The Significance of Distinguishing Functions from Algebraic Equations

Steven Bryant
Avitel Corporation / RelativityChallenge.com
www.RelativityChallenge.com
Steven.Bryant.Email@RelativityChallenge.com

The mathematical mistreatment of functions as algebraic equations results in errors that are impossible to detect without a thorough understanding of scope and namespaces. Algebra defines what variables are, how they interact with other variables to form equations, and how multiple equations can be combined or solved as a system of equations. Functions operate in a similar way, but introduce a new layer of complexity due to the abstraction that occurs by separating function definition from function invocation. Here we show that functions are different from algebraic equations and that, under certain circumstances, their mistreatment as an algebraic equation will result in nearly impossible to detect mathematical errors. Explaining these differences will require the introduction of scope and namespaces, which are cornerstone concepts in the proper treatment of functions. Within this context, we explain the differences between optimization and simplification, and between invocation and substitution.

Keywords: Functions, Function Syntax, Namespaces, Scope

Introduction

Algebra is one of the foundational subjects within mathematics. It defines variables and introduces rules for using them in equations. Functions are introduced in calculus and are often used synonymously as equations [1,2]. While functions are formally defined as a specific type of relation [3,4,5], mathematics does not provide sufficient detail around function syntax and operation to distinguish them from their algebraic counterparts. This can result in subtle, difficult to find, mathematical mistakes [6,7]. For example, consider the equation

$$y = x^3 + 2x + 7\,. \qquad \text{Eq. 1}$$

In calculus, one could describe this equation by saying "y *is a function of* x," leading one to rewrite the equation as the function

$$f(x) = x^3 + 2x + 7\,, \qquad \text{Eq. 2}$$

which they then treat as a substitute for Eq. 1. The problem is that although Eq. 1 and Eq. 2 may look similar, they are not the same thing. This point is illustrated by introducing the function definition

$$f(a) = a^3 + 2a + 7\,, \qquad \text{Eq. 3}$$

which is equivalent to Eq. 2. Although Eq. 3 is equivalent to Eq. 2, many would conclude that Eq. 3 is not the same thing as Eq. 1 simply because it uses a different variable in the function body. Since a function definition is simply a template, it must be *invoked* before it can be used. Thus if Eq. 2, or equivalently, Eq. 3, is invoked as

$$y = f(x)\,, \qquad \text{Eq. 4}$$

each function will produce the equation

$$y = x^3 + 2x + 7\,. \qquad \text{Eq. 5}$$

So, although Eq. 2 may look similar to Eq. 1, it would be premature to say it is the same thing as Eq. 1 based solely on the function definition. Similarly, it is only after a specific function invocation that we are able to say that Eq. 3 yields the same thing as Eq. 1. *Only* *after a function's invocation are we able to determine if a function is equivalent to an algebraic equation.* Additionally, functions can be transformed into new equations in ways not possible with algebraic equations. For example, if Eqs. 2 and 3 are *invoked* as $y = f(x^2)$, they produce

$$y = x^6 + 2x^2 + 7\ . \qquad \text{Eq. 6}$$

This equation cannot be produced from Eq. 1 since the required *substitution*, $x = x^2$, is not always true. These subtleties, which are often overlooked when dealing with functions, can result in extremely difficult to locate mathematical mistakes.

This paper explains the distinguishing characteristics between functions and equations, as well as identifies problems that can occur when functions are mistreated as equations. Specifically, it will discuss the differences between an algebraic *equation* and a function *definition*, and between algebraic *substitution* and function *invocation*. These differences will require us to introduce the concepts of *scope* and *namespaces*.

Function Definitions

When an algebraic equation is defined, it is immediately ready for use, as long as values can be found for each of the named variables. This is not true with functions. A function definition is essentially a template that does not take on the final form of an equation until after invocation. For example, if an equation is written as

$$y = 2a + b\ , \qquad \text{Eq. 7}$$

we can find the result, y, as long as we can determine values for a and b. Thus, the equation is ready to be solved or to take part in a system of equations. Functions, on the other hand, make a distinction between their definition and their usage. Consider the following four functions,

$$f() = \{2a + b\} \tag{Eq. 8}$$

$$f(a) = \{2a + b\} \tag{Eq. 9}$$

$$f(b) = \{2a + b\} \tag{Eq. 10}$$

$$f(a,b) = \{2a + b\}. \tag{Eq. 11}$$

Each function is different and uses the $\{\ \}$ notation to show that each is a function definition, distinguishing them from an algebraic equation, or from an invoked function. Like Eq. 7, each function definition contains $2a + b$ in the function body. As will be explained shortly, only the function definition given in Eq. 8 will produce the same equation as given in Eq. 7 for every invocation of the function. The remaining functions <u>can</u> produce different equations than Eq. 7, depending upon the argument(s) passed to the function during invocation.

