijkstra and van Gasteren in 2002, and the retirement of Feijen in 2006, the growth of calculational research and education

has slowed dramatically. This project would help revitalize the field, and inspire new directions for research.

Finally, in 2006, Apurva Mehta and Jeremy Weissmann launched the website *mathmeth.com*. The site has brought together calculational mathematicians around the world, and serves as a distribution point for new work in the field. In the coming year, an upload interface and message board will be added, allowing researchers to discuss their work electronically. This project can thus be carried out with scrutiny and feedback from a worldwide network of calculational mathematicians, thereby greatly improving its quality, and accelerating the dissemination of results.

2.2 Calculational mathematics

This subsection discusses the main ingredients of the calculational style, which will be explored in depth in the first strand of this project.

2.2.0 Approach

quotation from MoM [XX9] “a nearly undefinable mathematical discipline, aiming at simplicity and at not taking premature or hasty design decisions”

EWD619 [XX16]

—be organized, be explicit, be aware of the choices you make, separate concerns wherever possible, don’t draw needless distinctions, formulate goals clearly, name the right concepts, form generalizations, use knowledge sparingly and only as necessary, for example—

naming

...the choice of notation is no longer a mere matter of style, but often the deciding factor between success and failure. Many notations in the literature are like Roman numerals: a cumbersome shorthand and a pain in the neck to reason with.

mention basic algebra

For example, sometimes a well-chosen name can turn a difficult problem into a simple one. In an exercise from [XX11], the reader is asked to prove the correctness of the postcondition in a program with the following shape:

```

[[  var x, y : int
   ; x, y := 0, N-1
   ; do x ≠ y → ... od
     { f.x = ⟨↑i : 0 ≤ i < N : f.i⟩ }
]]
    ,

```

where \uparrow denotes a quantified maximum, here the maximum of all $f.i$, $0 \leq i < N$. When asked to solve this problem, many people interpret the program operationally, and conclude that the invariant should be “either $f.x$ or $f.y$ is the maximum over all indices inspected so far”, which has a cumbersome formal translation, and which makes the rest of the correctness argument long and unpleasant. By contrast, in a calculational approach [XX21], we simply observe that $f.x$ in the postcondition is asymmetric in

program variables x and y , and decide to replace it with some expression P , yielding invariant:

$$P = \langle \uparrow i : 0 \leq i < N : f.i \rangle .$$

This name is the only invention necessary. Now writing down the formal specification of P , we have:

$$\begin{array}{ll} x = 0 \wedge y = N-1 & \Rightarrow P = \langle \uparrow i : 0 \leq i < N : f.i \rangle & \text{for initialization, and} \\ x = y & \Rightarrow P = f.x & \text{for termination,} \end{array}$$

which, using substitution of equals for equals and basic arithmetic, gives:

$$P = \langle \uparrow i : x \leq i \leq y : f.i \rangle .$$

This choice of invariant allows for a smooth correctness proof, and while it is difficult to find intuitively, it is trivial to calculate, as we have seen.

2.2.1 The calculational system

I will discuss the ingredients of the calculational system, as well as some of the extensions to that system which have arisen recently. For example: symbols, contexts, types, calculations, WF's proof format and refinements, predicate calculus, lattice theory, interfaces, symbol dynamics (of transitivity, Leibniz, monotonicity, punctuality).

I will mention here how some of my departures from tradition can lead to great gains, like for instance the use of structure-valued contexts.

2.3 Problems and theories

In the core of this project, the calculational method will be applied to the design of various mathematical theorems and theories, ranging in complexity from simple to substantial. The primary reference will be [XX12], a rich resource of calculational problems and theories. This subsection discusses the sorts of problems and theories that will be explored.

2.3.0 Predicate and relational calculi

Thanks to their generality, the predicate and relational calculi have shown themselves to be indispensable tools in the proofs of theorems from diverse fields of mathematics. However, there are several reasons why this project will investigate the design of theorems from the predicate and relational calculi *per se*.

First, because these calculi consist of just a handful of symbols and postulates, they provide idealized "toy" mediums for introducing the notion of calculational proof design.

Second, because of their conceptual simplicity, the design principles outlined in Section 2.2.0 can be laid bare, without becoming entangled with conceptual concerns.

