Very 'final' version.

The Development and Use of Variables in Mathematics and Computer Science

Meurig Beynon *Steve Russ*

Department of Computer Science, University of Warwick

Abstract

There are a wide variety of uses of variables in mathematics which we cope with in practice through conventions and tacit assumptions. Experience with computers has made us articulate, criticise and develop these assumptions much more carefully. Historically the term 'variable quantity' was introduced in the context of describing and calculating *changing quantities* which corresponded to phenomena in the observable world (eg the velocity, or fluxion, of a body moving under the inverse square law). The evolution of the concept has divorced it from these 'roots of reference' and required us to establish the formal apparatus of interpretation and valuation. While the changes considered are highly structured this may be satisfactory, but computing power invites us to cope with change in vastly more complex, unstructured situations such as in simulation of 'real world' processes. We relate this challenge to the distinctive differences in the use of variables in mathematics and practical computing, and we develop a general framework in which all uses of variables can be described in a unified way.

§1. A general framework for variables

State and the notion of primitive variable

From about 3000 BC until about 10 years ago people occupied with engineering, navigation, commerce, science or mathematics depended to a great extent on the construction and use of large tables of data. These tables formed a vital part of their capacity to organise, explain and predict changes in the world. They ranged from Babylonian tablets recording astronomical data alongside data about the weather, the frequency of earthquakes and prices of wheat, through later tables for such matters as high-tides, mortality and compound interest, up to the 4-figure logarithm tables used by generations of school-children until the late 1970's. Such systematic association of different values with the same family of symbols illustrates the use of a primitive concept of variable that will be significant in the general framework to be proposed. Such *primitive variables* are typically used for describing *state*.

There are two fundamental elements involved in the notion of a primitive variable: a symbol and the capacity for that symbol to be associated with a value. A variable (of any description) is not the name of a value, like π, since it is characteristic that different values may be associated with the same variable (for example, choosing different points on a parabola or computing successive values of a counter in a loop). It is often convenient and powerful not to associate any value with a variable but only to manipulate it in ways appropriate to its type. This capacity to be associated with different values or no value is reflected in the picture of a variable (common in computing) in which we associate with a symbol a unique 'location' which may, or may not, 'contain' a value. A primitive variable serves this function of a 'marker' for a pigeon-hole. This expresses the idea of 'capacity to be associated with a value', while also matching the way our minds seem to deal with different values being associated with concepts. For example, if x metres is the height of a projectile, when we make a program declaration, "x : real;" there is a physical memory location associated with 'x'; and when we say "as x increases, at some time it will be greater than ... ", we mean by 'it', not the symbol 'x' but the value associated with the height of the projectile and so with 'x'.

In the past, our methods for recording complex state information were very laborious, and of limited scope. A major contribution of mathematics was to provide means of simplifying the representation of state. By identifying relationships between variables, it became possible to reduce the amount of information to be recorded in order to characterise the state of a system. In this way, modern mathematics has a central interest in systems whose behaviour can be understood in terms of states in which the possible values of variables form a class: $\{v_1,v_2,v_3,.... \mid P(v_1,v_2,v_3,....)\}$, where P is a system of relationships that might include explicit definitions of one variable in terms of others, or an equational constraint governing several variables.

In mathematics an isolated primitive variable has only a minor role. The applications for which there are no relationships between variables to be recorded, either because we are unable to perceive them or because they don't exist, have traditionally been divorced from mathematics. The nearest equivalent to the mathematical equation or functional relationship may be the methodologies by which we seek to systematise cataloguing, and the theories by which we organise species. The advent of the computer has revolutionised our power to represent state. By physically constructing arrays of millions of boolean variables that can be freely assigned, we can in principle conveniently record many more states than by any other means. Through our dependence upon computer records, the whole economy of advanced nations has come to rely upon such power. A potent side-effect of practical computing has been to liberate the potentially anarchic primitive variable upon which procedural programming at every level of abstraction depends, and to challenge mathematics to discipline its use.

