

# Towards Integrated Systems for Symbolic Algebra and Formal Constructive Mathematics\*

Robert L. Constable\*  
Paul B. Jackson\*  
Cornell University

May 13, 1998

---

\*Work supported in part by NSF grant CCR-9244739, NASA grant NGT-50786, ONR contracts N00014-92-J-1764 and N00014-91-J-4123

# 1 Introduction

## Background

The purpose of this paper is to report on our efforts to give a formal account of some of the algebra used in Computer Algebra Systems (CAS). In particular, we look at the concepts used in the so-called 3rd generation algebra systems, such as Axiom [4] and Weyl [9]. It is our claim that the Nuprl proof development system is especially well-suited to support this kind of mathematics.

We have discovered in the course of our work some interesting ways in which Nuprl can interact with a CAS, and we want to illustrate these. Discovering those interactions presented us with new ways of looking at the role of theorem provers, proof development systems and problem solving environments. We hope that people find these observations a valuable complement to the technical results we present.

The starting point for us was an attempt to apply Nuprl in the realm of computational science. We were led by discussions with Conal Mannion [6] to explore ways that we could provide a “semantics for computer algebra systems”. This reminded us of remarks by Dana Scott and led to reading carefully the papers of Clarke and Zhao on *Analytica* [1]. Finally we were led to looking in detail at Weyl and Axiom. We came to see a clear and deep connection between work in these systems and our own. Basically, the connection arises because these systems are very careful about the notion of an algebraic *domain*. They define such domains as we define types.

There is an even deeper connection between Axiom and Nuprl in that both are concerned with *constructive mathematics* in one form or another.<sup>†</sup> The issue of constructivity in algebra has a long and lively history, going back at least to Kronecker. Likewise, its role in logic is very significant. In fact, generally the connection between algebra and logic are deep and longstanding. So it is not surprising that today we find interesting connections between applied algebra and applied logic and between the CAS and the theorem proving systems.

As a first approximation to exploring these connections, let us draw an analogy between the subjects of ?????

**Symbolic Algebra** is concerned with computational content in algebra — with factoring polynomials, solving equations, differentiating expressions, etc. Although much of the work takes place in so called *classical mathematics*; nevertheless, careful development of topics in the subject, as in Axiom [4] and Weyl [9] for example, bears strong resemblance to constructive mathematics. We’ll see details of this in section ??????, but the reasons are clear, if we say we can factor a polynomial, we usually want to know the factors and compute

---

<sup>†</sup>Quotes from Davenport and Trager

further with them. Indeed the algorithms are so useful that many have been implemented and organized into computer algebra systems, CAS, (such as Axiom, Macsyma, Maple, Mathematica, Reduce, Weyl). These systems have become useful tools for scientists and engineers. The systems also provide more than algorithms, they present users with definitions of basic concepts that are used in the implementations.

**Constructive Mathematics** is also concerned with computational content in mathematics — with finding roots, defining functions implicitly, computing them, and in general constructing, analyzing and using mathematical objects. When a constructive mathematician claims that a polynomial can be factored, the claim is backed by an algorithm to factor (not necessarily a feasible one). Some of the constructions in this kind of mathematics are sufficiently useful that it makes sense to implement them. As in the algebra system, but now pervasively, the definitions must be implemented as well. Even more, and uniquely so, these systems will implement a formal concept of proof. There are now several systems that have followed the Cornell Nuprl (“new pearl”) system in *implementing constructive mathematics* (e. g. Alf, Constructor, Clam-Oyster, Coq, Lego).

Given the similarities between the subjects, what exact connections are there? Does it make sense to combine symbolic algebra systems and implementations of constructive mathematics, generally in the form of *theorem provers*? Is constructive mathematics an especially suitable foundation for computer algebra? Can algebra systems help in theorem provers?

There are obvious relationships between algebra systems and provers. As Clarke [1] has done, one can build a prover inside an algebra system. In this case, one must *trust* the algebra code, but typically this is complex and there is a lot of it. As the algebra system grows, the prover is more and more compromised. Already *Mathematica* is not a reliable system, and the code is not open to inspection. A system like Axiom is more open and checkable, but still is not coded for reliability in the same way as provers.