And finally, theorems from the predicate and relational calculi will best illustrate new, experimental calculational techniques, for example the use of nonscalar contexts [XX27, XX28], Eric Hehner's technique of local contexts [XX26], case analysis [XX30], and Skolemization [XX29].

2.3.1 Number theory

Many calculational approaches to number theory have been explored over the past 20 years, and the present project will extend existing research in several new directions.

The first direction is an investigation of the extent to which programs can be used to prove theorems of number theory. One of the earliest examples of calculational number theory is Chapter 11 of van Gasteren's PhD thesis, which explores how theorems involving the greatest common divisor can be proved using Euclid's algorithm. That technique was used again in Dijkstra's [XX40] to prove facts about the Fibonacci sequence. Given these successes, it seems clear that algorithmic approaches to number theory are ripe for investigation.

The second direction is the design and application of special-purpose calculi in order to capture number-theoretic concepts more compactly. There have already been promising experiments in this direction. For example, the technical note [XX24] explores a calculus of the predicate 'positive' and the relation ' \leq ', resulting in a host of useful calculational theorems. As another example, we define $(\#n)_p$ to be the exponent of p in n 's prime factorization. This leads to a calculus of $\#$ along with the other natural operators and relations, thereby compactly capturing all the information about a number's prime factorization without invoking sets or sequences. The $\#$ -calculus was used in [XX34] to prove that the greatest common divisor distributes over the least common multiple, and in [XX33] to derive a proof of Euclid's lemma.

The final direction is to continue efforts to revise early calculational work on number theory. These efforts have led to exciting new discoveries. For example, the work on residue classes in Dijkstra's early note [XX42] was taken up again in [XX12] and [XX41], inspiring a new collection of results dubbed the "chunking lemmata" [XX36, XX39]. These lemmata show calculational potential and merit further exploration. Also, an early effort [XX31] to prove unique factorization has been revised in [XX32], which represents a landmark in the use of symbols to guide the design of proofs.

2.3.2 Discrete mathematics

Within discrete mathematics, two fields will be investigated: game theory and graph theory. The first, smaller investigation aims to extend the technique in [XX25] used to solve a group strategy problem. In that problem, each member of a group of players is given a hat of some color, and can see others' hats but not their own. They each simultaneously guess the color of their own hat, and win as a group provided at least one player guesses correctly. In the solution in [XX25], the players' answers, strategies, and available information are encoded as functions from the players to the hat colors. Then using a simple extension of the predicate calculus, the winning strategy can be calculated straightforwardly. The hope is that this approach can be extended to more complex problems to form a small, calculational theory of games. Towards this end, Nathaniel Rudavsky-Brody's proposed research into the calculational design of probabilistic programs may prove invaluable.

The research in graph theory will follow Jeremy Weissmann's Fulbright research, which lays out a programme for calculational and algorithmic graph theory. Initial investigations [XX23] suggest that algorithmic techniques can be used to prove theorems of graph theory, through the notions of invariants and program inversion. For example, in [XX22] the theorem "a graph is bipartite if it has no odd cycles" is elegantly proved by designing a program to compute a bipartition of a graph. The sufficient condition "no odd cycles" does not need to be known beforehand, but is derived in the design process.

The explorations in [XX23] also suggest that some graph-theoretic concepts may need to be generalized before they can be amenable to formal treatment. For instance, by allowing edges to be pairs of some superset of the vertex set, graphs can be generalized —to *grophs*— in order to simplify many existing concepts and proofs. Given a groph G and subgroph H , we define $G \setminus H$ to be the collection of all vertices and edges of G not in H . This collection may not constitute a graph, but we can still define the degree of a vertex in this "generalized graph" to be the number of edges incident with it. In fact, we can even define the degree in $G \setminus H$ of a vertex not in $G \setminus H$. Thus, letting $d.X.v$ denote the degree in X of v , we have for instance:

$$d.G.v = d.H.v + d.(G \setminus H).v \quad (\text{for all } v)$$

a useful theorem which cannot be formulated in terms of traditional graphs and their complements. Using this notation, one can also formulate the theorem:

$$\neg(xy \in H) \iff (x \sim y \text{ in } G \setminus H) \quad , \quad \text{where } \sim \text{ is the "connected" relation}$$

which allows for a streamlined derivation of “an acyclic graph has a node of degree at most 1”.