The uses of variables and the notion of context

In modern practice a mathematical variable is used in a wide variety of ways. It is used as an indeterminate, as an arbitrary constant or as an unknown value to be found. It is used to express relationships and functions, as a parameter to a class of problems or objects, as a dummy (bound) variable in the context of operator symbols like summation and integration, and it is used in the context of limit operations. Comparing the notion of a primitive variable with the apparently more complex mathematical variable raises the issue of whether we should be speaking of different kinds of variables as well as different uses of variables. The question will become even more pertinent when we include variables as used in spreadsheets and variables over compound data structures in both computing and mathematics.

By a *valuation* we shall mean a relation between a set of primitive variables and a set of values of appropriate types. We define a *context* as a set of primitive variables together with a means for providing their valuations. An example of a valuation is the state of a computer system or experimental environment in which the set of values have been (or at least could be) 'observed' simultaneously. In a different kind of context some or all of the values in any valuation may be related to one another (eg the co-ordinates of points on a parabola). Contexts and their valuations can be viewed in many ways and form a convenient unifying framework for discussing different uses of variables. When using procedural variables and directly assigning valuations we may be interested in the transitions from one valuation (state) to another as a result of a program statement. A mathematical use of variables in writing equations or other relationships will lead us to consider the set of all possible valuations implicitly defined by the equations. To understand the use of a variable in a problem or theory it is necessary to know how valuations can be generated in the context for that variable. This might, for example, be a matter of function evaluation, of assignment of some variables and observation of others, or by the up-dating of a spreadsheet.

It is the main claim of this paper that all the uses of variables we find in mathematics and computing can be described in terms of this general framework of a context. Given a set of variables which are being used together in some way (for the solution of a problem, or within a procedure, for example), the relevant context will consist firstly of a corresponding set of primitive variables ('corresponding' in the sense of using the same symbols). But whereas the use of the original variable might be rather complicated, the use of the corresponding primitive variable is simple - serving only to mark the association of a symbol with a value. The distinctive, possible uses of the original variables relate to the ways in which valuations of the context arise (ie valuations of the primitive variables in the context).

By way of illustration, the following references to a variable x :

x:=y+34 (procedural program), {x,y | x=y+34} (problem-solving), x=y+34 (spreadsheet)

have very different semantics, and assume different characteristics of the variable. We can nonetheless think of these diverse constructions as representing particular assertions about the value, or actions to be performed upon the value associated with x. For instance, in the context of a particular valuation of a set of primitive variables in which x occurs, x:=y+34 denotes the action of assigning to x the value of y plus 34. That is to say, a procedural variable is associated with transition when valuations are interpreted as states. In contrast, {x,y | x=y+34} is the set of possible valuations of (x,y) such that the value of x equals the value of y plus 34. Finally the spreadsheet formula x=y+34 expresses the fact that in any particular valuation the value of x is the result of adding 34 to the value of y, and that x will be updated so that this relationship still pertains if y is subsequently redefined.

The most common use of variables consists of the expression of relationships and their manipulation within the appropriate language system (eg in number theory, logic, plane geometry, functional programming); this usually depends only on the *type* of the value, not on any particular valuation. With some uses (eg in cost-analysis, weather-forecasting, procedural programming) where there are few relationships known or simple enough to be written down, it is the changing valuations that matter most. Yet other uses of variables (eg in spreadsheets) involve some characteristic combination of these requirements. All of these uses can be described in terms of a context and its valuations and the remainder of the paper is largely devoted to illustrating and elaborating this claim.

§2. Mathematical Variables and Problem-Solving

Historical development

It was not until the second half of the 16th century that developments occurred in the use of symbolic notations that could really merit description as the beginnings of algebra. François Viète (1540 - 1603) was among the first clearly to be aware of the significance of symbols that generalise numbers or magnitudes. He introduced the use of a vowel for a quantity assumed to be unknown or undetermined and consonants for quantities that are assumed to be given. The idea of 'given' quantities being represented by a letter is the beginning of the modern role of variables as 'placeholders' for values. Thus Boyer remarks, "Here we find for the first time in algebra a clear-cut distinction between the important concept of a parameter and the idea of an unknown quantity" [10]. In 1636 Fermat (who had studied Viète) was writing equations like :

A *in* E *aeq* Z *pl.*

The Latin '*in*' means 'times' and '*pl*' is an abbreviation for 'planus'.) He showed that the corresponding locus is a hyperbola (cf $xy = c^2$)[12]. Viète and Fermat (at this time) still regarded the magnitudes associated with letters in a geometrical way so that a product was an area etc.