In the early years, we imagined deriving symbolic algebra algorithms in Nuprl and proving them correct. Indeed we were contacted by people from the algebra community who wanted to do this as well. But this is a large task; it would take years. Moreover, the verification technology is not yet adequate for a number of the most sophisticated algorithms.

Our recent work on using Nuprl to support aspects of scientific computation (as part of the Polya project) has revealed another connection between these systems that is quite promising. We have worked with Richard Zippel, using his algebra system Weyl as an example, and aspects of the joint work will be reported elsewhere. The new idea is that the prover, in this case Nuprl, is used in two ways:

1. to *justify* steps of algebra in an argument (assuming the algorithms are correct) and

2. to *glue* together simple steps of a proof that are a mixture of algebra, well-known theorems and trivial steps of reasoning. We use the algebra, cite formal versions of the theorems and supply the linking verification steps.

In the paper we discuss only the first kind of interaction because it concentrates on the domain of algebra and illustrates the synergism between the two kinds of systems.

## 2 Basic Definitions

### 2.1 Structure and Algebraic Structures

The Bourbaki encyclopedic volumes, *Elements of Mathematics*, reflect, indeed *created*, the modern style for presenting the basic definitions in 20th century mathematics. In the first volume, *Theory of Sets*, they lay out the concept of a *structure* (Chapter IV) and the notions now basic to algebra and universal algebra. Here is how Bourbaki put it: "... in later parts of this series we shall define the notions of *group, ring, field, topological space, uniform space*, etc., all of which are words denoting sets endowed with certain structures."

A structure consists of *principal base sets*  $C_1, \dots, C_n$ . There may also be *auxiliary base sets*,  $A_1, \dots, A_m$ . Then there are elements belonging to *types* built from the base sets and axioms specifying properties of the elements. The interesting point is that the structure consists of a finite number of objects arranged in some order.

In the second volume, *Algebra*, Chapter 1 is *Algebraic Structures*, these structures are singled out to have the form  $\langle C_1, \dots, C_n, A_1, \dots, A_m, F_1, \dots, F_k, R_1, \dots, R_p \rangle$  where the  $F_i$  are functions (or constants) over the base sets into a principal base set and the  $R_j$  are relations on the base sets. These are subject to certain laws or axioms,  $Ax_1, Ax_2, \dots, Ax_k$ .

One of the simplest algebraic structures is a *monoid*,  $\langle M, f, e \rangle$  where  $f : M \times M \rightarrow M$  and  $e : M$ . The axioms specify that  $f$  is associative and  $e$  is an identity, i.e.

$$\text{Monoid-axiom 1: } \forall x, y, z : M. \quad f(f(x, y), z) = f(x, f(y, z)).$$

$$\text{monoid-axiom 2: } \forall x : M. \quad f(e, x) = x = f(x, e).$$

[\*note on universal alg]

### 2.2 Type Theory Presentation of Structures

The concepts of modern type theory[5], [2], [8], [7] are well suited to defining algebraic structures. For example, the type of a monoid is just the dependent product

$$M : Type \times f : (M \times M \rightarrow M) \times e : M.$$

We call this the *Monoid Type*. The important point is that the type of the second and third components *depends on* the value of the triple such as

$$\langle Z, +, 0 \rangle \quad \text{for } Z \text{ the integers as carrier.}$$

We can imagine from universal algebra and category theory that we might want “large carriers”, say the collection of all sets or the collection of all monoids. In a formal type theory such as Nuprl[2], this could include monoids of large objects, e.g. a monoid of sets or types which can be written polymorphically as

$$M : U_i \times f : (M \times M \rightarrow M) \times e : M$$

where  $U_i$  is the  $i$ -th *universe* of types.