This project will organize and extend these explorations, ultimately demonstrating how calculational mathematics and program design can work hand in hand towards the creation of a mathematical theory of graphs.

2.4 Benefits

2.4.0 For calculational mathematics

Calculational mathematics lacks a clear, coherent exposition of its methods and methodology. The closest it has to this is [XX12]; however, the focus of [XX12] is primarily on providing particular formalisms and formalized proofs for established fields of mathematics, not on methods of designing formalisms or proofs, let alone mathematical theories. Moreover, the failure to exploit monotonicity, among other important new calculational techniques, renders that work obsolete. By synthesizing state of the art tools and methodologies, this project will help to unify calculational mathematics, thereby promoting the development of the field.

2.4.1 For mathematics in general

The calculational method can help mathematicians extend the boundaries of mathematics, by streamlining and reducing the amount of mental energy needed to find proofs. Intuition and creativity will undoubtedly always play important roles in mathematical discovery and development. The use of formal tools does not work counter to these implicit methods: the goal of the calculational style is not to try to reduce all problems to symbol manipulation, but rather to use calculation judiciously, wherever it can be fruitfully applied.

Often, calculation can heighten the effectiveness of intuition and creativity. For example, in a traditional mathematical proof of Sylvester’s theorem (“there is a line through precisely two of a collection of distinct, noncollinear points in the plane”), one has to invent the need for the Euclidean distance, even though the theorem itself makes no mention of a Euclidean metric. By contrast, an absolutely standard calculational/algorithmic approach to the problem [XX13] requires us, in the interest of a termination argument, to find some real function of a point and a line in Euclidean space, bounded from below. The invention of the Euclidean distance is then far more dictated.

By developing a thorough, unified exposition and demonstration of the calculational method, this project will help traditional mathematicians incorporate the calculational method into their own work.

2.4.2 For mathematical pedagogy

The explicitness and concreteness of calculational mathematics make it an excellent medium for math education. Symbols can help guide the design of mathematical concepts, theorems, and theories. Building on the efforts in [XX12, XX17, XX19, XX20], the second strand of this project is intended to form the basis for a curriculum in calculational mathematics pedagogy.

2.4.3 For wider pedagogy

Calculational mathematics is not only an excellent tool for teaching mathematics, but also thinking in general. When we are just learning to think, the concepts we reason about should be very simple, so we can focus on the reasoning itself. Learning to think by, for instance, solving the problem of world hunger is as futile as learning to swim in a tidal wave. So, even though the real-world concepts we wish to reason about are complex and hazy, it is essential to practice in an idealized domain. Mathematical properties are nearly ideal in this respect, as they are simpler and more easily articulated than nonmathematical ones, and among these, calculational properties are the simplest —because they are formulated in terms of symbol manipulation—, and hence truly ideal.

Calculation is just symbolic manipulation, so in a calculational problem we are able to focus entirely on questions of *how* to manipulate: how to define goals clearly, separate concerns, and form generalizations; how to be explicit, organized, and aware of choices. These skills, which outside of mathematics are fairly vague, can now be concretely understood and explicitly taught in terms of symbols. Thus the calculational method helps foster the skills of creativity and discipline of thought (and a taste for simplicity) which become so crucial as the complexity of problems grows.

3 Collaboration

3.0 Within the research group

Roland Backhouse, Arjan Mooij, Joao Ferreira, Nathaniel Rudavsky-Brody

3.1 External collaborators

Wim Feijen, Dan Grundy, Apurva Mehta, Diethard Michaelis, Tom Verhoeff

X.0 Work programme/timeline

The project comprises two separate yet related strands. In the first strand, existing research in calculational mathematics will be synthesized and generalized into a unified exposition of the calculational framework. By culling together the essential conventions and tools of the framework, this exposition will simultaneously fulfill a long-standing need within the calculational community, as well as increasing the exposure of formal techniques to the wider scientific community. In the second strand, the use of formal methods will be illustrated through the design of substantial mathematical theorems and theories. Calculational techniques will be explored in detail and in depth, yielding an important resource for mathematical methodology and pedagogy.