Descartes, in *La Geometrie* [11], introduced a radically new vision of mathematics in which he regarded all quantities to be of the same one-dimensional kind (so capable of taking the same range of values). Even when a solution to an equation was obtained geometrically Descartes used Viète's conventions to distinguish coefficients and variables, and to express the solution as a general algebraic solution to a whole class of equations. Throughout the work of Descartes and Fermat there is evidence of a major movement from the primarily geometric mode of thought which was widespread at the beginning of the 17th century to an algebraic mode of thought which was becoming dominant in the outlook of most of the leading mathematicians by the end of that century. The increasing use and understanding of variables over this century was a major factor in making possible the much greater generality and algorithmic power of the algebraic approach.

The rapid progress of analytical geometry led to many authors by the mid-17th century dealing with higher order curves (cubics etc) and transcendental (ie non-algebraic) curves such as the cissoid, cycloid and quadratrix. Newton studied *La Geometrie* in 1664 and mastered the algebraic methods there thoroughly although this in no way affected the fact that his outlook was always primarily geometric. He also had no hesitation about the incorporation of motion into geometry and mathematical methods. The independent development of the calculus by Newton in 1666 [15] and by Leibniz in 1684 [14] illustrated very well the fact that profound ideas and methods could be expressed in both geometric and algebraic patterns of thought. This has significance for our theme in that, "Newton considered variables as *changing in time*. He applied concepts of motion; the variables were considered as flowing quantities.... In modern terminology one might say that Newton considered all variables as functions of time..... Leibniz considered variables as *ranging over sequences of infinitely close values*. In his calculus there is little use of concepts of motion." [2] Newton in fact seems to have taken velocity as a primitive concept. It is likely to have been the prevalence of such kinematic ideas which promoted the use of the term 'variable quantity' (and hence 'variable'). For example, here are some samples of Newton's language suggesting that he has in mind an abstraction from the variable quantities which we can observe:

" I consider mathematical quantities in this place not as consisting of very small parts; but as described by a continued motion. ......Therefore considering that quantities, which increase in equal times, and by increasing are generated, become greater or less according to the greater or less velocity with which they increase and are generated; ...... calling these velocities of the motions or increments *fluxions*, and the generated quantities *fluents*. ....Let a, b, c, d, &c. be determinate and invariable quantities, and let any equation be proposed involving the flowing quantities z, y, x, &c. as $x^3 - xy^2 + a^2z - b^3 = 0$. ...." [16].

While there was no analytic concept of function (for Newton a function was identified with a curve), the notion of variable was, in at least some of its uses, playing the role of a function [9]. For example, the use of variables to express a dependency relation between the abscissae and ordinates of points on a curve was in many cases expressing a functional relationship. The 'fluents' used by Newton can usually be thought of (in modern terms) as functions of time. So the history of the notion of a mathematical variable is inextricably linked with the history of the function concept [21]. Not surprisingly the association of time and motion with the idea of a variable quantity and a function led throughout the 18th century to physical intuitions of dynamic behaviour obscuring the possibility of an arithmetic concept of the continuity of a function. The same sort of problems arose with the convergence of infinite series which the operational freedom of algebraic symbolisms made available long before they were understood. (Already in 1666 Newton was using infinite series to express areas under various curves [17].) It was not until 1817 that Bolzano published a paper ([8],[18]) in which the modern concepts of convergence and continuity are made clear and used for the first time. An essential ingredient of Bolzano's work is the clear understanding of the need for two distinct variables to define each of these concepts and the significance of arbitrary choice for the way values are assigned to one of them - the 'ε' of usual modern forms - further details appear in [19]. At this stage the dynamic concept of variable as a near-relation of varying quantities has been clearly replaced by a variable for which the changes in valuation arise in a precise context and in a specifiable manner. The mathematical variable had by now 'come of age' and was being used fluently as a placeholder and vehicle for expressing relationships of a quite abstract nature.