Structurally, a function definition has three parts; the function name (e.g., f), the function body (e.g., the part between the $\{\ \}$), and the function signature, which is the function name and any variables used as parameters by the function (e.g., $f(),\ f(a),\ f(b),\ f(a,b)$). Variables in the function signature are called parameters, or local variables. They are "placeholders" that must be resolved before the function can be evaluated. Local variables introduce a new level of abstraction, beyond that which is available in algebra, due to the concept of namespace.

Scope and Namespaces

In algebra, all variables exist within the same namespace, the g*lobal namespace*. In other words, algebra makes no distinction between a *global* and a *local variable*. Because algebra simply uses one type of *variable*, the concepts of scope and namespace can be safely ignored and are generally overlooked. These concepts cannot be ignored when working with functions.

A namespace is the name given to describe where any particular variable is valid (also known as its scope) [8,9,10]. Variables defined outside of a function exist in the global namespace and use the notation $::globalVariable$. Because there is no name prior to the $::$ symbol, these variables can be used anyplace, unless they are overridden by a locally scoped variable of the same name. Global variables are said to have universal scope.

Assuming a non-nested function, the namespace notation for a local parameter is $functionName::localVariable$. Variables that begin with $functionName::$ are only visible and can only be used within that function. Because they can only be used within the function, local variables are said to have limited scope. When a local variable is defined, it will hide any variables of the same name from a higher scope (e.g., global variables) until after the function has been invoked. In other words, locally scoped variables will always take precedence in a function definition.

The simple act of defining a function creates “local variables” if that function has one or more variables in the function signature. Because a local variable only exists within the namespace of the function and is essentially a placeholder, it can be replaced by any other local parameter, producing an equivalent function. Thus, $f(c) = \{2c + b\}$ is equivalent to Eq. 9 and could be used in its place without error. In namespace notation, this is written as $f(c) = \{2 * f::c + ::b\}$. Prior to invocation, a function may be composed of local and global variables. Following invocation, a function will no longer contain any variables local to that function.

Care must be taken to prevent simplifying a local variable with a global variable of the same name. These are called *overloaded variables* and must be treated distinctly from one another in order to prevent very difficult to find mistakes. Such incorrect simplification, as will be shown shortly, can be nearly impossible to detect without formal namespace notation.

Abstraction – Function Invocation

Invocation is the act of replacing a local placeholder variable with the value, equation, variable, or function that is passed as an argument. This is an act of variable replacement between different namespaces. This differs from algebraic substitution which performs all of its operations within one namespace. The number of formal arguments used when the function is invoked must match the number of parameters given when the function was defined.

An invocation occurs when the function is given arguments. Typically, function invocations appear as part of statements on the right-hand side on the equals sign. For example, given a and b are both 0, consider the function invocation

$$y = f(3)\,. \qquad \text{Eq. 12}$$

If this is the invocation for Eq. 9, then the resulting equation is

$$y = 2(3) + b \quad \text{or} \quad y = 6\,. \qquad \text{Eq. 13}$$

However, if this is the invocation for Eq. 10, then the resulting equation is

$$y = 2a + 3 \quad \text{or} \quad y = 3\,. \qquad \text{Eq. 14}$$

Thus, when a function is defined, it is not ready for use. It must be invoked. If $a = 1$ and $b = 10$, consider the confusion if Eq. 11 is incorrectly used before it was properly invoked as

$$y = f(b,a) \textit{ which becomes } y = 2b + a\,. \qquad \text{Eq.15}$$

This is not equivalent to Eq. 7, nor does it look like the function definition in Eq. 11. Notice that even when values for a and b are known, the function bodies in Eqs. 9, 10, and 11 generally will not produce a correct result prior to invocation. This separation between function definition and invocation is a subtle, yet significant, distinction between algebraic equations and functions.

As a result of the separation of function definition from invocation, there are four transformational activities that can be performed on functions; optimization, invocation, substitution, and simplification. However, only two of them, substitution and simplification, are performed on algebraic equations.

Optimization is the act of simplifying a function before its invocation. Proper optimization takes into account the namespaces of the variables within the function body to ensure that local variables are not incorrectly simplified with global variables of the same name. Since optimization considers variables in more than one namespace, this activity is not performed on algebraic equations.

Invocation is a required activity when using a function. Invocation maps arguments into the local parameters of the function. This mapping of arguments (from a higher-scoped namespace) to parameters (of the local namespace) is unique to function invocations and does not occur with algebraic equations.

Substitution is similar to invocation in that variables are replaced with literals, global variables, or other equations. With substitution, all of the replacements occur within the same namespace. This is a key differentiator between substitution and invocation.

