# The Unreasonable Effectiveness of (Basic High School) Mathematics

Dominic Orchard
dominic.orchard@cl.cam.ac.uk

It may have been Einstein, it may have been Feynman, but whoever said "*if you can't explain it simply you don't understand it well enough*" really laid down the gauntlet for us theoreticians. Whilst sometimes excruciating, I have found that this process of simplifying and explaining greatly benefits my understanding of my own work and its place within the wider research-context. This April I had the fantastic opportunity to once again put myself through this process at the $5^{th}$ annual Jesus College Graduate Conference.

Primarily, my research concerns the design of *programming languages* for creating reliable and efficient software. In contrast with natural languages, such as English, Latin, etc., programming languages are highly constrained and contain much less ambiguity. They allow programmers to define complex calculations and manipulations of data which are then translated into more simple, prosaic instructions understood by computer hardware. A significant aspect of my research consists of defining the *semantics* of programming languages using abstract mathematics. A formal and precise semantics allows programmers to more accurately reason about the correctness of their programs, which is especially important in safety-critical systems, such as medical devices and transport systems.

Faced with just fifteen minutes of air-time I wondered how I could explain *any* part of my fairly theoretical research in programming languages. How could I provide any intuition about what any of it means, particularly the mathematics? Fortuitously, many concepts in mathematics have the remarkable habit of appearing frequently in a variety of different contexts. Even more fortuitously for me, the underlying mathematical concepts of my research appear in a dizzyingly large number of areas, not least of which is in basic mathematics. The concept is remarkably simple and unknowingly embedded in the minds of most people with even basic arithmetic knowledge, yet it underpins a vast number of topics in mathematics, logic, philosophy, physics, and computer science.

Consider the following simple sums:

$$2+2=4 \qquad 3+0=3 \qquad 0+7=7 \qquad 0+0=0$$

[0]Appeared in the Jesus College Annual Report 2012, Cambridge, UK

The last three have a common form: $x + 0 = x$ or $0 + x = x$. These are two simple *axioms* of integer arithmetic, where a *variable* $x$ acts as a *place-holder* meaning "any number".

Consider now the following sum of three numbers:

$$3 + 4 + 2 = 9$$

The result can be calculated by adding together one pair of numbers at a time, in two different ways:

$$\begin{aligned} &1) \quad (3 + 4) + 2 = 7 + 2 = 9 \\ &2) \quad 3 + (4 + 2) = 3 + 6 = 9 \end{aligned}$$

where parentheses delimit addition of two numbers. Clearly it is irrelevant which pair of integers is added first; the final result is the same. This behaviour can be generalised as the axiom: $(x + y) + z = x + (y + z)$.

In a large variety of contexts there exist operations similar to addition which have axioms of the same form as the three axioms for integer addition shown here. These axioms are generalised to arbitrary operations and domains by the concept of a *monoid*.

A *monoid* comprises a collection of items, an operation $\oplus$ that from any two such items calculates another item, and a special item $\mathbf{n}$, called the *null* item, along with the following three axioms:

$$\begin{array}{lcll} x \oplus \mathbf{n} & = & x & \{\textit{right-null}\} \\ \mathbf{n} \oplus x & = & x & \{\textit{left-null}\} \\ (x \oplus y) \oplus z & = & x \oplus (y \oplus z) & \{\textit{associativity}\} \end{array} \tag{1}$$

The intuition for the null item is that it contributes nothing to the result of $\oplus$. For integer addition the collection of items are integers, the operation is $+$ and the null item is $0$. Another monoid you will be familiar with is integer multiplication with operation $\times$ and null item $1$ where, for example, the *right-null* property is thus $x \times 1 = x$. You might like to convince yourself of the truth of the remaining axioms.

There are many operations in mathematics that satisfy these axioms but there are some which do not; monoids are not trivially applicable. For example, integer subtraction has no null item that satisfies the *left-null* property and subtraction violates *associativity* e.g. $(2 - 3) - 4 \neq 2 - (3 - 4)$.

While monoids are relatively basic they are important in many systems, forming the basis for *simplification*, usually in conjunction with additional axioms or properties. For example, consider the property of addition that $x + (-x) = 0$, describing simplification of addition of a number to its negation. Given a longer sum, e.g. $x + y + (-y) + z$, this property can be applied to simplify the sum to $x + 0 + z$. Because $+$ on integers with $0$ is a monoid, the sum can be further simplified to $x + z$.

Monoids appear in many contexts other than arithmetic. A fun example I demonstrated live at the Jesus Graduate Conference is the monoid of *paint mixing*. Consider a collection of acrylic paints. There is an operation that takes two colours and calculates a new colour: *mixing them*! Mixing is associative: given three paints they can be mixed to the same colour by first mixing an arbitrary pair of paints and then mixing the remaining unmixed paint. But what is the null item? White acrylic paint is *not* null, it only lightens the hue of the paint. Instead a substance called *base extender* is the null item. Base extender adjusts the drying time and consistency of acrylic paint whilst *preserving its colour*. Thus in terms of the colour, base extender is the null item of the mixing operation, satisfying the left-null and right-null properties; it is a monoid! *Do* try this at home.

In my work with semantics, programs written in some programming language are described as abstract mathematical objects, representing programs or fragments of programs. These objects have a monoid structure whose operation combines two programs to form another program, similar to constructing a sentence from sub-sentences and clauses in natural languages. The null item for this monoid represents a program that does nothing. The monoid axioms form the basis for reasoning about the correctness of the programs and correctness of the translation between the language and the underlying instructions of some computer hardware. Thus, when developing a new semantics a key question is: what is the monoid for this semantics? This *monoidal* requirement often helps determine the mathematical components needed for a semantics of the particular language.

The properties of a monoid are certainly not the only interesting properties that arise in many different contexts, but for a wide variety of contexts they appear as basic properties – and they really are very simple! After all, you already learnt them when you were at high-school.