Phase 0: exposition of the calculational framework (1 year, 4 months)

Exposition of the basic conventions of the framework: notations, predicate calculus, lattice theory, quantifier notation, proof format. (4 months) Exposition of calculational strategies: symbol dynamics, in particular for transitive relations, widening/shrinking heuristics, use of scalar and structural context. (4 months) Exposition of discipline of thought: cleanliness, organization, explicitness, the use of formalization, naming, notation, interface design, separation of concerns/disentanglement, abstraction. (8 months)

Phase 1: application to theorem/theory design (1 year, 8 months)

Treatment of several examples from the predicate calculus: calculational theorem design, use of context, Skolemization, case analysis. (2 months) Calculational approaches to number theory: algorithmic number theory, (4 months) the design and application of special purpose calculi, new interfaces with number-

theoretic concepts. (4 months) Explorations into game theory: Applications of algebra, and probabilistic programs. (2 months) Development of calculational and algorithmic graph theory: application of relational calculus, abstraction of graph-theoretic concepts, design of usable formalisms for reasoning about paths, trees, cycles, colorability, (4 months) ; casting graph-theoretic concepts in terms of algorithms, designing graph theory algorithms using program inversion and stepwise invariant design. (4 months)

X.1 Project management

Progress towards the goals above will be reviewed weekly with Roland Backhouse and the other researchers in the Foundations of Programming Group.

X.2 Dissemination of results

Results will be presented first at meetings of the Tuesday Morning Clubs, with other members of the Foundations of Programming Group. They will also be shared with members of the mathematics department, in order to receive feedback from the larger mathematical community.

Significant results will be submitted to leading mathematical and pedagogical journals, with an emphasis on journals which bridge the two fields, such as the American Mathematical Monthly.

Naturally, the work will be available online, through mathmeth.com , a website which has brought together calculational mathematicians around the world, and serves as a distribution point for new work in the field. In the coming year, an upload interface and message board will be added, allowing researchers to discuss their work electronically. This project can thus be carried out with scrutiny and feedback from a worldwide network of calculational mathematicians, thereby greatly improving its quality, and accelerating the dissemination of results.

4.0 Work programme/timeline

The project comprises two separate yet related strands. In the first strand, existing research in calculational mathematics will be synthesized and generalized into a unified exposition of the calculational framework. By culling together the essential conventions and tools of the framework, this exposition will simultaneously fulfill a long-standing need within the calculational community, as well as increasing the exposure of formal techniques to the wider scientific community. In the second strand, the use of formal methods will be illustrated through the design of substantial mathematical theorems and theories. Calculational techniques will be explored in detail and in depth, yielding an important resource for mathematical methodology and pedagogy.

Phase 0: exposition of the calculational framework (1 year, 4 months)

Exposition of the basic conventions of the framework: notations, predicate calculus, lattice theory, quantifier notation, proof format. (*4 months*) Exposition of calculational strategies: symbol dynamics, in particular for transitive relations, widening/shrinking heuristics, use of scalar and structural context. (*4 months*) Exposition of discipline of thought: cleanliness, organization, explicitness, the use of formalization, naming, notation, interface design, separation of concerns/disentanglement, abstraction. (*8 months*)

Phase 1: application to theorem/theory design (1 year, 8 months)

Treatment of several examples from the predicate calculus: calculational theorem design, use of context, Skolemization, case analysis. (*2 months*) Calculational approaches to number theory: algorithmic number theory, (*4 months*) the design and application of special purpose calculi, new interfaces with number-theoretic concepts. (*4 months*) Explorations into game theory: Applications of algebra, and probabilistic programs. (*2 months*) Development of calculational and algorithmic graph theory: application of relational calculus, abstraction of graph-theoretic concepts, design of usable formalisms for reasoning about paths, trees, cycles, colorability, (*4 months*); casting graph-theoretic concepts in terms of algorithms, designing graph theory algorithms using program inversion and stepwise invariant design. (*4 months*)

4.1 Project management

Progress towards the goals above will be reviewed weekly with Roland Backhouse and the other researchers in the Foundations of Programming Group.

4.2 Dissemination of results

Results will be presented first at meetings of the Tuesday Morning Clubs, with other members of the Foundations of Programming Group. They will also be shared with members of the mathematics department, in order to receive feedback from the larger mathematical community.

Significant results will be submitted to leading mathematical and pedagogical journals, with an emphasis on journals which bridge the two fields, such as the American Mathematical Monthly.