Simplification is similar to optimization. It can occur with equations, or with functions after invocation. The key difference between substitution and optimization is that simplification is performed on variables within the same namespace; most typically the global namespace.

Function Syntax

One of the reasons that functions are mistreated as algebraic equations is because they are often written so that they look like equations. This makes it hard to know when the activities of functions (e.g., optimization, invocation, substitution, and simplification) apply as opposed to the activities of algebra (e.g., substitution and simplification). In order to avoid problems, functions should adhere to a formal style for both definition and invocation.

Some familiarity of syntax notation is assumed. The symbol $\rightarrow$ is read as "is a." Optional items are enclosed in brackets []. The symbol '|' is used to denote a choice between two or more alternatives. $void$ is a reserved word and should not be used as a variable name. The rule that defines a valid function definition is given by the following specification:

$$\begin{aligned}
&functionDefinition \rightarrow [Type]\ functionSignature\ \ functionBody\\
&functionSignature \rightarrow\ functionName \mid\ functionName(\,parameterList_{opt}\,)\\
&functionBody \rightarrow [=]\,\{\,functionStatement\} \mid =\ functionStatement\\
&parameterList_{opt}\ \rightarrow localParameters \mid void \mid \varepsilon\\
&localParameters \rightarrow localParameters,\, localParameter \mid localParameter\\
&localParameter \rightarrow [Type]\,[namespacePath ::]\,localVariable
\end{aligned}$$

It may not always be convenient to show a function definition with all of its parameters. In such cases, the "…" notation should be used, such as $f(...)$. This will distinguish a function that will take arguments during invocation from those that will not.

Use of $functionSignature \rightarrow\ functionName$ is discouraged since it can lead to confusion and the potential mistreatment of a function as an algebraic variable. Use of the $\{\ \}$ notation is recommended when defining functions in order to better distinguish them from algebraic equations. This will also serve as a reminder that the function must be invoked before it is used.

The rule that defines a valid function invocation is defined by the following specification

$$\begin{aligned}
&functionInvocation \rightarrow functionName \mid functionName(\, formalArgumentsList_{opt}\,) \\
&formalArgumentsList_{opt} \rightarrow formalArguments \mid void \mid \varepsilon \\
&formalArguments \rightarrow formalArguments,\ formalArgument \mid formalArgument
\end{aligned}$$

When invoking a function that takes no arguments, it is better to show the function invocation as $f()$ or $f(void)$, rather than as f. This will help prevent the mistreatment of the function as an algebraic variable.

One should avoid using a function definition and a function invocation as part of the same statement (e.g., $y = f(x) = 2x + 7$*).* This combining of definition with invocation can result in namespace confusion between global and local variables and may produce difficult to locate errors.

While these specifications are sufficient to cover most functions definitions and invocations one may encounter in mathematics, it should not be taken as a complete language specification for all functions. In some domains, such as Computer Science, such language specifications undergo many revisions and are ultimately standardized through a recognized governing body such as IEEE or ANSI. I recommend that any language specification for function usage in mathematics undergo a similar process.

Nuances of Function Use

The following example will illustrate the nuances between a system of algebraic equations and a system that consists of an algebraic equation and a function. Consider the following two algebraic statements,

$$x' = x - vt \qquad \text{Eq. 16}$$

$$T = t - vx'/(c^2 - v^2)\,. \qquad \text{Eq. 17}$$

T can be expressed in terms of x and t by substituting for x' in Eq. 17 with Eq. 16. In this case, v and c are treated as givens, or constants. This produces

$$T = t - v(x - vt)/(c^2 - v^2)\,, \qquad \text{Eq.18}$$

which can be further simplified as

$$T = (t - vx/c^2)/(1 - v^2/c^2) \qquad \text{Eq. 19}$$

by canceling the $v^2 t$ terms resulting from rewriting Eq. 18. As a second example, we can introduce the function T that uses no parameters. The function $T(void)$, or $T()$, is *defined* as

$$T() = \{t - vx'/(c^2 - v^2)\} \qquad \text{Eq. 20}$$

and is *optimized* as the function

$$T() = \{(t - vx/c^2)/(1 - v^2/c^2)\}. \qquad \text{Eq. 21}$$

In this case, each of the variables involved in the function is a global variable. While the function should still be thought of as requiring an invocation, in reality, the function body remains the same after invocation since there are no local variables to be replaced. However, care must be taken if the function is defined with local parameters, especially if those parameters share the same variable name as a global variable. As a third example, consider a function $T(...)$ that is defined using the local variable t. Notice what happens if we *optimize* this function as we did in the previous two examples. We begin with