Problem-solving and the 'such-that' construction

Historically, variables have been closely associated with problems of finding unknowns from givens. The study of such problems has been so influential in the development of mathematics that it is tempting to regard all mathematics as a problem-solving activity. Computer science has in some respects reinforced this view. A large body of theoretical computer science is concerned with the solution of algorithmic problems that take the form of finding unknowns from givens. Declarative approaches to programming can be seen as an attempt to cast all computation into this paradigm.

A typical use of variables in formulating a mathematical problem requires a logical framework established by naming function and predicate symbols defined over a domain, or a sorted family of domains. The constraints of the problem to be solved are expressed by means of a system S of well-formed formulae containing free variables $v_1, v_2, v_3, \ldots.$. On fixing an interpretation of the domains, and the function and predicate symbols, the system S implicitly defines a context (set of valuations) which is the set of sorted values that can be substituted for the free variables so as to satisfy all the formulae in S:

$S(v_1, v_2, v_3, \ldots.) \equiv \{ v_1, v_2, v_3, \ldots. \mid \text{for all } P \text{ in } S\text{: } P(v_1, v_2, v_3, \ldots.) \}$.

A set of valuations generated implicitly in this way is what we mean by the 'such-that' construction and it serves to represent a very general class of problems. It is important to note that the use of variables as a means of representing all possible valuations does not in itself distinguish one valuation from another, nor does it describe any relationship between valuations. In this context, the class $S(v_1, v_2, v_3, \ldots.)$ is being used to distinguish two kinds of valuation: viz those that satisfy the system of formulae S, and those that do not. In describing a problem in this fashion, some of the free variables used in specification of the problem take the form of parameters to be specified for each instance of the problem, and the rest are the variables to be instantiated appropriately.

§3. Variables in computer science

Problem-solving versus modelling and simulation

The computer is a physical system with a superhuman capacity to represent state. Each memory location corresponds to a primitive variable, and each assignment of values to memory locations defines a valuation, or state. The set comprising all such valuations is one possible context within which a computation may be viewed as taking place. The primitive operations that the processor can perform are the means by which transitions occur, and a program essentially comprises a prescribed sequence of such transitions.

There are two principal ways in which a computer can be used. We can exploit the power of computers to perform routine computation fast and accurately when following a particular recipe for the solution of a given problem. We can also use a computer to record state information. These uses correspond to two quite different ways in which contexts (in the technical sense of §1) are typically specified. In problem-solving, a system of relations between variables is used to define a set of valuations implicitly. In modelling physical systems with which we interact, observed or specified system states are represented by sets of valuations of primitive variables. The former is associated with mathematical variables as they occur in the 'such that' construction, and describes a context declaratively. The latter is associated with the assignment to primitive variables, and determines a context procedurally. In a declarative program, there are explicit relationships between variables, but only implicit valuations. In a procedural program, the valuations are explicit, and the relationships between variable values implicit.

The potential conflict between the declarative formulation of a problem, and the procedural nature of its computer solution is evident. A most striking aspect of problem-solving by computer is the manner in which computation has to be expressed using variables whose values can be freely assigned, and can only be constrained implicitly by imposing an order upon the procedures that manipulate them. The relationship between these valuations within the machine and problem-solving at the highest level of abstraction may be very obscure. For example, there is no reason to suppose that a valuation of the variables significant in the problem should be encoded in any intermediate state of the computation. The principal difficulty in describing a computer solution to a problem can be seen as identifying the states through which the computation must pass in the construction process. The need to unify a procedural and a declarative use of variables has led to the development of methods of describing and manipulating values that we can interpret as novel uses of variables within the framework introduced above. For example, there is spreadsheet programming and definition-based programming as described below.

Procedural versus declarative programming

Procedural use of variables serves as a direct way to capture state. Procedural methods reflect the manner in which state is manipulated in a conventional computer, but their adoption has provoked controversy. Some extreme criticism is made of the concept of variables that are given different valuations at different times within the same program. For the advocates of pure declarative programming: "variables always denote the same quantity, provided we remain in the same context" [7]. This concept of referential transparency seems to involve a restrictive interpretation of the term "context" (viz that supplied by a specific fixed valuation), rather than a legitimate interpretation of the usual concept of variable. After all, the semantics of variables is determined not by a single chosen valuation, but by a set of possible valuations.