Naturally, the work will be available online, through *mathmeth.com*, a website which has brought together calculational mathematicians around the world, and serves as a distribution point for new work in the field. In the coming year, an upload interface and message board will be added, allowing researchers to discuss their work electronically. This project can thus be carried out with scrutiny and feedback from a worldwide network of calculational mathematicians, thereby greatly improving its quality, and accelerating the dissemination of results.

5. Bibliography

- [XX0] Halmos, P. *How to write mathematics*
- [XX1] Knuth, D. *Mathematical writing*
- [XX2] Hilbert, D. *Grundlagen der Geometrie*
- [XX3] Russell, B. and Whitehead, A. *Principia Mathematica*
- [XX4] Mooij, A.J. *Master's thesis*
- [XX5] van Gasteren, A. *On the shape of mathematical arguments*
- [XX6] Backhouse, R. *Program Construction: Calculating Implementations from Specifications*
- [XX7] Dijkstra, E.W. *A Discipline of Programming*
- [XX8] Dijkstra, E.W. and Feijen, W.H.J. *A Method of Programming*
- [XX9] Feijen, W.H.J. and van Gasteren, A. *On a Method of Multiprogramming*
- [XX10] Gries, D. *The Science of Programming*
- [XX11] Kaldewaij, A. *Programming: The Derivation of Algorithms*
- [XX12] Gries, D. and Schneider, F. *A Logical Approach to Discrete Math*
- [XX13] Dijkstra, E.W. *EWD1016: A computing scientist's approach to a once-deep theorem of Sylvester's*
- [XX14] Dijkstra, E.W. and Scholten, C.S. *Predicate Calculus and Program Semantics*
- [XX15] Dijkstra, R. *Relational Calculus and Relational Program Semantics*
- [XX16] Dijkstra, E.W. *EWD619: Essays on the nature and role of mathematical elegance*
- [XX17] Ralph-Johan, B. and von Wright, J. *Mathematics with a little bit of logic: Structured derivations in high school mathematics*
- [XX18] Dijkstra, E.W. *Mathematical methodology (EWD1059, EWD1067, EWD1063, EWD1068, EWD1070, EWD1078, EWD1090, EWD1091, and EWD1094)*
- [XX19] Backhouse, R. *Algorithmic Problem Solving*
- [XX20] Gries, D. *ProgramLive!*
- [XX21] Weissmann, J. *JAW91: How can we learn to be creative?*
- [XX22] Weissmann, J. *JAW17: A first experiment in algorithmic graph theory*
- [XX23] Weissmann, J. *Notes on calculational graph theory*
- [XX24] Weissmann, J. *JAW13: The pos calculus, via a problem from Wim*
- [XX25] Weissmann, J. *Presentation to the Eindhoven Tuesday Afternoon Club, 30 May 2006*
- [XX26] Weissmann, J. *JAW74: A new calculational technique, via Eric Hehner*
- [XX27] Weissmann, J. *JAW87: The utility of unbracketed results*
- [XX28] Weissmann, J. *JAW89: Using structure-valued formulae*
- [XX29] Weissmann, J. *JAW77: Skolemization and case analysis*
- [XX30] Weissmann, J. *JAW59: Case analysis from a calculational perspective*
- [XX31] Dijkstra, E.W. *EWD755: Very elementary number theory redone*
- [XX32] Mehta, A. and Weissmann, J. *ARM7/JAW92: Designing a proof of unique factorization*
- [XX33] Weissmann, J. *JAW10b: A summary of my work on Euclid's lemma (revised)*
- [XX34] Weissmann, J. *JAW63: Yes, lcm distributes over gcd (and vice versa)*
- [XX35] Mehta, A. *Proving program correctness*
- [XX36] Weissmann, J. *JAW71: The chunking lemmata, via Kaldewaij*
- [XX37] **Weissmann, J. *JAW72: On long sequences of composite numbers (revised)***
- [XX38] **Weissmann, J. *JAW6: Not about primality***
- [XX39] Weissmann, J. *JAW94: An exercise in cleanliness and abstraction, via Knuth*
- [XX40] Dijkstra, E.W. *EWD1077: Fibonacci and the greatest common divisor*
- [XX41] Grundy, D. *Calculating with congruences*
- [XX42] Dijkstra, E.W. *EWD984: The study of a notion, viz. that of residue classes*
- [XX43]
- [XX44]
- [XX45]
- [XX46]

[XX47]
[XX48]
[XX49]
[XX50]