$$T(t) = \{t - vx'/(c^2 - v^2)\}. \qquad \text{Eq. 22}$$

Since x' is a global variable in the function, we replace it with Eq. 16 to produce

$$T(t) = \{t - v(x - vt)/(c^2 - v^2)\}, \qquad \text{Eq. 23}$$

which is now ambiguous and misleading because it is not readily apparent that the two t variables in the equation are, in fact, different variables. Without careful identification of the global and local variables, as can be performed using namespace notation, as in

$$T(t) = \{T::t - ::v\ (::x - ::v * ::t)\ /\ (::c^2 - ::v^2)\} \qquad \text{Eq. 24}$$

one would incorrectly *simplify* Eq. 23 to produce

$$T(t) = \{(t - vx/c^2)/(1 - v^2/c^2)\}. \qquad \text{Eq. 25}$$

I must emphasize that Eq. 25 is incorrect since it involved the cancelation of a term containing the local placeholder variable $T::t$ with a term containing the global variable $::t$. The t variables are different because they are in different namespaces.

As a final example, reconsider Eq. 11, which when invoked as $y = f(b,a)$ yields $y = 2b + a$. Producing this equation from Eq. 7 using algebraic substitution would be impossible without first introducing a third, temporary, variable. However, understanding why this invocation works the way it does, without error, is illustrated using namespace notation. Using this notation, the function is defined as

$$f(a,b) = \{2 * f::a + f::b\}, \qquad \text{Eq. 26}$$

highlighting the use of two local placeholder variables in the function. When the function is invoked as

$$y = f(b, a) , \qquad \text{Eq. 27}$$

it uses variables from the global namespace as arguments. This is made more apparent by showing the invocation using namespace notation as

$$y = f(::b, ::a) . \qquad \text{Eq. 28}$$

Thus after invocation, the resulting equation, when written in namespace terminology is

$$y = 2 * ::b + ::a . \qquad \text{Eq. 29}$$

This example illustrates how Eq. 29 is produced and how namespace notation differentiates functions (which can use variables from multiple namespaces) from algebraic equations (which typically work only with variables within the global namespace). The reader is asked to rewrite Eqs. 8 through 10 in namespace notation to better understand the differences between each.

These examples have illustrated nuances that must be considered when dealing with functions in order to prevent mistakes that one would not realized were there if assessed solely from an algebraic perspective [6,7].

Conclusion

Functions differ from their algebraic counterpart with regard to scope and namespaces. Variables in algebraic equations exist in the global namespace. For this reason, concepts such as scope and namespaces can be safely ignored when dealing solely with algebraic equations. However, any function that is defined with a local variable in the function's signature automatically creates a new namespace. This means that functions can use local variables, global variables, or both. This distinction is critical to understanding the proper operation of functions since functions may have *overloaded variables* (e.g., variables of the same name in multiple namespaces).

Unlike algebraic equations, functions consist of two actions; definition and invocation. As a result, functions can have four transformational activities performed on them; optimization, invocation (which is required), substitution, and simplification. Algebraic equations, on the other hand, only have two activities; substitution and simplification. While functions can be treated as equations, often without problem, there are differences between algebraic equations and functions that can lead to difficult (or impossible) to detect mathematical errors. Without an understanding of the differences between functions and algebraic equations, extremely subtle mathematical errors will go undetected.

References

[1] E. Swokowski, "Calculus with Analytic Geometry", Alternate Edition, PWS Publishers, (1983)

[2] J. Stewart, "Calculus – Early Transcendentals", 6th edition, Brook/Cole – Thompson Learning, (2008)

[3] E. Block, "Proofs and Fundamentals – A First Course in Abstract Mathematics", Birkhauser, (2003)

[4] J. Tremblay and R. Manohar, "Discrete Mathematical Structures with Applications to Computer Science", McGraw-Hill, (1975)

[5] J. Matousek and J. Nesetril, "Invitation to Discrete Mathematics", 2nd Edition, Oxford University Press, (2008)

[6] S. Bryant, "*Mistake Identification – Function Method (Advance Method)*", (2009), Page posting available at www.RelativityChallenge.com

[7] S. Bryant, "*A Brute-Force Mathematical Challenge to Special Relativity*", 14th Annual NPA 2007 Conference Proceedings, University of Connecticut, (2008), Available at www.RelativityChallenge.com

[8] A. Aho, M. Lam, R.Sethi, “Compilers, Principles, Techniques, & Tools”, 2nd edition, Addison Wesley, (2006)

[9] B. Stroustrup, “The C++ Programming Language”, 3rd edition, Addison Wesley, (1997)

[10] R. Lischner, “C++ In A Nutshell”, 1st edition, O’Reilly, (2003)