Procedural methods are characterised by the explicit manipulation of valuations. The simplest procedural approach to problem-solving can be viewed as manipulating the valuations of the characteristic variables of the problem directly, as when enumerating the possible set of valuations in solving a combinatorial problem such as the 8 Queens, or computing integers P and Q such that Pp+Qq=gcd(p,q) by the extension of Euclid's greatest common divisor algorithm. In such programs we can think of the characteristic variables of the problem as corresponding abstractly to literal machine locations that are set to initial values and attain appropriate values on termination.

Declarative programming methods have evolved in response to the difficulties of representing relationships in a procedural framework, especially in connection with problem-solving. The most direct way to support mathematical problem solving on a computer is to develop a programming system that permits the specification of relationships between variables and as far as possible automates the extraction of satisfying valuations. As explained in detail by Kowalski [13], logic programming aims to automate problem-solving in just this way.

An alternative declarative approach is adopted by functional programming. Its underlying philosophy is that all computation is equivalent to the evaluation of an expression over a rich and sophisticated algebra, and that program development entails the formulation and manipulation of such expressions. In pure functional programming, explicit functional relationships between variables are established by a script that determines the environment for all variable evaluation.

Pure functional programming systems most effectively eliminate state. In a referentially transparent framework, there is no scope for using valuations for state information; supplying alternative parameters has no side-effect upon the environment. Though variables representing higher-order functions can express very subtle parametric systems, the limitations of a purely functional approach where state information is concerned stem from adherence to a single environment. Several methods of capturing state have been proposed (cf. Backus [1]). Some involve the encapsulation of history in values. As studied in data flow models of computation, this involves variables that are in certain respects analogous to Newton's flowing variables (cf. Lucid [20]).

A definition-based approach to programming

We shall briefly consider a novel approach to programming which combines characteristics of both procedural and declarative styles and that has been under development at the University of Warwick over several years. This work was originally concerned with interaction between the user and computer, but has developed in many different directions. A central theme in this research is the representation of state as it appears to an agent participating in a computation through a use of variables that integrates both declarative and procedural characteristics. An extended exposition is beyond the scope of this paper; we shall merely identify the main points of connection with the theme of this paper. For more details, the interested reader may consult several other references (such as [3], [4],[5],[6]).

The central abstraction in our approach ("definitive = definition-based programming") is variable definition. As a simple example, the integer variable a may be defined by the formula "a = b+c". This is to be interpreted as saying that the value of a is specified as the sum of the values of the variables b and c. Such definitions of variables resemble those of a functional programming script; but definitive programming differs from functional programming in that redefinition of a variable is a legitimate action within the programming paradigm. If for instance the value of the variable b is incremented, the value of a is incremented by the same amount. As in functional programming, some procedural action is regarded as invisible, viz the mechanisms by which the value of a variables is guaranteed to be consistent with its definition at all times. This automatic updating of variables according to their definitions is the essential principle underlying the spreadsheet.

The use of variables in definitive programming has many points of contact with the other paradigms discussed above. Of particular interest is the way in which a definitive program can effectively "edit its own script". From the perspective of this paper, this integrates the manipulation of relationships that apply to the value of a variable and changing valuations. This feature is particularly important in applications such as design, where the relationships between variables must be fluid and easily modified within the programming system [6].

§4. Conclusion

We have argued that the many uses of variables can be described in terms of an underlying concept of primitive variable that serves to establish an association between symbol and value. This association has to be interpreted with reference to a context defined by a set of valuations of a family of primitive variables. Such contexts are familiar as models of physical systems that we observe and manipulate. Contexts provide the cognitive basis for a variety of abstractions, and their description and manipulation corresponds closely to different uses of variables. Each method of using variables deals with the set of valuations of a set of primitive variables in a characteristic way. In some cases, it is more natural to think of each valuation as a state, and to consider possible transitions between one state and another. In other cases, the entire set of valuations - rather than the individual valuations - has primary significance. Reference to values through their symbolic variable names is required in whatever manner we choose to address the set of valuations. The semantics of the variable in each case is derived from the nature of the assertions being made about values, or the actions being performed upon them.