## 2.3 Constructivity Issues

### Historical Concern

There has long been serious concern in algebra for what we now call constructivity. In the late 19th century, Leopold Kronecker developed a substantial amount of field theory using strictly “finitistic” methods; the elements of a field had to be concretely represented, the operations had to be computable and the properties of elements decidable.<sup>†</sup> This line of development continues. In 1926, G. Hermann showed how to construct certain ideas explicitly, but had assumed that all fields satisfying the above conditions had the property that uncountable polynomials over them could be factored into indecomposable polynomials (such fields are said to have *splitting algorithms*). In 1930 B. L. van der Waerden introduced the notion of an *explicitly given field*, essentially one satisfying Kronecker’s conditions, and showed that not all such fields have splitting algorithms.

In the 50’s the techniques of recursive function theory were used to obtain other negative results. For example, Fröhlich & Shepherdson [3] showed that there are domains in which a ??? can be computed, but there is no factorization algorithm. This shows the inapplicability

---

<sup>†</sup>Kronecker is popularly known for the after dinner speech in which he said “God made the integers, all else is the work of man.”

of the classical theorems that an noetherian domain with geds is a unique factorization domain.

Results like those of van der Waerden, Fröhlich and Shepherdson show that results from classical algebra that claim the existence of factorizations do not guarantee the existence of algorithms to produce them. This situation abounds in all fields of classical mathematics — objects are claimed to exist which we have no way of explicitly constructing.

Mathematicians have devised many ways of keeping track of computational content, e.g. defining *explicit domain*, requiring computable operations, splitting one classical concept into many distinct constructive ones, etc. Basically they introduce more distinctions and make fewer assumptions (e.g. rejecting indiscriminate use of axioms like  $P$  or  $\neg P$ ).

Logicians have designed formal languages which help keep track of computational content systematically say following suggestions of Brouwer or Bishop. Nuprl takes advantage of these systematic methods. Our plan is to see how well they support the *practical constructivity* that arises so naturally in algebra, especially in computer algebra systems.

### Modern Example

We choose for illustration an example from Axiom of how constructivity concerns are expressed in modern algebra systems. The definition of an *integral domain* often is stated by adding to the ring axioms the condition that there are no zero divisors, i.e. there do not exist non-zero  $c, d$  in the ring carrier such that  $cd = 0$ . But axiom needs a way to build a quotient of  $a$  by  $b$  if there is one. So its integral domain condition is expressed in terms of a partial function  $exquo : R \times R \rightarrow +fail$  such that if  $\exists c : R \ bc = a$  then  $c$  is unique and  $exquo(a, b) = c$ , otherwise  $exquo(a, b) = fail$ . Our definition of an integral domain is also constructive in this way.

We want to explore in our work the extent to which the systematic treatment of computational issues inherent in constructive algebra serves the same computational needs that arise in computer algebra. To this end we will present our algebra in the “Bishop-method”. The idea is that the entire account will be *both* classically *and* constructively meaningful. The book by Mines, Richman, and Ruitenburg *A Course in Constructive Algebra* is written in this style and can be seen as a contribution to both ordinary algebra and constructive algebra.

## References

- [1] E. Clarke and X. Zhao. Analytica—an experiment in combining theorem proving and symbolic computation. In D. Kapur, editor, *11th International Conference on Automated Deduction*, Lecture Notes in Art. Int., Vol. 607, pages 761–765. Springer-Verlag, New York, 1992.
- [2] R. L. Constable et al. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, 1986.
- [3] A. Frölich and J. C. Sheperdson. Effective procedures in field theory. *Phil. Trans. Roy. Soc. Ser. A*, 248:407–432, 1955–56.
- [4] R. Jenks and R. Sutor. *Axiom: The Scientific Computation System*. Springer-Verlag, New York, 1992.
- [5] M. Löf. Constructive mathematics and computer programming. In *Proceedings of the 6th International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North Holland.
- [6] C. L. T. Mannion. Scientific computation, symbolic computation and inference. Will be a techreport soon., September 1993.
- [7] B. Nordstrom, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory*. Oxford Sciences Publication, Oxford, 1990.
- [8] S. Thompson. *Type Theory and Functional Programming*. Addison-Wesley, 1991.
- [9] R. Zippel. The Weyl computer algebra substrate. Technical Report TR 90-1077, Computer Science Dept., Cornell University, Ithaca, NY, 1990.