We have seen how the use of variables in mathematics illustrates many different ways in which contexts are described and manipulated. In mathematical argument, or in the process of constructing a solution to an algorithmic problem, we may focus attention upon a particular valuation, or construct sequences of valuations. In modern computing, the importance of state-transition models of computation has revived interest in the explicit manipulation of valuations. The need to develop programming paradigms that are well-suited to problem-solving, and to modelling and simulation, has forced us to re-examine the relationship between state-based and relational views of primitive variables in a context, and led us to seek a more satisfactory synthesis of the procedural and declarative computational perspectives they respectively represent. In particular, it appears that more powerful methods of dealing with sets of valuations as states and state transitions are required, and that it is necessary to consider when, and by what agents, state changes are enabled. We have proposed the application of "definitive methods", a generalisation of the principle underlying the use of variables in a spreadsheet, as a way of addressing some of these problems.

References

[1] Backus, J., "Can programming be liberated from the Von Neumann style?" CACM, 21,8 (August), 613-641,1978.

[2] Baron, M.E., and Bos, H.J.M., "History of Mathematics : Origins and Development of the Calculus 3" , Open University Unit AM289 C3, 1974, p.55

[3] Beynon, W.M., "Definitive principles for interactive graphics", NATO ASI Series F: 140, 1083 - 1097, 1988.

[4] Beynon, W.M., Norris, M.T. and Slade, M.D., "Definitions for modelling and simulating concurrent systems" Proceedings IASTED Conference ASM 1988, Acta Press, 1988.

[5] Beynon, W.M., Slade, M. and Yung, Y.W., "Parallel Computation in Definitive Models", Proceedings CONPAR'88 (to appear).

[6] Beynon, W.M., "Definitive programming as a framework for design", Preliminary Proceedings Third Eurographics Workshop on Intelligent CAD Systems CWI 1989.

[7] Bird, R.and Wadler, P., "Introduction to Functional Programming", Prentice-Hall, 1988,p.4.

[8] Bolzano, B., "Rein analytischer Beweis des Lehrsatzes...." , Prague, 1817.

[9] Bos, H.J.M., "Differentials, Higher-Order Differentials and the Derivative in the Leibnizian Calculus", Archive for History of Exact Sciences, 14, 1974, Sec.1.

[10] Boyer, C., "A History of Mathematics", Wiley, 1968, p.335.

[11] Descartes, R., "La Geometrie", 1637; tr. Smith, D.E. and Latham, M.L., "The Geometry of Rene Descartes", Dover, 1954, p.13.

[12] Fermat, P., "Isagoge ad locos planos et solidos", 1679 [written 1636].

[13] Kowalski, R., "Logic for Problem Solving", North-Holland, 1979.

[14] Leibniz, G., "Nova methodus pro maximus et minimis ...... ", Acta Eruditorum, 1684.

[15] Newton, I., "De Analysi per Aequationes Numero Terminorum Infinitas", 1669.

[16] Newton, I., "Tractatus de quadratura curvarum" in Opticks , 1704 , tr. Struik , D.J., "A Sourcebook in Mathematics, 1200-1800 ", Harvard University Press, 1969, p.303.

[17] Newton, I., "Epistola Posterior", 1676, in Fauvel, J. and Gray, J. (ed.) "The History of Mathematics : A Reader", Macmillan and Open University, 1987, p.404.

[18] Russ, S.B., "A Translation of Bolzano's Paper on the Intermediate Value Theorem" Historia Mathematica 7, 156 - 185, 1980.

[19] Russ, S.B., "The Mathematical Works of Bernard Bolzano published between 1804 and 1817", Open University, 1980, unpublished Ph.D thesis, Ch.4.

[20] Wadge, W., Ashcroft, E.A., "Lucid - the dataflow programming language", Academic Press, 1985.

[21] Youshkevitch, A.P., "The Concept of Function up to the Middle of the 19th century", Archive for History of Exact Sciences, 16, 1976.